Abstract—The implementation of load balancing is a challenging task for correct system functioning. Load balancing is the process of distributing the load among all the available nodes in a proper way. We present the concept of split point where underloaded node is not absorb all the load of overloaded node, rather it will be distributed among several underloaded nodes. In a distributed system, the directory is maintained at all the information of all the nodes. Another term defined in this algorithm is threshold. Threshold limit defines the capacity of each node. Ideally load is not above the define threshold limit, if overloaded condition arise then load will be migrated from one node to another. In this paper we implement the formal specification of split point load balancing algorithm under Rodin platform on event B. Formal methods are mathematically based tools for the verification of software system and hardware system. Event B is formal method for modeling and mathematical reasoning about the system that may consist of physical components and software modeling.

Keywords—Load Balancing, Threshold, Event B, Formal Specification, Formal Methods, Formal Verification

I. INTRODUCTION

Load balancing [1] is one of the important concept in distributed system. Our system is the collection of nodes where all the nodes defined by some load. Load balancing improves the resources utility, maximize the throughput, minimize response time and minimize the overhead in entire system [2][3][4][5][6]. Split point is one of the best load balancing algorithm [2], where load will be transferred to several underloaded nodes. In this paper we prove the algorithm using formal methods and verification process.

Defining the system model, if any new request is processed in node then we increase the load value and according the load value we define node status under these categories:

A) Underloaded node
B) Overloaded node

If load of the node is greater than threshold value then we define the node is overloaded. If load value less than threshold value then we can say that node is underloaded. After defining the type of node if load is overloaded then we transfer the load from overloaded node to underloaded node. In this structure we maintain a directory of each node with corresponding load. To overcome the load of overloaded nodes, we broadcast the message from overloaded nodes to underloaded nodes. After broadcasting the request message, underloaded nodes deliver the request and reply with load value and node information. Overloaded nodes receive the information with corresponding load and save the information in own directory. After saving the nodes and corresponding load information, overloaded node search the suitable node for sending the load. One important point is there load will not be transferred to that node which has recently received the load from other node.

If any overloaded node which is not suitable to transfer the load because of load value of overloaded node is excess and none of the underloaded nodes are taking the load in one node so we shall split the load in overloaded node according to the information of directory. Means we select the least load of underloaded nodes and according the underloaded load value we split the load in overloaded node. In their one important point is this none of the node' load value greater than threshold limit.

After selection of nodes, we define node movement mechanism. In this mechanism a node is overloaded due to many reasons. First reason is a node gets overloaded due to high popularity of more than one of its node. Second, a node gets overloaded because of high amount of loads put on it while none of them is highly popular. In this condition we select a node and transfer the load from overloaded to under loaded node. During the transfer of the load, the load value of overloaded node is decrease and load value of underloaded node is increase according to corresponding threshold value.

In this paper, we present a formal model of verification and validation of a distributed load balancing (how can we balance the load and transfer the load in appropriate node). According the algorithm we convert the each condition into set theory which have defined under the eclipse based RODIN platform to the extension of Event B [7] language. We define every single module and satisfy the assumption of proof obligation (PO) methods in the formal based approach. Rodin tool is generated the proof tree and statistics of context and machine. It also generate the proof information of every event and invariants. The Event B formalism is a part of the B method [7][8]. It is a state based formal approach to promote the construct of correct paradigm of algorithm and formal verification by proving the theorem [9].

The remainder of this paper is organized as follows: Section 2 briefly outline about the Event B and Rodin platform, Section 3 describes system model and informal description about events, Section 4 presents Event−B Model of Load balancing for distributed system. Section 5 defines the result analysis of proposed model and Section 6 concludes the paper.
II. EVENT B AND RODIN PLATFORM

Event B is a modeling language to provide a transition methods of any algorithm. Event B has been simplified the general notation into discrete mathematical expression. A general notation called event. Every event represents a different modules under the where clause and according to where clause, act module is implemented. Additionally, we read some modeling structure that are important to understand the concept of Rodin platform and provide the verification of any algorithm. Event B model [7] [8] [9] [10][11] is divided into two components - Context and Machine.

