Formal Specification of Memory Coherence Protocol

Jahanzaib Khan, Muhammad Atif,
Muhammad Khurram Zahoor Bajwa, Muhammad Sohaib Mahmood
Department of Computer Science
and Information Technology
The University of Lahore

Sobia Usman
Department of Computer Science
COMSATS University Islamabad
Lahore Campus

***Abstract*—Memory coherence is the most fundamental requirement in a shared virtual memory system where there are concurrent as well as loosely coupled processes. These processes can demand a page for reading or writing. The memory is called coherent if the last update in a page remains constant for each process until the owner of that page does not change it. The ownership is transferred to a process interested to update that page. In [Kai LI, and Paul Hudak. Memory Coherence in Shared Virtual Memory Systems, 1986. Proc. of Fifth Annual ACM Symposium on Principles of Distributed Computing.], algorithms ensuring memory coherence are given. We formally specify these protocols and report the improvements through formal analysis. The protocols are specified in UPPAAL, i.e., a tool for modeling, validation and verification of real-time systems.**

***Keywords*—Memory coherence; formal specification; shared memory; address space; analysis**

I. Introduction

In a loosely coupled multiprocessors system, virtual memory is useful due to its parallel infrastructure instead of using memory hierarchy. Application programs can use the shared virtual memory just as they do the traditional virtual memory. The data can be naturally migrated between processors on demands because the shared virtual memory discussed in [1] is not only pages data between physical memory and disk but it is also pages data between physical memory and individual processors, as shown in Fig. 1. The shared virtual discussed in [1] provides address space which is shared among all processors in the loosely coupled distributed memory multiprocessors systems. As the shared virtual memory on the loosely coupled multiprocessors has no physically shared memory and the communication cost between processors is nontrivial. Thus the conflicts are not likely to be solved with negligible delay [2]. The problem that Kai Li faced in building the shared virtual memory was memory coherence problem and in [3] Kai Li et al. are focusing on memory coherence problem for shared virtual memory and they provide a number of algorithms as a to solve memory coherence problem. These algorithms include the Central Manager algorithm in which the manager is just like a monitor. The second Algorithm is Improved Central Manager Algorithm. The detail of these algorithms is provided below. We investigate the algorithms with respect to their functional requirements. Our approach for formal verification is based on model-checking. We formally specify the algorithms using UPPAAL. This is comprehensive analysis of the algorithms provided in [1] along with the verification of detailed functional requirements. We give the formal specification of algorithms in functionalism: the timed automata language of UPPAAL [4].

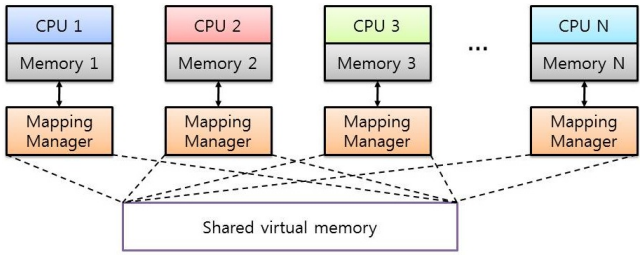

Fig. 1. Shared virtual memory mapping [3].

A. Memory Coherence

A coherent memory means the value returned by a read process is always the same as the value is written by the most recent write process to the same address. In [3] the algorithms for memory coherence are presented. Two of them are:

1) Centralized Manger Algorithm.
2) Improved Centralized Manager Algorithm.

Each algorithm has the following four basic components:

- **Read Server**: It provides a page as read only.
- **Write Server**: It provides a page for writing.
- **Participant**: A process that is owner of some pages or demands pages for reading/writing.
- **Centralized manager**: It keeps record of all the pages, like who is owner of what and who has taken a page for reading/writing. It is also responsible for changing ownership of a page.

B. Centralized Manager Algorithm

The Central Manager Algorithms maintains a table called info table having tree fields.

1) The Owner field contains the processor that is owner of the page and it is the processor which has most recently performed the writ operation on that page.
2) The copy-set field contains the list of processors having the copy of the page.
3) The Lock field to synchronize the operation.

Each process has also a table called PTable having the fields: access and lock [5]. This table keeps the information about the accessibility of the page on the local processor. In this algorithm there is no fixed owner of the page because

