FORMAL VERIFICATION OF DISTRIBUTED CHECKPOINTING USING EVENT-B

Girish Chandra[1], Raghuraj Suryavanshi[2] and Divakar Yadav[1]

[1]Department of Computer Science & Engineering, Institute of Engineering & Technology, Lucknow, Uttar Pradesh 226021, India
[2]Department of Computer Science & Engineering, Pranveer Singh Institute of Technology, Kanpur, Uttar Pradesh 209305, India

ABSTRACT

The development of complex system makes challenging task for correct software development. Due to faulty specification, software may involve errors. The traditional testing methods are not sufficient to verify the correctness of such complex system. In order to capture correct system requirements and rigorous reasoning about the problems, formal methods are required. Formal methods are mathematical techniques that provide precise specification of problems with their solutions and proof of correctness. In this paper, we have done formal verification of check pointing process in a distributed database system using Event B. Event-B is an event driven formal method which is used to develop formal models of distributed database systems. In a distributed database system, the database is stored at different sites that are connected together through the network. Checkpoint is a recovery point which contains the state information about the site. In order to do recovery of a distributed transaction a global checkpoint number (GCPN) is required. A global checkpoint number decides which transaction will be included for recovery purpose. All transactions whose timestamp are less than global checkpoint number will be marked as before checkpoint transaction (BCPT) and will be considered for recovery purpose. The transactions whose timestamp are greater than GCPN will be marked as after checkpoint transaction (ACPT) and will be part of next global checkpoint number.

KEYWORDS

Formal Methods, Formal Specifications, Formal Verification, Event-B, Distributed Transaction, Check-pointing, Local checkpoint number, Global checkpoint number.

1. INTRODUCTION

A distributed database system is collection of several sites where the database is distributed across different location. Data at any site may be replicated or fragmented either vertically or horizontally. Since there is no system wide global clock or shared memory, sites in these systems communicate the information in form of messages to other sites for successful completion of any global computation [1]. The database present at any site can be accessed through the transactions. A distributed transaction is a user activity which update different data objects located at different sites. A distributed transaction is collection of several sub-transactions. Depending on their requirements these sub-transactions may execute at several sites for reading or updating data objects [2].

Checkpointing is an approach in which state information of each site is periodically saved known as checkpoint or recovery point. In distributed system, every site or a set of sites which are involved in the global computation will take local checkpoints which contain the local information about the sub-transactions at that site. All local checkpoints, one from each site, form a global checkpoint [1], [3]. At the time of recovery, a recovered site resumes its execution from the previous error free consistent global state recorded by the checkpoints of all sites. For recovery purpose, it is necessary that global checkpoint must be consistent. To make global checkpoint consistent it is required that it must not have any local checkpoint which is depend on an event happened after the global checkpoint [1]. The global consistency for distributed database systems must address the issue like which transaction updates will be included in the checkpoints. Therefore, it is required to define unique global checkpoint number or recovery line. For recovery purpose, the updates of only those transactions are included in the checkpoints whose timestamps are less than global checkpoint number and the transactions whose timestamps are larger than global checkpoint number will be considered in next global checkpoint number [4].

There is need to formally verify and ensure the correctness of checkpointing process for distributed database systems. The traditional testing techniques are not suitable to verify the correctness of such systems. It is unfeasible to explore every execution path because size of generated state space is very large. Formal methods are mathematical techniques that use the concepts and ideas from mathematics and formal logic to specify and reason about system properties [5], [6]. It provides a framework which make possible to write specification, analyze and verify the model in a systematic way. The formal methods allow complete analysis of system requirements, design and the behaviour of system including the possibility of faults. The tools also provide automated proofs support for verification of system properties.

In this paper, we have done the formal verification of checkpointing process in a distributed database system using Event-B. Event-B [7], [8], [9], [10], [11], [12], [13] is event driven approach used to develop formal models of distributed database systems. It contains set of variables, constants, and property of model in form of invariant. For ensuring correctness of system invariant properties of model must always be satisfied. The remainder of this paper is organized as follows: section 2 describes the Event B and Rodin, section 3 presents system model and informal description of events, section 4 describes Event-B model of checkpointing process for DDBS, and section 5 concludes the paper.

2. EVENT-B AND RODIN PLATFORM

Event-B model [14], [15], [16], [17], [18] is made of several components of two kinds: contexts and machines [19], [20]. Contexts which represent static part of model contain sets, constants and axioms. Sets may be enumerated or carrier. Axioms are used to describe the properties of those sets and constants. Machines represent behavioural properties of model. It contains the system variables, invariants, theorems, and events of a model. The state of machine is defined through variables. The mathematical constructs such as relations, functions, sets and numbers are represented by variables. The invariants of machine represent constraints that must be applied on machine's variables. During execution of model, the state of machine change from one state to other but the invariants of machine which give properties of those variables should not be violated. All invariants must be satisfied by every state of machine. If violation occur, it means model is not working according to the specification and there is need to modify the machine. The theorem of machine is derived from context and invariants of that machine. The machine can see the context directly or indirectly [21]. Besides its state, a machine contains a number of events which specify how the state may evolve. An event is made up of three elements its name, guards and actions. The guards are the necessary conditions for the event to occur. An event known as initialization event has no guard and it gives initial position of model. For any event, if all guards of event become true then list of actions is performed by that event. The action is a substitution or assignment of new values to variables. There are three kinds of substitutions associated with an event [19] deterministic multiple substitution, non deterministic multiple substitution and empty substitution. An event is triggered and performs list of actions when guards of that event will becomes true. The properties of machine are verified through proof obligations [19], [8].