A) Context: Context specify the static part of model. It contains sets, constant, axioms and theorems. Sets may be enumerated or carrier sets. The properties of constant and sets describe under the axioms section. Axiom is set of constant identifier and these identifier describe the properties under the axioms section. The context may be seen by the machine directly or indirectly.

B) Machine: Machine represents the dynamic part of model which have contained the behavior model. Machine contains refine, sees, variables, invariants, variants and events. Under the machine, we create a local variables and define the operations in event section which start with when and begin keyword. Under the guard section we write a condition, if all the guards is true then act module will be implemented. In guard all the conditions are define according to our model. If one of the guard is false then act module will not be correctly implemented.

Invariant term defines under the machine model, which is definition of the variable. Variable contain three modules like variable keyword, invariant and initialization part. Variables are constrained by invariants and these invariants are to be safe whereas change the value of variables. All the module should be clearly define. A machine can refines by the other machine, but each machine refines only one machine.

For making the consistent model, invariants should be formally prove. Machine sees the context part, if we do not define the context variables in machine model then we cannot conclude static part in model. The guard states the essential condition under an event and action (act) describe on behalf of the guard when the successfully events occur.

Event B is a process refinement, state based model. Refinement is a technique of changing some values of model using different state model that proofs the correctness of model. For a correct state of refinement, every possible execution of refined machines correspond to execution step of its abstract machine. Verification of a model as well as correctness of refinement steps should be discharging by proof obligation. The Rodin platform [8] is support of event B that provides the environment of automatically generated proofs and manages an automated proves that is automatic discharging the goal of the model. The level of automation should be high to make a realistic model.

There is some B notation frequently used in our model. Let P and Q be two sets, then the relational constructor \( \leftrightarrow \) defines the set between P and Q.

\[ P \leftrightarrow Q = \text{Power}(A*B) \]

Where \( * \) is Cartesian product of P and Q. These relation written as \( p \leftrightarrow q \) where \( p \in P \) and \( q \in Q \)

III. SYSTEM MODEL AND INFORMAL DESCRIPTION OF EVENTS

In this section, we shall consider an informal approach about the split point load balancing under distributed environment. They have a set of nodes where every node maintain a dictionary. In dictionary they contain the information about requesting node and their respective load. Request queue also contain the message of overloaded node with respective load. In a load balancing system, every node maintains a status of node. Node status defines in two categories like overloaded node and under loaded node. Assume one of the node’s load is overloaded then their respective dictionary search the under loaded nodes. During the searching process overloaded node broadcast the message and received by the every node except sending node. The node whose load value is less than threshold, send reply to sender node. Overloaded node selects one of the best nodes in underloaded nodes and transfer their respective load. Each time when a message is send by any node their respective dictionaries will be updated. During the load transfer if one receiver is not able to received whole value of sender load then the load will be divided and transferred to two or more underloaded node. The main objective is that load value of receiver should also not exceed the threshold limit.

The informal description of events are as follows:

A) Load submission: When a load is submitted to node, its load value is increased by one. After increasing the load value, we compare the load value with threshold value. If the load value is greater than threshold value then we set the node status is overloaded and if the load value is less than threshold value then we set the status of node is underloaded. Overloaded nodes search the underloaded node for achieving to balance the load in the system.

B) Broadcasting and delivery of a request message: The overloaded node broadcast the request message to all the participating nodes. For transferring the load from overloaded node to underloaded nodes, contain the information of each node is necessary. The request message delivers in underloaded nodes from overloaded node.

C) Reply to the requesting node: After receiving request message, receiver (participant) nodes send the reply message with the node and load information to the overloaded node.

D) Receiving of reply message to requesting node: The requesting node receives the reply message from underloaded nodes (receiver) and makes an entry for each reply message in dictionary. The requesting node counts the total number of replied nodes and compare the load value of each nodes.
On the basis of different load values in overloaded node, the node selects the most appropriate node for transferring the load.

E) Split point sender and decrement the load: Whenever underloaded node does not capacity to receive whole load of sender (which is overloaded node), the overloaded node splits the load and choose two or more underloaded nodes to transfer the load. Split the load in sender node corresponding the load value of underloaded node. After splitting the load value node can transfer the load in various underloaded nodes and decrease the value from overloaded node.