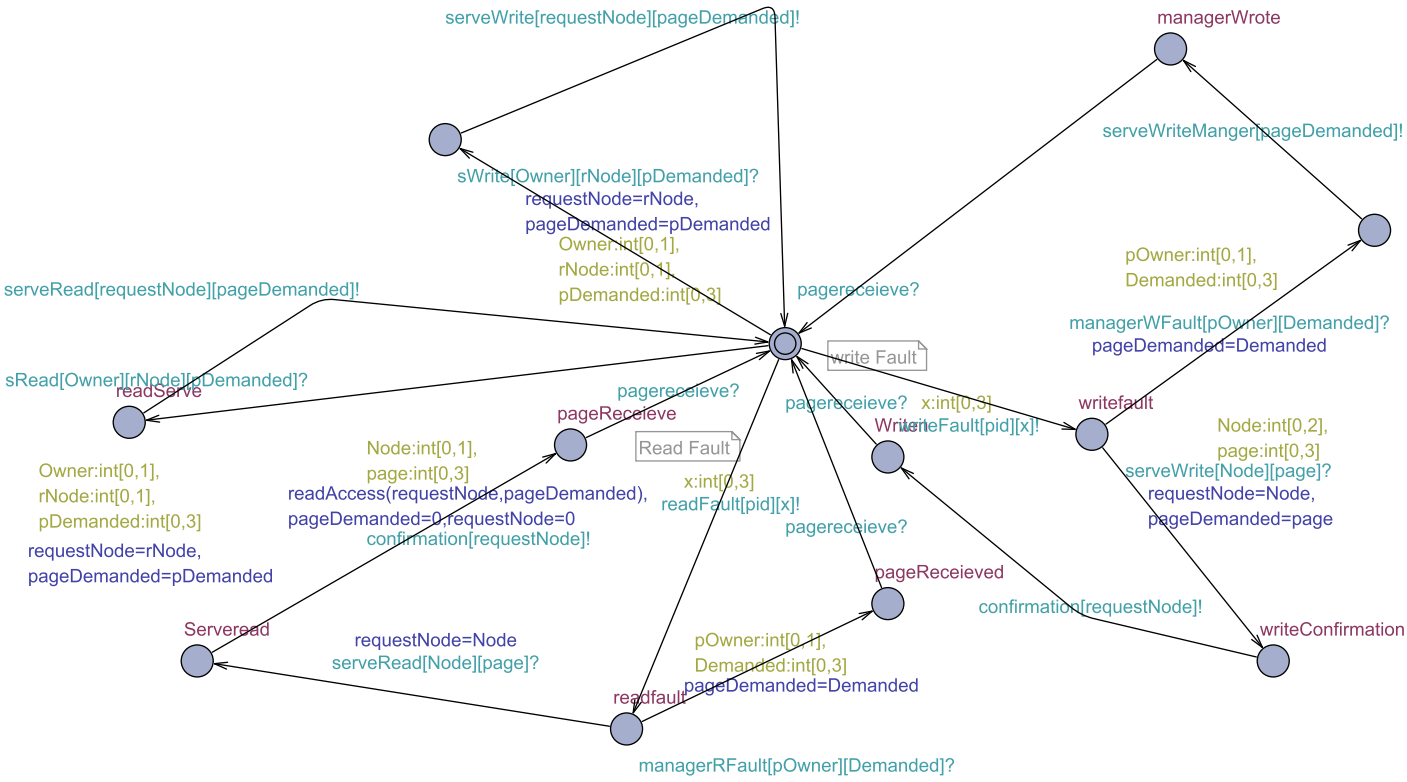

Fig. 2. Client Process.

the owner is considered to the processor who has performed the most recent write operation on the page. After the write operation an invalidation message is sent to all processors having the copy-set of that specific page [6].

Both PTable and info Table have page based locked and when there are more than one process waiting for read or write operation then this locking mechanism prevents the processor to send request [7].

C. Improved Centralized Manager Algorithm

The main difference in Central Manager and Improved Central Manager is the elimination of confirmation operation to manager [2]. The locking mechanism not only deals with local requests but also with remote requests. Compare to the cost of read fault in the Central Manager Algorithm it saves one "send" and one "receive" per page for all operations [8].

II. Related Work

In [9] author is using state model checker SPIN and he is combining the results of this checker with a testing approach which is model based to support embedded system validation. The author is using Siemens SIMATIC S7-400H a programmable logic controller as an example and he claimed that his model covered crucial part of this controller. The author concluded that formal verification is not suitable as a standalone method. He suggested that it should be combined with a suitable validation method such as testing to achieve maximum benefits. In [10] the author is verifying Chinese Lunar Rover control software, which is a real time multitasking embedded software. The purpose of the paper is to verify that system is satisfying a real time functional property. For this the author modeled an application and used physical environment as a timed automata and he is analyzing the system using a model checker of modeled in UPPAAL. He concluded that his model was able to trace and track down the undesired behavior in the system [11]. In [12] the author is providing a methodology to extract models for a wireless sensor and then he is using UPPAAL for verification of functional and non-functional properties of the developed model. In this paper the author claimed that the basic properties which are hold by a node has not been performed by any wireless network and in this research work he is addressing this individual node.

III. Formal Specification of the Memory Coherent Protocol

A. Main Process

The protocol is specified with three parallel processes to circumvent state space problem. These processes communicates with each other and are called by other processes. The protocol comprises the following sequence of actions.

1) Check the page is not locked before request (Guard).
2) Locking the page for which request has been generated for read or write (Boolean data structure is used).
3) Sending request to server.
4) Adding the process in the copy-set of the page after it has received the request for read or write (array is used).
5) Update the Node and Requested Page variables (Integers).