In our research work, we have used Rodin platform [13], [19], [22]. It is an open extensible tool for specification and verification of Event B. The tool support construction and verification of Event-B models and provides a seamless integration between modelling and proving. It also provides an environment for generation and discharge of proof obligations. It supports incremental development of model whereby verification is done automatically in the background during model development. Therefore, each incremental modification generates a small change to the set of proof obligations. It is embedded by various plugins such as provers, model checkers, UML transformers, proof-obligation generators etc.

3. SYSTEM MODEL

We have considered a group of sites which coordinate each other for taking checkpoints in such manner that the resulting global state is consistent. We have used Lamport's logical clock to assign the timestamp to sites and messages which are involved in the communication. In order to take the consistent global checkpoint in DDBS, it is required to decide which transactions are to be included in the checkpoint. While taking a checkpoint it is also needed that it must not interfere or block the transaction which are already executing at that site. In our model, checkpointing process is initiated by a site known as coordinator site. This site broadcast a timestamped request message to all other sites (participant sites). After receiving the request message, participant site updates its local checkpoint number and send back timestamped reply message to the coordinator. For assigning timestamp to message, each time when a message is sent by any site, it increments its own local checkpoint number by one and that incremented value is assigned to message. At the time of delivery of message the receiving site update its local checkpoint number with the maximum value of timestamp of received message and current checkpoint number. In order to decide which transaction are to be included in the checkpoint, all participant sites must agree upon a special timestamp value known as global checkpoint number. After receiving the reply message from all participant sites, coordinator site compute global checkpoint number. This global checkpoint number is broadcast to all participants so that they can include in their local checkpoint to all those transaction whose timestamp value is less than global checkpoint number. The informal descriptions about the events are as follows:

1. *Broadcasting a request message:* The coordinator site broadcast timestamped request message to all the participant sites. For assigning the timestamp to request message coordinator site increment its own local checkpoint number by one and this incremented value is assigned to request message.

2. *Submission of transaction:* Transaction may be submitted at any site. After submission of a transaction a timestamp value is assigned to it. The current local checkpoint number of site is assigned to transaction as its timestamp.

3. *Delivery of request message:* A request message sent by coordinator site will be delivered to all participant sites. After the delivery of request message, participant site update its local checkpoint number with the timestamp of received request message or its current timestamp value. The value which one is maximum will be assigned to it.

4. *Local marking of transaction:* Each participant site locally marks all those transactions whose timestamp value is less than local checkpoint number of site as a before checkpoint transaction (bcpt) and rest of transaction as after checkpoint transaction (acpt).

5. *Sending of reply message:* After local marking of transaction, each participant site will send timestamped reply message to the coordinator site. In order to assign the timestamp to reply message, participant site increments its local check point number by one and that incremented value is assigned to reply message.

6. *Delivery of reply message:* The coordinator site counts the number of sites which have sent the reply message. Every time when a reply message is delivered to coordinator site it increments its counter value by one. It also makes the entry of timestamp of each received reply message.

7. *Computation and broadcasting of global checkpoint number:* After receiving the reply message from all participant sites, the coordinator site compute global checkpoint number. The global checkpoint number is the maximum value of timestamp of all received reply messages. The coordinator site broadcasts timestamped global checkpoint message to all participant sites.

8. *Receiving of global checkpoint message:* When participant site receives global checkpoint message it updates its local checkpoint number as a timestamp of received global checkpoint message.

9. *Final marking of transaction:* After the delivery of global checkpoint message at all participant sites each participants will have its local checkpoint value equal to global checkpoint number. Finally, all participant sites mark all transactions whose timestamp value is less than global checkpoint number as a before checkpoint transaction and rest of transactions as after checkpoint transaction.

4. EVENT-B MODEL OF CHECKPOINTING IN DISTRIBUTED DATABASE SYSTEM

In the context of model, we have declared sets of *SITE*, *MESSAGE* and *TRANSACTION* as carrier set. The other sets *status*, *type* and *cpstatus* are defined as enumerated set. The set status has values *waiting*, *received_all_replies*, *globalcpnbroadcast*, *idle*. The set type has values *local_cp_request*, *local_cp_reply*, *global_cp_msg* and set *cpstatus* has values *pending*, *globalmark*, *localmark*. The variables and invariants of machine are given in Fig. 1.