F) Split point receiver and increment the load: After receiving the load from overloaded node, value of nodes increase the load in underloaded nodes. After all the overloaded node turn into normal node, load transfer is complete.

We have read many algorithms which are implemented under the Event B model for verification and provide the proof obligation [11][12][13][14][15].

IV. EVENT B MODEL OF LOAD BALANCING IN DISTRIBUTED SYSTEM

Event B model contains a context and a machine. Machine contains various invariants and events. In context we declare the sets of MESSAGE & NODE as carrier set. The other term is set status, capacity, and type, which are define as enumerated set. In the set status we have declared the two type of load like under load & overload. The set status define the status of node. The set type contain a constant like type of load like under load & overload. The set status define the type of load like under load & overload. The set status define the type of message whenever it is request or reply. Invariant defined as “sender” variable is a partial function from MESSAGE to NODE. A mapping of form \( mm \rightarrow nn \) \( \in \) sender indicates that message mm is send by node nn. Define another variable is loadmessage. Loadmessage manages a load by natural number (N). Invariant defines by (loadmessage \( \in \) MESSAGE \( \rightarrow \) N) means message contain the load which have natural number. The variable reponse promising is subset of NODE set and it contains only those nodes which are overloaded.

The variables, initialization and invariants terms of machine are defined as follows:

Variables: sender, loadmessage, deliver, nodestatus, loadvalue, replymsgsend, replymsgrcd, msgrcd, dir, messagenodevalue, msgagertype, replynode, trans, transferload

Initialization: We declare all the initial values according to the modeling. Most of them declare with \( \emptyset \) value. But some values can’t be \( \emptyset \). The initial value of node status is under loaded and initial value of load value is 0 which is a natural number.

<table>
<thead>
<tr>
<th>INVARIANTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>inv1: sender ( \in ) MESSAGE ( \rightarrow ) NODE</td>
</tr>
<tr>
<td>inv2: deliver ( \in ) NODE ( \rightarrow ) MESSAGE</td>
</tr>
<tr>
<td>inv3: nodestatus ( \in ) NODE ( \rightarrow ) status</td>
</tr>
<tr>
<td>inv4: loadvalue ( \in ) NODE ( \rightarrow ) ( \mathbb{N} )</td>
</tr>
<tr>
<td>inv5: replymsgsend ( \in ) (MESSAGE ( \rightarrow ) MESSAGE) ( \rightarrow ) NODE</td>
</tr>
<tr>
<td>inv6: replymsgrcd ( \in ) NODE ( \rightarrow ) (MESSAGE ( \rightarrow ) MESSAGE)</td>
</tr>
<tr>
<td>inv7: msgrcd ( \subseteq ) MESSAGE</td>
</tr>
<tr>
<td>inv8: dir ( \in ) NODE ( \rightarrow ) (MESSAGE ( \rightarrow ) ( \mathbb{N} ))</td>
</tr>
<tr>
<td>inv9: msgagertype ( \in ) msgrcd ( \rightarrow ) type</td>
</tr>
<tr>
<td>inv10: messagenodevalue ( \in ) MESSAGE ( \rightarrow ) ( \mathbb{N} )</td>
</tr>
<tr>
<td>inv11: replynode ( \in ) NODE ( \rightarrow ) (MESSAGE ( \rightarrow ) ( \mathbb{N} ))</td>
</tr>
<tr>
<td>inv12: trans ( \subseteq ) NODE</td>
</tr>
<tr>
<td>inv13: transferload ( \rightarrow ) trans ( \rightarrow ) loadtransfer</td>
</tr>
<tr>
<td>inv14: loadmessage ( \in ) (MESSAGE ( \rightarrow ) ( \mathbb{N} ))</td>
</tr>
</tbody>
</table>

Fig 1: Invariants of load balancing algorithm

All those invariants (see figure 1) described below:

a) The variable sender send the message from one node to another (inv1). The invariant inv(2) is defined as follows:

\[ \text{deliver} \in \text{NODE} \rightarrow \text{MESSAGE} \]