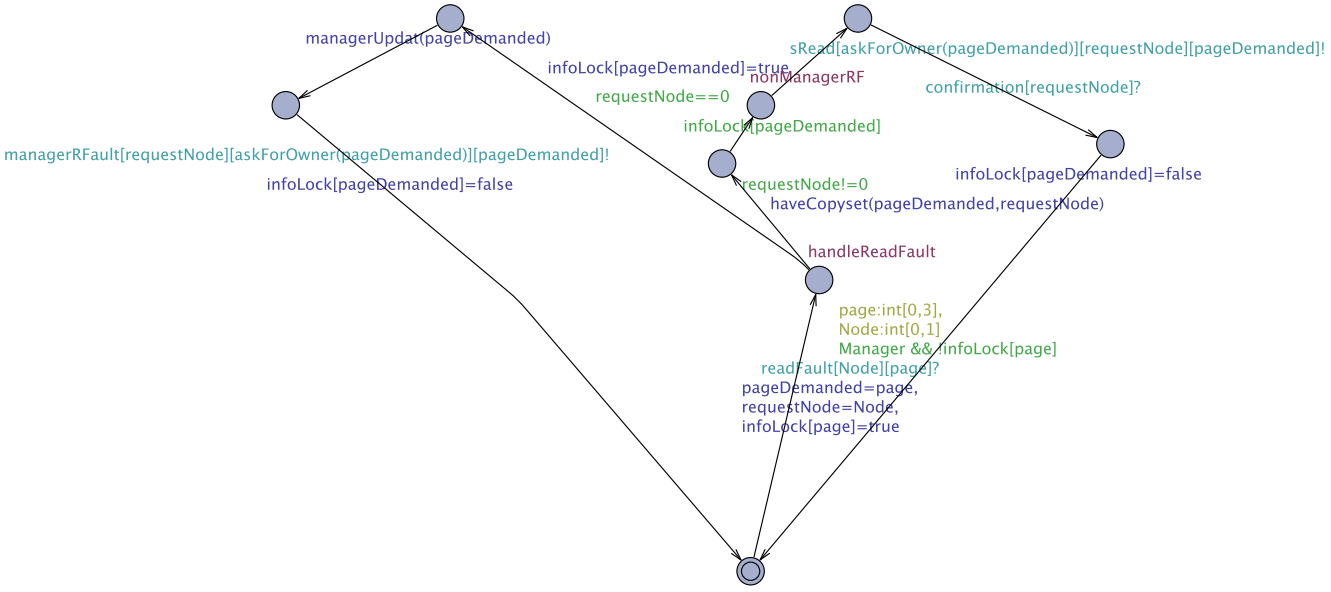

Fig. 3. Read Server Process.

6) Process is checked whether it is owner of the page (Boolean).
7) All the copy-sets of the page are invalidated in case of write fault.
8) Confirmation of successful operation is sent to the manager.
9) At completion of operation page is unlocked at manager as well as at owner end.

B. Channels

Synchronization of process is done using channels. Channels receive messages from one process and deliver these messages to other process. There are many types of channels used for synchronization of communication which includes:

1) Unicast channel: This channel is mostly used in handshaking of the processes.
2) Multicast channel: This channel is used for broadcasting a request in system.
3) Urgent channel: At urgent location time progress may not be made. At this location interleaving with normal states is not allowed
4) Committed channel: At committed state the possible transition is only one going out of committed state and the committed state has to be left immediately.

How these channels are working in our model is explained below.

C. Central manager algorithm

1) readFault [4][pages] is a channel used to send request by client as sender and received by Server where 4 is total number process on which fault can occurred.
2) managerRFault [4][4][totalPages] is a channel which is triggered by the manager if and only if the faulting process is manager itself. Manager has information about all pages and the owner of the pages there for manager will send the request to the owner of the page directly and page number is also sent along with.
3) sRead[4][4][totalPages] is a channel used to send request to owner of the page by the manager if read fault occurs on non-manager node. It is sending the requested node address and the page demanded by that node.
4) serveRead[4][totalPages] is a channel triggered by the owner of the page to the requesting process insure that access for read is granted.
5) confirmation [4] is a channel used to send confirmation to the manager for the completion of operation the id of the requesting process is also sent with confirmation.
6) writeFault[4][totalPages] is a channel used to send request by client to the server to grant access for writing in the page and if access is granted the requesting process is also declared temporarily owner of the page. On the completion of this operation all the copy-sets of the page are invalidated by the manager of the system.

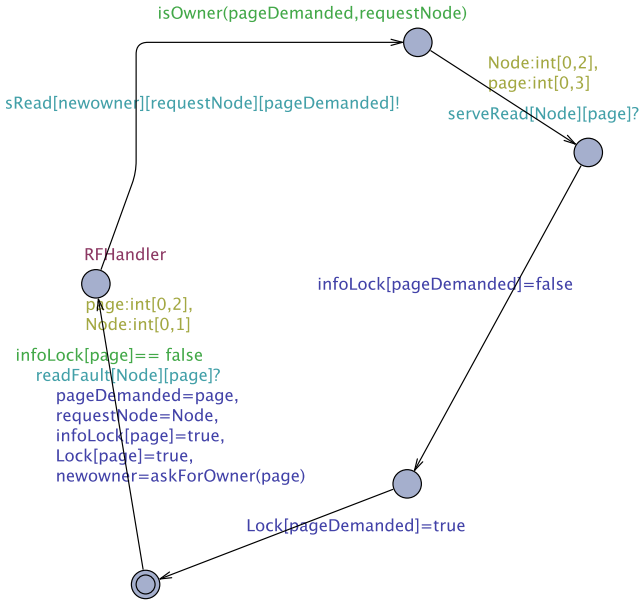

Fig. 4. Improved Read Server Process.