The variable *sender* is a partial function from set *MESSAGE* to *SITE*. A mapping of the form *(mms)∈ sender* indicates that message *m* was sent by a site *s*. For recovery purpose, every site maintains local checkpoint number (lcpns) in order to record all the events occur local to it. it is declared as total function from *SITE* to natural number which indicates that each site have local checkpoint number associated with it. For any pair *(ssmn1)∈ lcpns* indicates that site *ss* has local checkpoint number *n1*. Descriptions about other variables are as follows:

MACHINE Checkpointm

VARIABLES

sender, coordinatorstatus, totalrepliedsite, lcpns, tsmsg, trans_at_site trans, timestamp, deliver, bcpt, acpt, messagetype, checkpointstatus tsreplymsg, sentmessages

INVARIANTS

$inv1 : sender \in MESSAGE \nrightarrow SITE$

$inv2 : lcpns \in SITE \rightarrow Natural$

$inv3 : sentmessages \subseteq MESSAGE$

$inv4 : tsmsg \in MESSAGE \rightarrow Natural$

$inv5 : messagetype \in sentmessages \rightarrow type$

$inv6 : deliver \in SITE \leftrightarrow MESSAGE$

$inv7 : tsreplymsg \subseteq Natural$

$inv8 : trans\_at\_site \in SITE \leftrightarrow TRANSACTION$

$inv9 : trans \subseteq TRANSACTION$

$inv10 : timestamp \in trans \rightarrow Natural$

$inv11 : bcpt \in SITE \leftrightarrow trans$

$inv12 : acpt \in SITE \leftrightarrow trans$

$inv13 : coordinatorstatus \in \{coordinator\} \rightarrow status$

$inv14 : checkpointstatus \in SITE \rightarrow cpstatus$

$inv15 : totalrepliedsite \in Natural$

$inv16 : acpt \cap bcpt = \emptyset$

Fig. 1. Variables and Invariants of Machine

(i) The variable *sentmessages* represent set of messages sent by any site. The timestamp of message is formalised using variable *tsmsg*.

(ii) Every message must have a unique type. The variable *messagetype* maps each sent messages to one of its type: *local_cp_request, local_cp_reply, global_cp_msg*.

(iii) The variable *deliver* represents delivery of messages at a site. There are number of messages which are delivered to any site. This requirement is formalized by declaring variable deliver as a relation between *SITE* and *MESSAGE*. A mapping *(ssm mm1)∈ deliver* represents that message *mm1* has been delivered to site *ss*. The message *mm2* can also be relate with same site *ss* due to relation.

(iv) The variable *tsreplymsg* is a set of natural number which represents timestamp of all reply messages.

(v) The set of transactions at any site is represented by a variable *trans_at_site*. Relational image of site *ss* under the relation *trans_at_site* is represented by *trans_at_site*[*{*ss*}*] and it contains all the transactions at site *ss*.

(vi) The variable *trans* is a set of transaction that are submitted to any site.

(vii) The variable *bcpt* represents the set of transaction which are marked as before checkpoint transaction. A mapping *(smtr)∈ bcpt* denotes that site *s* has marked transaction *tr* as before checkpoint transaction. Similarly, the variable *acpt* represents the set of transaction which are marked as after checkpoint transaction.

(viii) Any site can work as a coordinator which coordinates with other site for deciding global check point number. The status of coordinator is represented by the variable *coordinatorstatus*. At any time coordinator may be in the state of *waiting, received_all_replies, globalcpnbroadcast* and *idle*.

(ix) The variable *checkpointstatus* represents checkpoint status of each site. The checkpoint status of site may be one of the following: *pending, globalmark, localmark*.

(x) The variable *totalrepliedsite* is a set of natural number.

The *invariant 16* denotes that any transaction may be either in set *acpt* or *bcpt*. Initially, coordinator status and checkpoint status of each site is set to as *idle* and *pending* respectively. The local checkpoint number of each site and timestamp of each message is set to as 0.

```
Trans_Submit ≙
Any tr,ss Where
grd1 : $tr \in TRANSACTION$
grd2 : $tr \notin trans$
grd3 : $ss \in SITE$
grd4 : $tr \notin trans\_at\_site[\{ss\}]$
grd5 : $checkpointstatus(ss) = pending$
Then
act1 : $trans := trans \cup \{tr\}$
act2 : $timestamp(tr) := lcpns(ss)$
act3 : $lcpns(ss) := lcpns(ss) + 1$
act4 : $trans\_at\_site := trans\_at\_site \cup \{ss \mapsto tr\}$
act5 : $acpt := acpt \cup \{ss \mapsto tr\}$
End
```