The variable deliver represents delivery of message at a node. A mapping of form \( (mn \rightarrow mm) \in deliver \) represents that a node \( mn \) has delivered the message \( mm \) to participating nodes.

b) The variable nodestatus defines the status of node, either load is overloaded or underloaded. (Inv3)

c) The invariant inv(4) represents the variable of loadvalue. It defines the value of load in a node. Load value represent by a natural number.

d) The Invariant inv(5) represents replymsgsend as follows:

\[ \text{replymsgsend} \in (\text{MESSAGE} \rightarrow \text{MESSAGE}) \rightarrow \text{NODE} \]

The invariant of replymsgsend defines the sending of message from participant node (underloaded node) to corresponding node (overloaded node).

A mapping of form \( ((mm \rightarrow m)) \rightarrow m) \in \text{replymsgsend} \) represents that an underloaded node \( n \) send the reply message \( m \) for corresponding request message \( mm \).

e) The variable replymsgrcd represents the receiving of message from participating nodes (receiver) to corresponding node (sender). (Inv 6)

\[ \text{replymsgrcd} \in \text{NODE} \rightarrow (\text{MESSAGE} \rightarrow \text{MESSAGE}) \]

A mapping of the form is as follows:

\( (mn \rightarrow ((mm \rightarrow m))) \in \text{replymsgrcd} \)
Split Point Load Balancing Algorithm Based on Event B

An overloaded node nn has received the reply message m from corresponding request message mm.

f) The invariant Inv (7) msgsend specifies set of send and receive messages.

The invariant Inv (8) msgsend specifies set of send and receive messages.

g) The variable dir represents the directory structure. It store the information of load value of node. Directory exchange the load information between the overloaded node and underloaded node. (Inv 8)

h) The variable messagetype specify the type of message. Message type is either request or reply. (Inv 9)

i) The variable messagenodevalue carry the information of load value of node. (Inv 10)

j) The variable replynode represent only those nodes which have accept the load from the underloaded nodes and reply to overloaded node. (Inv 11)

k) The invariant (inv 12) represents the trans variable, which is belong to NODE set. The invariant (inv 13) of transferload represent under the axioms in context section. The transfer load is either completed or inprogress.

l) The variable loadmessage represent only these messages which are reply to overloaded nodes. (Inv 14)

4.1 Events

In last section we define variable and invariants of load balancing machine. Now we discuss the events in load balancing process. Events are also called the operation or transitions. Event has no parameter, user is define as own event according to machine requirement. There is no access to state variable. At most one event is seen by system at a time. Different guard in each event makes different operation perform by machine and generate a proof obligation. Any invariant I (v) where v is variable, which is initialized with initial condition and which is safe by events or transitions in the list of events.

There are some events in load balancing process to provide the correctness and efficiency our model.

A) Increase the load value: If any fresh node is processed in the system then we have increased the load value by one. (See figure 2)

```
loadvalueincrease Δ ANY nn, ld
WHERE
grd1 : nn ∈ NODE
grd2 : ld=loadvalue(nn)
grd3 : ld∈ N
THEN
act1 : loadvalue(nn)≔loadvalue(nn)+1
END
```

Fig 2: Event of Load value increase

The guard grd1 ensures that node nn is node and guard grd2 defines its loadvalue ld of the node nn. If the guard is true then action act 1 increases the value by one.

B) Identify the under loaded node: To identification of the node as under loaded, we compare the value of threshold. If the load value of node is less than threshold then we define the status of node is under loaded. (See figure 3)

```
underload Δ
ANY nn
WHERE
grd1 : nn∈NODE
grd2 : loadvalue(nn)<value_threshold
THEN
act1 : nodestatus(nn)≔underloaded
END
```

Fig 3: Event of under loaded node

System compare the load value of node nn with the value of threshold in guard grd2. If the guard is true then the action is perform and status of node is set to the undeloaded (act1).