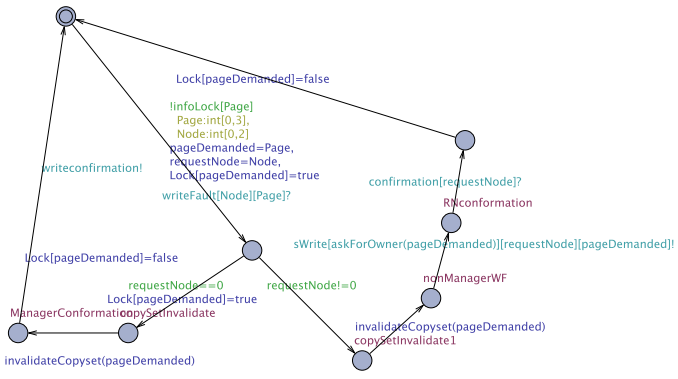

Fig. 5. Write Server Process.

7) sWrite[4][4][totalPages] is a channel used to send request to owner of the page by the manager if write fault occurs on non manager node. In this channel address of requested node and the page number is also sent as parameters.
8) serveWrite[4][totalPages] is a channel used by the owner of the page for sending the paged to the requested node for writing. After sending the page the lock of page is removed both by owner of the page and manager of the system and now it is available for new operation.
9) writeConfirmation[totalPages] is channel used to send the confirmation to the manager of system by the requesting node to assure that it has been granted the access for writing the page.

D. Declarations

Global declarations are made to access the variables throughout the system. The global variables used in our model are placed here.

const int totalPages=4; This describes the total number pages for which read or write request can be generated by the processes.

chan readFault[4][totalPages] and *chan writeFault[4][totalPages]* are unicast channels and are used to send request to the manager of the system to grant the access for their request.

chan serveWrite[4][totalPages], channel is triggered by the owner of the page to send the page for writing purposes to the requesting node.

serveRead[4][totalPages], channel is used to send the requested page to the requesting node.

sRead[4][4][totalPages], channel is used to send the request to the owner of the page to send the page for reading.

sWrite[4][4][totalPages], channel is used to send the page to the requesting node for writing by the owner of the page.

chan confirmation[4], channel is used to send confirmation of receiving a page to the manager.

managerRFault[4][4][totalPages], channel is used when read fault is occurred on the node which is also manager of the system.

serveReadManger[totalPages], channel is triggered by the manager to request the owner of the page to send the required page to the manager.

writeConfirmation, channel is used to send confirmation message to the manager for write fault.

bool Lock[8] describes the Lock that is used to lock the page. Lock is made true before serving the page to the requesting process and it is made false when page is served by the owner of the page.

bool infoLock[8], the information of the page is on manager end and before the request is sent to the owner of the page for reading or writing the page the information of the page is locked by making the infolock true and after the it is made

false by the manager after receiving the confirmation message by the requesting process.

int loop,loop2, are used in *invalidateCopySet* function.

int [-1,7] pTable[8][8]={
0,1,1,-1,1,-1,-1,0,
0,1,1,-1,1,-1,-1,0,
1,1,1,-1,1,-1,-1,0,
1,1,1,-1,1,-1,-1,0,
2,1,1,-1,1,-1,-1,0,
2,1,1,-1,1,-1,-1,0,
3,1,1,-1,1,-1,-1,0,
3,1,1,-1,1,-1,-1,0 }

This array contains page table information maintained by the owner of the page. Each row in the array is representing the page number i.e. row number 1 is showing the information about the page number 1 and row number 2 is showing the information about page number 2 and so on for all the other rows in the array. The columns of the array are showing the attributes of the pages which are explained as under.

First column is representing the owner of the page i.e in row number 1 we have written the 0 in first column which means that the owner of the first page is 0 process and in 7th row there is 3 in first column which shows that the owner of 7^{th} page is process number 3. It is hard coded because in the protocol the ownership is not changed it is only shifted temporarily in write fault case. Second column in the table is representing the information about the access of the page it can be either read or write if the access of the page is marked as read it is made 1 and for write it is made 2 and if the page has both read and write access it is then made 3. Third column in the table is representing the lock information of the page lock. If it is 0 then it means that page is locked and if it is 1 it shows that page is not locked it is unlock and available for operation. Column number 4 to the column number 7 is representing the information about the process having the copy-set of the page i.e 4th is for first process and 5^{th} is for second if it is 1 against any process this shows that the specific process has the copy-set foe that page and if it is 0 it shows that the process has not copy-set of that page. Column number 8 is representing information about either manager has copy-set of the page or not. If it is 1 it shows the manager has the copy-set of that page and if it is 0 it shows that manager has not the copy-set of the process.

int[-1,7] copyset[8][5]={
1,1,-1,-1,0,
-1,-1,-1,-1,0,
-1,-1,-1,-1,0,
-1,1,-1,-1,0,
-1,-1,-1,-1,0,
-1,-1,-1,-1,0,
-1,-1,-1,-1,0,
-1,-1,-1,-1,0 }

This array represents the copy-set table. In this table each row is representing the page number i.e row number 1 represents page number 1 and row number 2 is representing page number 2 and so on for all rows in the table and columns are representing the other properties of the pages which are as follow.

Columns are representing the processes having the copy of the page. Value 1 means that a particular process has the copy-set of the page and it is maintained in a sequence such as if in row 1 at first column the value is 1, it means process number 1 has the copy of page number 1. Similarly in row number 1 and column number 2 value 1 means that process number 2 has the copy of the page number 1. Moreover value 1 in row number 4 and column number 2, represents that copy of the page number 4 is also available on process number 2. The values -1 means that the process does not have the copy-set of corresponding pages.