Fig. 2. Submission of Transaction

4.1. Submission of Transaction

The event *Trans Submit* models the submission of transaction at any site [Fig. 2]. The guard *grd1* and *grd2* ensure that *tr* is a transaction and it is a fresh transaction respectively. The guard *grd4* is written as: *tr* $\notin$ *trans_at_site[ss]*, it ensures that transaction *tr* is not present at site *ss*. The guard *grd5* specifies that checkpoint status of site *ss* is *pending*. After the submission of transaction *(act1)* a unique time stamp is assigned to it *(act2)*. The action *act2* assigns local checkpoint number of site *ss* to transaction *tr*. Each time when a site assigns a timestamp to transaction, it increments its own timestamp value by one *(act3)*. The action *act4* records that transaction *tr* is present at site *ss*. The action *act5* marks the transaction *tr* at site *ss* as after checkpoint transaction.

```
Remote_Subtran_Submit ≙
ANY tr, ss WHERE
  grd1 : $tr \in trans$
  grd2 : $ss \in SITE$
  grd3 : $tr \notin trans\_at\_site[\{ss\}]$
  grd4 : $finite(\{timestamp(tr), lcpns(ss)+1\})$
  grd5 : $ss \mapsto tr \notin acpt$
  grd6 : $ss \mapsto tr \notin bcpt$
  grd7 : $checkpointstatus(ss)=pending$
THEN
  act1 : $trans\_at\_site := trans\_at\_site \cup \{ss \mapsto tr\}$
  act2 : $lcpns(ss) := max(\{timestamp(tr), lcpns(ss)+1\})$
  act3 : $acpt := acpt \cup \{ss \mapsto tr\}$
END
```

Fig. 3. Submission of Sub-Transaction at Remote Site

4.2. Submission of sub-transaction at remote site

Depending on the requirements, distributed transaction may execute at several sites. This event models the submission of sub-transaction at remote site (Fig. 3). The guard *grd1* ensures that transaction *tr* has been submitted at any site. The guard *grd3* ensures that transaction *tr* is not present at site *ss*. The guard *grd5* and *grd6* ensure that transaction *tr* at site *ss* has neither been marked as after checkpoint transaction (acpt) nor before checkpoint transaction (bcpt). The guard *grd7* specifies that checkpoint status of site *ss* is *pending*. The action *act1* makes the entry of transaction *tr* at site *ss*. When a transaction is submitted at remote site then it updates its knowledge in form of local checkpoint number. The local checkpoint number is updated as *(act2)*:

$$lcpns(ss) := max(\{timestamp(tr), lcpns(ss)+1\})$$

It takes maximum of current local checkpoint number and timestamp of transaction. The value which is maximum is allotted as local checkpoint number of site *ss*. The action *act3* marked the transaction *tr* at site *ss* as after checkpoint transaction (acpt).

4.3. Broadcasting of request message

In order to decide global checkpoint number, coordinator site broadcast request message to all sites [Fig. 4]. The guard *grd1* and *grd2* ensure that site *ss* is *coordinator* and its status is *idle* respectively. The message *mm* has not been sent is ensured by guards *grd3* and *grd4*. Each time when a message is sent by any site, it increments its local checkpoint number by one and this updated timestamp value is assigned to message. The action *act1* increments local checkpoint number of site *ss* by one. The action *act2* assigns timestamp to message *mm*. The action *act3* specifies that message *mm* is sent by site *ss*. The status of coordinator is set to as *waiting* and message *mm* is added to set *sentmessages* through *act4* and *act5* respectively. The type of message *mm* is set to as *local_cp_request* through the action *act6*.

```
Coordinaor_Broadcast  ≙
ANY  ss, mm  WHERE
 grd1  :  ss = coordinator
 grd2  :  coordinatorstatus(ss)= idle
 grd3  :  mm ∈ MESSAGE
 grd4  :  mm ∉ dom(sender)
THEN
 act1  :  lcpns(ss) ≔ lcpns(ss)+1
 act2  :  tsmsg(mm) ≔ lcpns(ss)
 act3  :  sender ≔ sender ∪ {mm ↦ ss}
 act4  :  coordinatorstatus(ss) ≔ waiting
 act5  :  sentmessages ≔ sentmessages ∪ {mm}
 act6  :  messagetype(mm) ≔ local_cp_request
 END
```

Fig. 4. Broadcasting of Request Message

```
Participant_Receive  ≙
 ANY mm, ss  WHERE
 grd1  :  ss ∈ SITE
 grd2  :  ss ≠ coordinator
 grd3  :  mm ∈ sentmessages
 grd4  :  messagetype(mm) = local_cp_request
 grd5  :  mm ∉ deliver[{ss}]
 grd6  :  finite({tsmsg(mm), lcpns(ss)+1})
 grd7  :  checkpointstatus(ss) = pending
THEN
 act1  :  deliver ≔ deliver ∪{ss ↦ mm}
 act2  :  lcpns(ss) ≔ max({tsmsg(mm), lcpns(ss)+1})
 END
```

Fig. 5. Delivery of Request Message

4.4. Delivery of request message at participant site

This event models the delivery of request message at participant site [Fig. 5]. Site *ss* is not coordinator site is ensured through guards *grd1* and *grd2*. The message *mm* has been sent and its type is request message is ensured through guard *grd3* and *grd4* respectively. The guard *grd5* ensures that delivery of message *mm* has not been done at site *ss*. This event makes the delivery of request message *mm* at site *ss (act1)*. At the time of delivery of message site *ss* update its local checkpoint number with the maximum of current local timestamp and timestamp of received request message *(act2)*.