B) Identify the overloaded node: The event of overload node shows in figure 4. To identify the node status whether it is underloade or overloaded, we compare the value of threshold. If the value of threshold is greater than load of the node, then the node is overloaded.

```
overload Δ
ANY nn
WHERE
grd1 : nn ∈NODE
grd2 : loadvalue(nn) ≥ value_threshold
grd3 : loadvalue(nn) ∈ N
THEN
act1 : nodestatus(nn)≔overloaded
END
```

Fig 4: Event of overloaded node

Event of overloaded node defines the load value of node nn is greater than or equal to threshold value in guard grd2. Load value of node mn is natural number (grd 3). We set the status of node mn is overloaded in action act1.

C) Broadcast the message: After identify the status of nodes if any node is overloaded then we broadcast the request and send the message to every participating nodes. All the participating nodes receive the request message. Figure 5 defines the event of broadcast.
broadcast \(\triangleq\) ANY \(nn, mm\)
WHERE
\[
\begin{align*}
grd1 & : nn \in \text{NODE} \\
grd2 & : mm \in \text{MESSAGE} \\
grd3 & : mm \in \text{dom}(\text{sender}) \\
grd4 & : \text{nodestatus}(mm) = \text{overloaded} \\
\end{align*}
\]
THEN
\[
\begin{align*}
act1 & : \text{sender} := \text{sender} \cup \{mm \leftarrow nn\} \\
act2 & : \text{msgsend} := \text{msgsend} \cup \{mm\} \\
act3 & : \text{messagetype}(mm) = \text{request} \\
\end{align*}
\]
END

Fig 5: Broadcast of message

In broadcast event we define the variable \(nn\) and \(mm\) where \(nn\) is belong to node and \(mm\) is belong to sending message \((grd\ 1, 2)\). We set the status of node is overloaded \((grd\ 4)\). We send the newly message that not occurring in domain of sender list \((grd\ 3)\) in receiving node. We perform the action clause where message \(mm\) is broadcast to all the system send by node \(nn\) updated in action \(act1\). Message \(mm\) is reflect in msgsend list represent in \(act2\). Type of message \(mm\) is request \((act3)\).

E) Delivery of request message: When the overloaded node send a request to underloaded node then all nodes receive the request. Message must be fresh in the receiving node. Delivery event is shown in figure 6.

deliver request \(\triangleq\) ANY \(nn, mm, n, ld\)
WHERE
\[
\begin{align*}
grd1 & : nn \in \text{NODE} \\
grd2 & : mm \in \text{MESSAGE} \\
grd3 & : mm \in \text{dom}(\text{messagetype}) \\
grd4 & : \text{messagetype}(mm) = \text{request} \\
grd5 & : (mm \leftarrow nn) \in \text{sender} \\
grd6 & : n \in \text{NODE} \\
grd7 & : (n \leftarrow mm) \in \text{deliver} \\
grd8 & : ld = \text{loadvalue}(nn) \\
\end{align*}
\]
THEN
\[
\begin{align*}
act1 & : \text{deliver} := \text{deliver} \cup \{n \leftarrow mm\} \\
\end{align*}
\]
END

Fig 6: Event B model of request delivery

In request deliver event model, the delivery of request message at participating node. The guard \(grd5\) ensures that message \(mm\) is send by node \(nn\) and message is in the sender list. The guard \(grd7\) defines, delivery of message \(mm\) in node \(n\) does not exist. We deliver the message \(mm\) in node \(n\) successfully in action \(act1\).

relymsend \(\triangleq\) ANY \(n, nn, mm, m, ld, nn\)
WHERE
\[
\begin{align*}
grd1 & : nn \in \text{NODE} \\
grd2 & : mm \in \text{MESSAGE} \\
grd3 & : mm \in \text{msgsend} \\
grd4 & : mm \in \text{dom}(\text{sender}) \\
grd5 & : (n \leftarrow mm) \in \text{deliver} \\
grd6 & : ld = \text{loadvalue}(n) \\
grd7 & : \text{messagetype}(mm) = \text{request} \\
grd8 & : (mm \leftarrow n) \in \text{sender} \\
grd9 & : m \in \text{MESSAGE} \\
grd10 & : n \in \text{msgsend} \\
grd11 & : n \in \text{dom}(\text{sender}) \\
grd12 & : [mm \leftarrow m] \in \text{dom}(\text{relymsend}) \\
grd13 & : mm \in \text{NODE} \\
grd14 & : [m \leftarrow ld] \in \text{dir}[\{nn\}] \\
grd15 & : \text{loadvalue}(n) \leq \text{value\_threshold} \\
\end{align*}
\]
THEN
\[
\begin{align*}
act1 & : \text{msgsend} := \text{msgsend} \cup \{m\} \\
act2 & : \text{messagetype}(m) = \text{reply} \\
act3 & : \text{sender} := \text{sender} \cup \{m \leftarrow n\} \\
act4 & : \text{relymsend} := \text{relymsend} \cup \\
& \quad \{(m \leftarrow mm) \leftarrow n\} \\
act5 & : \text{messagesend}(m) := \text{loadvalue}(n) \\
\end{align*}
\]
END