Because we are using just four processes, therefore, we require just four columns to cover all the processes additionally the 5^{th} column is used for the manager of the system if the manager has the copy-set of any page then its value is 1 and the value 0 means that the manager has not the copy-set of the particular page.

int[-1,7] iTable[8][7]={
0,1,1,-1,-1,-1,0,
0,1,1,-1,-1,-1,0,
1,1,1,-1,-1,-1,0,
1,1,1,-1,-1,-1,0,
2,1,1,-1,-1,-1,0,
2,1,1,-1,-1,-1,0,
3,1,1,-1,-1,-1,0,
3,1,1,-1,-1,-1,0 }

This array is page table information maintain on the by the manager. Each row in the array is representing the page number i.e. row number 1 is showing the information about the page number 1 and row number 2 is showing the information about page number 2 and so on for all the other rows in the array. The columns of the array are showing the attributes of the pages which are explained as under.

First column is representing the owner of the page i.e in row number 1 we have written the 0 in first column which means that the owner of the first page is 0 process and in 7^{th} row there is 3 in first column which shows that the owner of 7^{th} page is process number 3. It is hard coded because in the protocol the ownership is not changed it is only shifted temporarily in write fault case. Second column in the table is representing the information about the access of the page it can be either read or write if the access of the page is marked as read it is made 1 and for write it is made 2 and if the page has both read and write access it is then made 3. Third column in the table is representing the lock information of the page lock. If it is 0 then it means that page is locked and if it is 1 it shows that page is not locked it is unlock and available for operation. Column number 4 to the column number 7 is representing the information about the process having the copy-set of the page i.e. 4^{th} is for first process and 5^{th} is for second if it is 1 against any process this shows that the specific process has the copy-set foe that page and if it is 0 it shows that the process has not copy-set of that page. Column number 8 is representing information about either manager has copy-set of the page or not. If it is 1 it shows the manager has the copy-set of that page and if it is 0 it shows that manager has not the copy-set of the process.

int[0,7] owner[4][2]={ 0, 0,
1, 1,

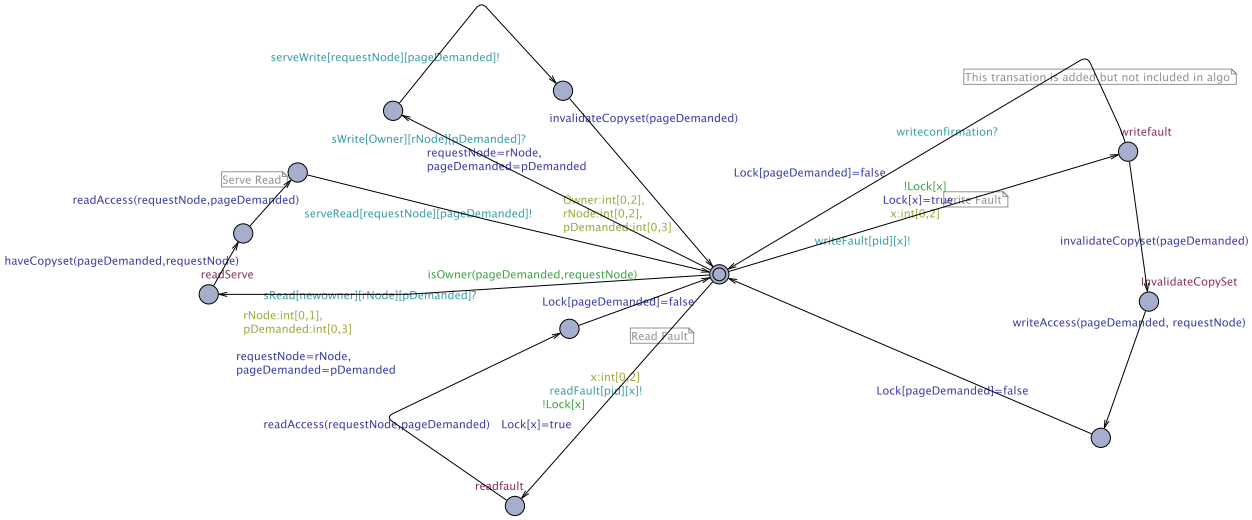

Fig. 6. Improved Central Manager.

2, 2,
3, 3 }

This array is used to store the information about the ownerships of the pages. In the array row number is representing the number of the page and the column values are representing the owner values of the pages i.e owner of first two pages (page number 1 and page number 2) is process number 0 there for in row number 1 and row number 2 there is 0 values and the owner of page number 3 and 4 is process number 1 there for the values of row number 3 and 4 are 1 and same procedure for remaining pages.