4.5. Local marking of transaction

This event formalizes the marking of transaction on the basis of local checkpoint number of that site [Fig.6]. After receiving the request message from coordinator site, all participant sites mark those transactions as before checkpoint transaction *bcpt* whose timestamp are less than local checkpoint number of that site. The guard *grd2* specifies that site *ss* is not coordinator site. The request message *mm* has been delivered at site *ss* is ensured through guard *grd4* and *grd5*. The guard *grd6* and *grd7* specify that transaction *tr* at site *ss* is marked as after checkpoint transaction (acpt). The timestamp of transaction *tr* is less than local checkpoint number of site *ss* is ensured through *grd8*. The checkpoint status of site *ss* is pending is ensured through guard *grd9*. Due to occurrence of this event transaction *tr* is removed from *acpt* (after checkpoint transaction) set *(act1)* and added in to *bcpt* before checkpoint transaction set *(act2)*.

```
Trans_Marking  ≜
 ANY tr, ss, mm  WHERE
 grd1  :  $tr \in trans$
 grd2  :  $ss \neq coordinator$
 grd3  :  $mm \in sentmessages$
 grd4  :  $messagetype(mm) = local\_cp\_request$
 grd5  :  $ss \mapsto mm \in deliver$
 grd6  :  $tr \in trans\_at\_site[\{ss\}]$
 grd7  :  $ss \mapsto tr \in acpt$
 grd8  :  $timestamp(tr) \leq lcpns(ss)$
 grd9  :  $checkpointstatus(ss) = pending$
 THEN
 act1  :  $acpt := acpt \setminus \{ss \mapsto tr\}$
 act2  :  $bcpt := bcpt \cup \{ss \mapsto tr\}$
 END
```

Fig. 6. Local Marking of Transaction

```
Reply  ≜
ANY ss, mm   WHERE
 grd1  :  $ss \in SITE$
 grd2  :  $ss \neq coordinator$
 grd3  :  $mm \notin dom(sender)$
 grd4  :  $\forall tr \cdot (tr \in trans \land tr \in trans\_at\_site[\{ss\}] \land$
          $timestamp(tr) \leq lcpns(ss) \Longrightarrow ss \mapsto tr \in bcpt)$
 grd5  :  $checkpointstatus(ss) = pending$
 THEN
 act1  :  $checkpointstatus(ss) := localmark$
 act2  :  $sentmessages := sentmessages \cup \{mm\}$
 act3  :  $messagetype(mm) := local\_cp\_reply$
 act4  :  $sender := sender \cup \{mm \mapsto ss\}$
 act5  :  $tsmsg(mm) := lcpns(ss) + 1$
 act6  :  $lcpns(ss) := lcpns(ss) + 1$
 END
```

Fig. 7. Reply to Coordinator Site

4.6. Reply to coordinator site

After marking all transactions whose timestamps are less than local checkpoint number of that site, participant site sends the reply message to the coordinator site [Fig. 7]. The site *ss* is not a coordinator site is ensured through guard grd2. The guard *grd4* specifies that for all transactions at site *ss* whose timestamps are less than local checkpoint number of that site has been marked as *bcpt* (before checkpoint transaction). The guard *grd5* specifies that checkpoint status of site *ss* is *pending*. This event changes the checkpoint status of site *ss* as *localmark (act1)* and sends timestamped reply message *mm (act 2, act3 & act4)*. For assigning timestamp to message *mm* site increments its own timestamp by one and this incremented timestamp value is assigned to reply message *mm (act5)*. The action *act6* updates local checkpoint number of site *ss*.

```
Reply_Delivery  ≙
ANY mm, ss  WHERE
 grd1  :  ss= coordinator
 grd2  :  mm ∈ sentmessages
 grd3  :  messagetype(mm)= local_cp_reply
 grd4  :  mm ∉ deliver[{ss}]
 grd5  :  coordinatorstatus(ss)= waiting
 THEN
 act1  :  deliver ≔ deliver ∪ {ss↦mm}
 act2  :  totalrepliedsite ≔ totalrepliedsite+1
 act3  :  tsreplymsg ≔ tsreplymsg ∪ {tsmsg(mm)}
 END
```

Fig. 8. Delivery of Reply Message at Coordinator Site

4.7. Delivery of reply message at coordinator site

This event models the delivery of reply message at coordinator site (Fig. 8). The guard *grd1* specifies that site *ss* is coordinator site. Delivery of reply message *mm (grd3)* has not been done at coordinator site *ss* is ensured through guard *grd4*. The guard *grd5* ensures that status of coordinator *ss* is *waiting*. When this event triggers, it makes delivery of reply message at coordinator site *(act1)*. The coordinator site also counts the total number of reply messages received from participant site. Each time when a message is delivered, it increments *totalrepliedsite* count by one *(act2)*. The action *act3* makes the entry of timestamp of reply message *mm*.

4.8. Broadcasting global checkpoint number

After the delivery of reply message from all participant sites, coordinator site changes its status from *waiting* to *received_all_replies* [Fig. 9]. The guard *grd2* ensures that coordinator has received reply message from all participant sites. The action *act1* changes status of coordinator.