Fig 7: Reply of request message

F) Reply of request message: After receiving the delivery message by participating nodes, underloaded nodes send a reply message to coordinator node (overloaded node). Coordinator node receive a reply message from underloaded nodes and update the directory with nodes and respective load information. (See figure 7)

In relymsend event shows that message \(mm\) is in the domain of sender list in guard \(grd\ 4\) and type of message is request define in guard \(grd\ 7\). Message \(m\) is a fresh message and not deliver yet \((grd\ 9 \text{ and } 10)\). Message \(m\) is not in domain of sender \((grd\ 11)\). Reply message does not send in domain of sender \((grd\ 12)\) and directory of node \(n\) does not update with load value \(ld\) by sending message \(m\) \((grd\ 14)\). Nodes send the reply message, the load value of nodes are less than threshold limit \((grd\ 15)\). If all the guard is true, then we perform the action, the new message \(m\) is in message send list \((act1)\). Type of the message \(m\) is reply message \((act\ 2)\). The sender node is updated with reply message \(m\) \((act\ 3)\). Reply message is contain the new message \(m\) which was send by node \(n\) correspond to request message \(mm\) in action \(act4\). We snatch the load value of node \(n\) from the message \(m\) shown in action \(act5\).

G) Delivery of reply message at coordinator node: After sending the reply message from participating nodes, coordinator node (overloaded node) receive the reply message. (See figure 8)
In guard grd 4 shown that message type of mm is request and type of message m is reply message in guard grd 4 and 8 respectively. The guard grd 5 shows request message mm successfully deliver in node nn and grd 10 ensures that message m is in sender list of node n. The guard grd 11 shows that reply message send in coordinator node. After that node nn is not received the reply message in guard grd 14. Node n which are send the message m with load (ld) is not shown in directory of node nn in guard grd 16. Message m is send by node n is not deliver in node nn shown in guard grd 15.

The action act 1 represents the message m is successfully delivered in node nn. Action act 2 shows that reply message received by node nn that is send by message m to corresponding request message mm. Action act 3 ensures that load value of receiver node n is successfully shows in directory of sender node nn. Action act 4 ensures that load value ld of message m send by reply node n.

H) Split load and decrease the load value: After successfully receiving of reply message of node nn, system chooses the overloaded node to transfer the load. We compare the value of node with threshold value. If the load value is less than threshold then select the node and transfer the load into it. Any overloaded node search the directory to send the load in under loaded nodes and node can’t migrate the load due to unavailability of sufficient load in underloaded nodes. We split the load in overloaded node and transfer the load in two or more underloaded nodes. (See fig 9)
split point at receiver △

ANY n, ll, nn, mm

WHERE

grd1 : n\in\text{NODE}
grd2 : nn\in\text{NODE}
grd3 : \text{loadvalue}(nn)\in\mathbb{N}
grd4 : \text{loadvalue}(n)\in\mathbb{N}
grd5 : \text{loadvalue}(n)\times\text{value\_threshold}
grd6 : \text{loadvalue}(n)\times\text{loadvalue}(n)
grd7 : ll\in\mathbb{N}
grd8 : \text{loadvalue}(nn)\times\text{loadvalue}(nn)\times\text{loadvalue}(n)
grd9 : mm\in\text{MESSAGE}
grd10 : nn\in\text{dom}(\text{transferload})
grd11 : \text{transferload}(nn)\times\text{inprogress}
grd12 : \text{loadvalue}(n)\times\text{value\_threshold}