E. Methods or Functions

Following are the methods used:

```
//pid is process id
//x is page number
bool isOwner(int pid,int x)
if((pid == 0 or pid == 1)and x == 0 )
return true;
if((pid == 2 or pid == 3)and x == 1)
return true;
if((pid == 4 or pid == 5)and x == 2 )
return true;
if((pid == 6 or pid == 7)and x == 3 )
return true;
return false;
```

This function is used to find weather the process is owner of the page or not. It is taking process number and page number as arguments and using if statement to compare the value of process with the owner of the page value and if it matches it return true and if it does not match with owner value it returns false as its return type is bool (a Boolean data type). At the last statement if any if statement is not satisfied then it returns false a default value in case of process is not owner of the page. Infect it is using exhaustive search algorithm to find the owner value it is comparing the process and page id with all the possible combinations and then returning the value.

```
void readAccess(int pid, int x)
pTable[pid][x+3] = 1;
```

This function is used to grant access for reading to the requesting process against the required page. It is taking process id and page number as arguments and it is making the change in pTable and changing its access bit discussed earlier in this paper to 1 which means the process have read access of that page. As its return type is void there for it is returning nothing just making the change in pTable.

```
void haveCopyset( int p, int n)
pTable[p][n+3]=1;
```

This function is used to make the entry of a process in the copy-set of the page. It is taking process number and page number as arguments. This function is returning nothing it is just making the entry in pTable. This function is used in read

operation according to the protocol discussed in this paper if any process has read the page it must have its copy and page table is maintained for this purposes any process which has read the page is entered in the copy-set of the page in pTable.

```
int askForOwner(int p)
if(p == 0 or p == 1)
return 0;
else if(p == 2 or p == 3)
return 1;
return -1;
```

This function is used to find out the owner of the page. It takes page as an argument and returns the owner of the page and returns an integer value which is owner of page which it received. This function is used in read fault and in as well as in read fault. if it receive the page number who has no owner it returns -1. The following method is used to invalidate all copy-set of the page. It takes the page number as an argument and returns nothing. It is making the change in the pTable and changing all the values of the page against copy-set bit to -1. According to the protocol if any process writes the page then it's all copy-sets must be invalidated by the owner of the page and as well as by the manager of the system as an information. Loop is used because in pTable according to the page there is series of bits showing the processes having the copy-sets of that page. We are just putting -1 in place of that process which shows that the page has no copy-set now.

```
void invalidateCopyset(int p)
for(loop=0;loop¡4;loop++)
pTable[p][loop+3]=-1;
```

The following method is used in case when the process on which read or write fault occurs is the manager of the system. It is taking the page number as an argument and making the change in copy-set and also in ITable. This method is used only on the manager end and only the information of the page is updated and the values against the manager bits are updated in the tables.

```
void managerUpdat(int p)
copyset[p][4]=1;
iTable[p][6]=1;
```

F. Improved Central Manager Algorithm

The primary difference in central manager and improved central manager is that ownership of page is moved on individual owners so confirmation operation is eliminated.

G. Automaton of Client Process

The automaton of client process is shown in Fig. 2. The client process has 11 states and they are named as initial and it is denoted by double circle and other stages are *readFault, managerReadFault, serveRead, confirmation, pageLocking, readServe, accessRead, writeFault, writeConfirmation and requestToOwner*. The initial stage is used to send the request for read fault and write fault request to owner of the page and serve read requests are also generated from initial stage. The major actions performed by the process are described as. The automaton for client process is depicted in Fig. 2. The initial state is named as Client. The client process has two states. The first state is initial state and the second state is committed state, which is used with action of sending repaired packets to end receivers. There are overall four major actions described as:

1) Sending request to read server for read fault of the process against a specific page.
2) Receiving request from server as an owner of the page to serve a page for reading.
3) Send the page to the requesting node for reading.
4) Receive the page form owner for reading purposes.
5) Sending the confirmation after receiving the page from owner.
6) Receiving manager read fault request is fault is occurred on the manager of the system.
7) Sending request to write server for write fault of a process for a page.
8) Receiving request from server as an owner of the page to grant access of writing in a page.
9) Sending a page for writing purposes to the requesting process.
10) Receiving page from owner of the page for reading.
11) Sending confirmation to the manager after receiving the page.

The process is using two functions which are described as:

1) Changing the access of the page to read for specific process.
2) Asking a process to weather it is owner of the page or not.

To communicate with the read server readFault[] channel is used. To communicate with write server writeFault[] channel is used. Client process uses the channel sRead[] receive the request as an owner of the page to serve the page to the requesting process. If the page fault occurs on manager process the managerReadFault[] channel is used by the client process. Client process uses the channel readserve[] to send the required page to the requesting process. Client uses confirmation[] channel to send the read operation completion confirmation to the manager process. Client process uses writeFault[] channel to communicate with write server. Client process uses the channel sWrite [] to receive the request as an owner to serve the page to the requesting process. Client process uses the channel servewrite[] to send the required page to the requesting process. writeConfirmation[] Channel is used by the client process to send the confirmation of completion of write operation. To perform the first step read fault process, the variable !Lock[x] is used to check weather page is locked or not and operation is performed only against unlocked pages. The variable Lock[x]=true is used to lock the page by the client process for which read fault or write fault has been occurred so that it should not be accessed by other process for operations. The variable Lock[pageDemanded]=false is used to unlock the page which is locked before page is served for reading or writing purposes. The function readAccess(requestNode,pageDemanded) is used by the client process (Owner of the page) to grant access for read to requesting process. The variable Node:int[0,2] is used at receiving side and passed to request Node variable and after word it is used to send confirmation message to the manager and also used as a parameter to readAccess(requestNode,pageDemanded) function. The variables rN-