The event *Broadcast_Gcpn* formalizes broadcasting of global checkpoint number message to all sites [Fig. 9]. The guards *grd1* and *grd4* ensure that site *ss* is coordinator site and it has received reply messages from all sites respectively. After receiving reply message from all participant sites *(grd4)*, coordinator site compute global checkpoint number on the basis of timestamp of received reply message. The global checkpoint number is the maximum value of timestamp of reply message *(grd6)*. Due to occurrence of this event, coordinator site *ss* broadcast timestamped global checkpoint message *mm (act1,act2,act3)*. The action *act4* specifies that global checkpoint number *globalcpn* is assigned as timestamp of message *mm*. The action *act5* changes the status of coordinator site as *globalcpnbroadcast*.

```
Change_Co-ordinator_Status  $\triangleq$
ANY ss  WHERE
 grd1  :  $ss = coordinator$
 grd2  :  $totalrepliedsite = card(SITE) - 1$
 grd3  :  $coordinatorstatus(ss) = waiting$
 THEN
 act1  :  $coordinatorstatus(ss) := received\_all\_replies$
 END

Broadcast_Gcpn  $\triangleq$
ANY ss, globalcpn, mm  WHERE
 grd1  :  $ss = coordinator$
 grd2  :  $tsreplymsg \neq \varnothing$
 grd3  :  $mm \notin dom(sender)$
 grd4  :  $coordinatorstatus(ss) = received\_all\_replies$
 grd5  :  $finite(tsreplymsg)$
 grd6  :  $globalcpn = max(tsreplymsg)$
 THEN
 act1  :  $sender := sender \cup \{mm \mapsto ss\}$
 act2  :  $sentmessages := sentmessages \cup \{mm\}$
 act3  :  $messagetype(mm) := global\_cp\_msg$
 act4  :  $tsmsg(mm) := globalcpn$
 act5  :  $coordinatorstatus(ss) := globalcpnbroadcast$
 END
```

Fig. 9. Broadcasting Global Checkpoint Number

4.9. Delivery of global checkpoint number message

This event models the delivery of global checkpoint message at participant site [Fig. 10]. The global checkpoint message *mm (grd4)* has been sent by coordinator site *s* is ensured through guard *grd5*. The guard *grd6* ensures that delivery of message *mm* has not been done at site *ss*. The guard *grd7* specifies that for all transaction *tr* at site *ss* if they are marked as *bcpt* (before checkpoint transaction) then timestamp of transaction will be less than timestamp of global checkpoint message *mm*. Due to occurrence of this event, delivery of message *mm* is done at site *ss (act1)*. The site *ss* also updates its knowledge in form of local checkpoint number with the timestamp of globalcheckpoint message *mm (act2)*.

4.10. Final marking of transaction

This event formalizes the final marking of transaction [Fig. 11]. It marks all transactions as before checkpoint transactions *bcpt* whose timestamps are less than timestamp of global checkpoint message. The message *mm* is global checkpoint message is ensured through guard *grd4*. The delivery of message *mm* has been done at site *ss* is ensured through guard *grd6*. The transaction *tr* is not marked as before checkpoint transaction is ensured through guard *grd7* and its timestamp is less than local checkpoint number of site *ss* is ensured through guard *grd8*. This event marks the transaction *tr* as before checkpoint transaction by adding it to *bcpt* (before checkpoint transaction) set.

```
GCPN_Message_Receive ≙
ANY ss, mm, s WHERE
 grd1 : s= coordinator
 grd2 : ss∈SITE
 grd3 : mm∈sentmessages
 grd4 : messagetype(mm)=global_cp_msg
 grd5 : mm↦s ∈sender
 grd6 : ss↦mm ∉ deliver
 grd7 : ∀tr·tr∈ trans ∧ tr∈ trans_at_site[{ss}]
        ∧ ss↦tr∈ bcpt⟹ timestamp(tr)≤ tsmsg(mm)
 grd8 : checkpointstatus(ss)=localmark
 THEN
 act1 : deliver≔deliver ∪{ss↦mm}
 act2 : lcpns(ss)≔tsmsg(mm)
 END
```

Fig. 10. Delivery of Global Checkpoint Number Message

```
 Final_Trans_Marking ≙
ANY tr, ss, mm WHERE
 grd1 : tr∈trans
 grd2 : ss↦tr∈acpt
 grd3 : mm∈sentmessages
 grd4 : messagetype(mm) = global_cp_msg
 grd5 : ss ≠ coordinator
 grd6 : ss↦mm ∈deliver
 grd7 : ss↦tr∉bcpt
 grd8 : timestamp(tr) ≤ lcpns(ss)
 THEN
 act1 : acpt≔acpt \ {ss↦tr}
 act2 : bcpt≔bcpt∪{ss↦tr}
 END
```

Fig. 11. Final Marking of Transaction

4.11. Update checkpoint status of participant

After marking all transactions whose timestamps are less than local checkpoint number of that site (which is now global checkpoint number) as before checkpoint transaction, checkpoint status of that site must be marked as *globalmark* [Fig. 12]. The global checkpoint message *mm (grd4)* has been received by site *ss* is ensured through *grd5*. The guard *grd6* ensures that for all transactions *tr* whose timestamp are less than local checkpoint number of that site *lcpns(ss)* then that transaction must be present in before checkpoint transaction set *bcpt*.