THEN

act1 : \text{loadvalue}(n)\times\text{loadvalue}(n)\times\text{loadvalue}(n)\times\text{loadvalue}(n)
act2 : \text{transferload}(nn)\times\text{completed}

END

Fig 10: Split point at receiver node

In figure 10 guard grd 5 ensures that load value of n is less than threshold value. Guard grd 6 shows that load value of nn is always greater than load value of node n. Load ll belongs to natural number (grd 7). We extract the load from overloaded node in guard grd 8. Guard grd 11 ensures that transferload is inprogress and loadvalue of node n is less than value of threshold ensure by guard grd 12. To perform the action we add the load in various underloaded nodes shown in action act1 and load transfer of node nn is completed in action act2.

V. RESULT ANALYSIS OF PROPOSED MODEL

In distributed load balancing we have balanced the load from overloaded nodes to under loaded nodes. All the activities in the system is verified and validated with Event B model. Event B is mathematical model which contain the set and logic operations. We convert all the load balancing steps into events and invariants for generating a proof obligation methods. Proof obligation methods are manually or automatically discharged. Rodin tool generates a total 80 proof obligations and all of them are discharged automatically. Here is a statistics of the proof table:

<table>
<thead>
<tr>
<th>Element name</th>
<th>Total P.O.</th>
<th>Automatic P.O.</th>
<th>Manual P.O.</th>
<th>Reviewed</th>
<th>Undischarged</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load context</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Load balance Machine</td>
<td>80</td>
<td>80</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

VI. CONCLUSION

To balance the load in each node is main objective of this paper and it is ensure by the formal verification. In this paper, we focus on split point load balancing technique. We have developed the model of load balancing through send and receive of messages in global environment and formalization of the model in B language on RODIN platform. In the refinement, we shows how to transfer the load from sender to receiver. Load balancing is a key role in distributed network, without it we cannot imagine to implement the system. Our model implements correctly and all the events and invariants are successfully discharge. We proof the correctness of a model using Event B. In this model, there is not any invariant violation to implement the model and all those properties must be maintained.

ACKNOWLEDGEMENT

This work is done under the DSLR project (Distributed load balancing and system recovery) governed by Uttar Pradesh Council of Science and Technology (UPCST) and supported by PSIT College Kanpur.

REFERENCES

1. Mukesh Singhal and Nirjan G. Shivratri " Advanced concept in operating System- Distributed, Database and multiprocessor operating system " Tata Mcegraw Hill 2001
3. Neeraj Rathore & Indeverve Chana “Load balancing and job Migration Techniques in Grid-A Survey of Recent Trends” In wireless Personal communication ISSN 0929-6212
5. Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek. Hari Balakrishnan " Chord: A Scalable Peer to peer Lookup Service for Internet" MIT Laboratory for Computer Science
10. M.Butler “ Concise summary of Event B mathematical toolkit”
13. Raghuraj Suryavanshi1 and Divakar Yadav " Modeling Of Distributed Mutual Exclusion System using Event B”
14. Girish Chandra1, Raghuraj Suryavanshi2 and Divakar Yadav" Formal Verification Of Distributed Checkpoint Using Event-B
15. Alexei Iliassov1, Linas Laibinis2, Elena Troubitsyna2, and Alexander Romanovsky’ “ Formal Derivation of a Distributed System”

AUTHORS

Shantanu Shukla is a research assistant in Department of Computer Science and Engineering at PSIT college, Kanpur. He has received M.Tech degree from AKTU University, Lucknow. He has presented several research paper in international journal. His current research area is in distributed system and formal methods.
Raghuraj Suryavanshi is working as Associate Professor in Department of 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. His research interests are formal verification and validation of critical properties of distributed database systems.

Divakar Yadav is professor in Computer Science and Engineering Department at Institute of Engineering and Technology, Lucknow. He holds a Ph. D. Degree in Computer science from University of Southampton, UK. His research interests include distributed computing, databases and formal methods.