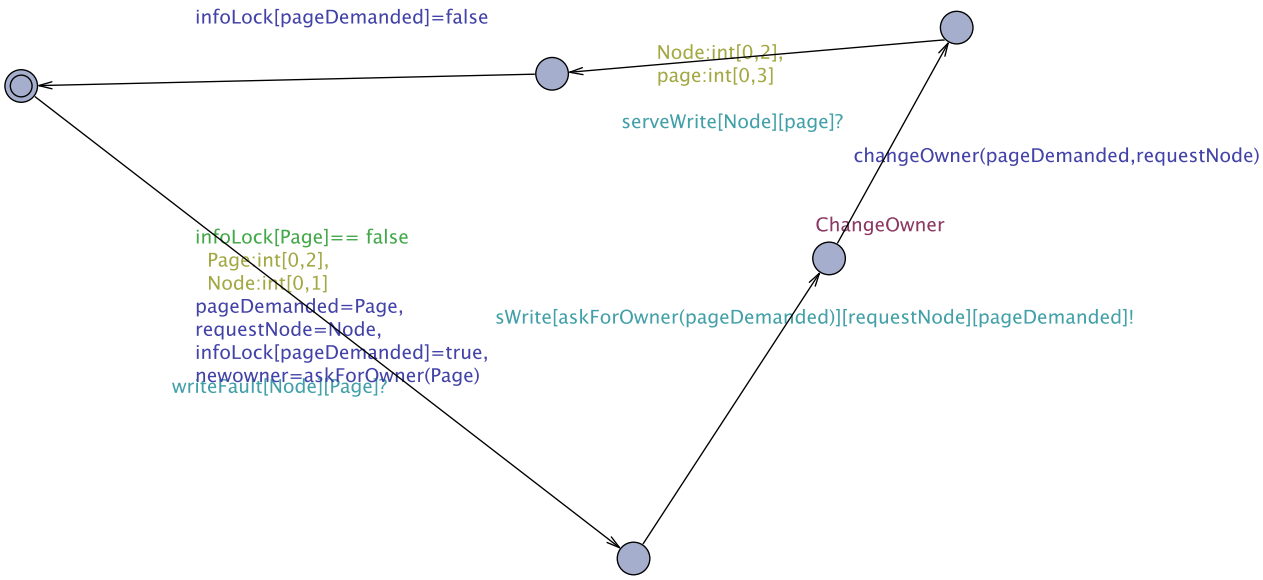

Fig. 7. Improved Write Manager.

ode:int[0,2],pDemanded:int[0,3], Owner:int[0,2] are used on by the owner on receiving side and are passed to request node and page demanded respectedly and all of the three variables are used as parameter to sWrite[Owner][rNode][pDemanded]? Channel.

These variables are also used in channel to send page to the requesting process using channel sWrite[Owner][rNode][pDemanded]? For write operation.

In the model shown above the client process is shown in figure. In the first step of read fault is the guard! Lock[x] is used to check the page and insure that it must be unlocked and process locks the page using update Lock[x]=true. In second step of synchronous local variables Owner:int[0,2], rNode:int[0,2], pDemanded:int[0,3] are used and passed to function isOwner(pageDemanded,requestNode) to check the owner ship of the page. These variables are also used in the readAccess(requestNode,pageDemanded) function by the owner of the page to grant the access of reading to the requesting process in desired page. In third step of read fault the channel serveRead[requestNode][pageDemanded]!, is used by client to send the required page to the requesting process. in the fourth step of the client process there are two possible transitions first if requesting process is not manager then the channel serveRead[Node][page]? Is executed and if the requesting process is manager then managerRFault[pid][Node][page]? Channel is executed. The fifth step read fault process in the client process is confirmation[requestNode]! In which requesting process sends the confirmation of receiving the page to the manager of the system and it unlocks the page which was locked at the starting of the operation using the update Lock[pageDemanded]=false.

While in write fault the first step of synchronization process is the guard !Lock[x] that is used to check the page and insure that it must be unlocked and process locks the page using update Lock[x]=true. The local variable x:int[0,3] is used to indicate the process on which write fault can occurred and transaction is performed using writeFault[pid][x]! channel. In second step of the client process synchronization the local variables Owner:int[0,2], rNode:int[0,2] and pDemanded:int[0,3] are used and passed to the sWrite[Owner][rNode][pDemanded]? channel to receive the request from write handler by the owner of page. In third step of the synchronization process of write fault the channel serveWrite[requestNode][pageDemanded]! is

used to send the page to requesting process for writing. At fourth step of write fault process there are two possible options available first if the requesting process is the manager of system then page is received and secondly if requesting node is not the manager then the channel serveWrite[Node][page]? is used to receive the page from owner of the page. At fifth step of write fault in client process confirmation message is sent to the manager by requesting node using the channel confirmation[requestNode]!, and required page is unlocked using update Lock[pageDemanded]=false.

H. Automaton of Read Server

The automaton of read server is depicted in Fig. 3. The initial stage is denoted by double circle. The read server has 8 states. First one initial state and named as as ReadServer and other states are handleReadFault, InfoLock, nonManagerRF, RecieveConfirmation, InfoLock,padetManager and UnLock-Info. There are over all two functions described as:

1) haveCopyset(pageDemanded,requestNode)
2) managerUpdat(pageDemanded)

The read server plays main role between client process and server process. It communicate with client process and server as well. It receives request from client of faulting page and checks the availability of the page. If page is available for the operation then the variables page:int[0,3] and Node:int[0,1] are initialized and passed to the variables pageDemanded and requestNode respectively. In the mean while the demanded page is locked using the Boolean variable infoLock[page]=true. After this the read server will check the requesting node for manager and if it is not manager of system then it will add the requesting process in the copy-set of the page using the function haveCopyset(pageDemanded,requestNode). After making the transation the information of the page is locked using the Boolean variable infoLock[pageDemanded]=false. The channel sRead[askForOwner(pageDemanded)][request Node] is used by the read server to send the request the owner of the page asking it to send the required page to the requesting process.