Participant_Change_Status $\triangleq$
ANY *ss, mm* **WHERE**

```
grd1 : $ss \neq coordinator$
grd2 : $checkpointstatus(ss) = localmark$
grd3 : $mm \in sentmessages$
grd4 : $messagetype(mm) = global\_cp\_msg$
grd5 : $ss \mapsto mm \in deliver$
grd6 : $\forall tr \cdot tr \in trans \land tr \in trans\_at\_site[\{ss\}] \land$
       $timestamp(tr) \leq lcpns(ss) \Longrightarrow ss \mapsto tr \in bcpt$
THEN
act1 : $checkpointstatus(ss) := globalmark$
END
```

Fig. 12. Update Checkpoint Status of Participant

5. Conclusions

Modern distributed systems are very difficult to develop and reason about. There is need to formally verify and ensure the correctness of distributed systems and algorithms. During last few years the research in the field of formal methods has been grown up and has reported significant success in the development of describing and analyzing the complex systems in formal languages. In distributed database systems, formal methods take important role for ensuring the correctness. In this paper, we have done formal verification of distributed checkpointing for the recovery of distributed transaction. A distributed transaction may be divided in several subtransactions which may execute at different sites to accomplish global computation. Each site maintains its state information in form of local checkpoint. In order to recover a distributed transaction it is required to decide a global recovery line or global checkpoint number which must include all sub-transactions of a transaction whose timestamp are less than it. For others whose timestamp are larger will be included in next checkpoint.

In our model, we have presented formal model of global checkpointing in distributed environment using Event-B. Event-B is a formal method which is used to verify distributed algorithms. It rigorously verifies all the properties of a model by discharging proof obligations generated by it. Our model formally specifies the computation of global checkpoint number GCPN and verifies that GCPN only includes those transactions whose timestamps are lesser than it. More specifically, it includes all sub-transactions which are submitted before the global checkpoint number. In order to ensure correctness of our model, we have added following invariants:

Inv17: $\exists tr,si,sj.\ tr \in trans \land si \in SITE \land sj \in SITE \land tr \in trans\ at\ site[si] \Rightarrow tr \in trans\ at\ site[sj]$
Inv18: $\forall ss,tr.\ tr \in trans \land ss \in SITE \land tr \in trans\ at\ site[ss] \land ss \mapsto tr \in bcpt \Rightarrow timestamp(tr) < lcpns(ss)$
Inv19: $\forall ss,tr.\ tr \in trans \land ss \in SITE \land ss \neq coordinator \land tr \in\ trans\ at\ site[ss] \land checkpointstatus(ss) = globalmark \land timestamp(tr) \leq lcpns(ss) \Rightarrow ss \mapsto tr \notin acpt$

The invariant 17 verifies the submission of transaction under distributed environment. The distributed computation involves execution of transaction at several sites. This invariant ensures that if transaction *tr* is present at site *Si* then it may also be present at other site *Sj*.
The invariant 18 ensures the correctness of our model by ensuring that there will be no transaction in set *bcpt* whose timestamp are larger than local checkpoint number of that site. It verifies that for all transactions *tr* if site *ss* has marked this transaction as before checkpoint transaction then timestamp of transaction *tr* will be less than local checkpoint number of site *ss*.

The invariant 19 verifies that if the site has completed its final marking then all transactions whose timestamps are less than local checkpoint number of that site will not be present in set *acpt* (after checkpoint transaction). We have used Rodin tool for writing Event-B specifications. The model generates 106 proof obligations out of which 51 are discharged automatically by the prover of tool while 55 proof obligations are discharged manually. The proof obligations generated by model give the rigorous reasoning about the design of model. During execution of model all invariants are preserved which ensures that model is correct. In future, we aim to use vector timestamp in place of scalar timestamp.

REFERENCES

[1] M.Singhal, N.G.Shivratri: Advanced Concepts in Operating Systems. Tata Mc-GrawHill Book Company, India (2005).

[2] A.Helal, A.Heddya and B. Bhargava: Replication Techniques in Distributed System. Kluwener Academic Publishers (1997).

[3] R.Koo,S. Toueg:Checkpointing and Rollback-Recovery for Distributed Systems. In: IEEE Transactions on Software Engineering, vol. 13, no. 1, pp. 23-31, (1987).

[4] S.H.Son, A.K. Agrawala: Distributed Checkpointing for Globally Consistent States of Databases.In: IEEE Transactions on Software Engineering, vol. 15, no. 10, pp. 1157-1167, (1989).

[5] M.G.Hinchey, JP. Bowen and R.L. Glass: Formal methods: Point-counterpoint. Computer, 29(4):1819, 1996.

[6] C.Jones, D. Jackson and J. Wing: Formal methods light. Computer, 29(4):2022, 1996.

[7] R.Banach: Retrenchment for Event-B: UseCase-wise development and Rodin integration. Formal Aspects of Computing, 23, pp. 113131, (2011).

[8] S.Hallerstede: On the purpose of Event-B proof obligations. Formal Aspects of Computing, 23: pp. 133150, (2011).

[9] S.Hallerstede and M. Leuschel: Experiments in program verification using Event-B. Formal Aspects of Computing, 24: pp. 97125, (2012)

[10] D.Basin, A. Furst, T.S. Hoang, K. Miyazaki, and N. Sato: Abstract Data Types in Event-B - An Application of Generic Instantiation. CoRR, 2012.

[11] J-R.Abrial. From Z to B and then Event-B: Assigning Proofs to Meaningful Programs. In E.B. Johnsen and L. Petre, editors, IFM, volume 7940 of Lecture Notes in Computer Science, pages 115. Springer, 2013.

[12] M.Butler and I.Maamria: Practical theory extension in Event-B. In Zhiming Liu, Jim Woodcock, and Huibiao Zhu, editors, Theories of Programming and Formal Methods, volume 8051 of Lecture Notes in Computer Science, pages 6781. Springer, 2013.

[13] J.R.Abrial : A system development process with Event-B and the Rodin platform. In: Lecture Notes In Computer Science 4789, Springer, pp.1-3, (2007).

[14] R.Suryavanshi, D.Yadav: Formal Development of Byzantine Immune Total Order Broadcast System using Event-B. In: ICDEM 2010, Andres,F., Kannan, R. (eds.) LNCS, Vol. 6411, Springer, pp.317-324, (2010).

[15] J-R.Abrial. Modeling in Event-B: System and Software Engineering. Cambridge University Press, 2010.

[16] T.S.Hoang, Proving almost-certain convergence properties using Event-B, Tech. Rep. 768, Department of Computer Science, ETH Zurich, http://www.inf.ethz.ch/research/disstechreps/techreports (Jul. 2012).

[17] T.Hoang, H. Kuruma, D.Basin, J.R.Abrial:Developing topology discovery in Event- B, Science of Computer Programming 74 (11-12) (2009) 879899.
[18] T.S.Hoang, J.R. Abrial: Reasoning about liveness properties in Event-B,in: S. Qin, Z. Qiu (Eds.), International Conference on Formal Engineering Methods 2011, Vol. 6991 of Lecture Notes in Computer Science, Springer-Verlag, 2011, pp. 456471.
[19] C. Metayer, J.R. Abrial, L.Voison: Event-B language. RODIN deliverables 3.2, http://rodin.cs.ncl.ac.uk/deliverables/D7.pdf, (2005).
[20] Kriangsak Damchoom, Michael Butler, J-R Abrial: Modelling and Proof of a Tree- Structured File System in Event-B and Rodin. In Shaoying Liu, Tom Maibaum and Keijiro Arak, editors, ICFEM-2008, Formal Methods and Software Engineering, Lecture Notes in Computer Science, Volume 5256, pages 25-44, Springer, 2008.
[21] J-R. Abrial and S. Hallerstede: Refinement, Decomposition, and Instantiation of Discrete Models: Application to Event-B. Fundam. Inform., 77(1-2):128, 2007.
[22] J-R. Abrial, M. Butler, S. Hallerstede, T.S. Hoang, F. Mehta, and L. Voisin. Rodin: an open toolset for modelling and reasoning in Event-B. International Journal on Software Tools for Technology Transfer (STTT), 12(6):447466, 2010.

AUTHORS

Girish Chandra is an Associate Professor in Department of Computer Science and Engineering at Institute of Engineering Technology, Lucknow.. He has received M.Tech from IIT Kanpur. He is doing Ph.D. from Uttar Pradesh Technical University, Lucknow. He has presented several international papers in IITs and other universities. His research interest includes Cryptography, Formal Verification and Distributed Systems.

Raghuraj Suryavanshi is working as Assistant Professor in Computer Science and Engineering at Pranveer Singh Institute of Engineering & Technology Kanpur. He has completed Ph.D. from Uttar Pradesh Technical University, Lucknow. He has received Teacher Fellowship award from Uttar Pradesh Technical University. He has presented several international papers in India and abroad. His research interests are formal verification and validation of critical properties of distributed database systems.

Prof. Divakar Yadav is working as Director of Dr. Bhim Rao Ambedkar Engineering College of Information Technology, Banda. He obtained Ph.D. in Computer Science from University of Southampton, U.K under Commonwealth Scholarship & Fellowship Plan, U.K. earlier, he obtained M.Tech in Computer Science from Indian Institute of Technology, Kharagpur. Dr. Yadav possesses more than 25 years of experience in academics/research in India and abroad. He is Professor of Computer Science and Engineering at Institute of Engineering and Technology, Lucknow His primary research interests are in formal methods, refinement of distributed systems using Event-B.