After sending request to the owner of the page the read server will wait for confirmation from the requesting process after receiving the page by using the Chanel confirmation [requestNode]? At the receiving end and after receiving the confirmation from the requesting process the information of the page is unlocked using the Boolean variable infoLock[pageDemanded]=false. If the requesting node is the manager of the system then at first step the information of the page is locked using the variable infoLock[pageDemanded]=true. At the second step the function managerUpdat(pageDemanded) is called by the read server to make the entry in the Ptable in the row of the page (as discussed earlier) for the manager having the copy-set of the page. At third step the read server will send the request to the owner of the page to send the requested page to the requesting process (the manager) and unlock the information of the page using the Boolean variable infoLock[pageDemanded]=false.

I. Automaton of Write Server

The automaton of write server is shown in Fig. 5. The initial stage is double circled and named as Start. The write server has 8 states. The first one is initial stage and other are Write, copySetInvalidate, ManagerConfirmation, copySetInvalidate1, nonManagerWF, RNconfirmation, UnLockPage.

In Fig. 6, the client process of the improved central manager is given. We can see that the main difference between the central manager and the improved central manager is that the later does not send or receive any kind of acknowledgment.

In Fig. 4, the behaviour of read server for the improved central manager is given. It receives a request for page demanding process and operates like a central manager but with the only difference, that is the requesting process does not send or receive any acknowledgment.

In Fig. 7, the model of write server is given. It receives the request from requesting process for writing a page and operates like the central manager's write server but with the only difference, that is the requesting process does not send or receive any confirmations.

IV. Conclusion

Memory coherence is a vital issue in today operating system. This paper models the read and write process by using UPPAAL tool. The modeling results highlight the missing details of the read and write protocols. Modeling encounters two types of limitations. First is to limit the number of demand pages, second one to reduce the number of processes. These limitations prevent the system in generating a very huge state space and also in avoiding the state space explosion problem. These limitations are imposed due to limited memory of the machine. The machine can go out of memory during verification phase. Models properties are not affected due to these limitations because a few number of pages and processes can reflect the behavior of a huge system. This small system is transparent reflection of a huge system with maximum number of pages and huge number of processes .

References

[1] C. Baier and J.-P. Katoen, *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[2] Y. Feng, L. Zhang, D. N. Jansen, N. Zhan, and B. Xia, "Finding polynomial loop invariants for probabilistic programs," in *Automated Technology for Verification and Analysis - 15th International Symposium, ATVA 2017, Pune, India, October 3-6, 2017, Proceedings*, 2017, pp. 400–416. [Online]. Available: https://doi.org/10.1007/978-3-319-68167-2_26

[3] K. Li and P. Hudak, "Memory coherence in shared virtual memory systems," in *Proceedings of the Fifth Annual ACM Symposium on Principles of Distributed Computing*, ser. PODC '86. New York, NY, USA: ACM, 1986, pp. 229–239. [Online]. Available: http://doi.acm.org/10.1145/10590.10610

[4] A. David, K. G. Larsen, A. Legay, M. Mikuăionis, and D. B. O. Poulsen, "Uppaal smc tutorial," *Int. J. Softw. Tools Technol. Transf.*, vol. 17, no. 4, pp. 397–415, Aug. 2015. [Online]. Available: http://dx.doi.org/10.1007/s10009-014-0361-y

[5] J. Li, L. Zhang, S. Zhu, G. Pu, M. Y. Vardi, and J. He, "An explicit transition system construction approach to LTL satisfiability checking," *Formal Asp. Comput.*, vol. 30, no. 2, pp. 193–217, 2018. [Online]. Available: https://doi.org/10.1007/s00165-017-0442-2

[6] W. Shen, G. Li, C. Lin, and H. Liang, "Foundation of a framework to support compliance checking in construction industry," in *Structured Object-Oriented Formal Language and Method - 7th International Workshop, SOFL+MSVL 2017, Xi'an, China, November 16, 2017, Revised Selected Papers*, 2017, pp. 111–122. [Online]. Available: https://doi.org/10.1007/978-3-319-90104-6_7

[7] A. Legay, D. Nowotka, D. B. Poulsen, and L. Traonouez, "Statistical model checking of LLVM code," in *Formal Methods - 22nd International Symposium, FM 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 15-17, 2018, Proceedings*, 2018, pp. 542–549. [Online]. Available: https://doi.org/10.1007/978-3-319-95582-7_32

[8] A. Boudjadar, A. David, J. H. Kim, K. G. Larsen, M. Mikucionis, U. Nyman, and A. Skou, "Statistical and exact schedulability analysis of hierarchical scheduling systems," *Sci. Comput. Program.*, vol. 127, pp. 103–130, 2016. [Online]. Available: https://doi.org/10.1016/j.scico.2016.05.008

[9] A. Ulrich and A. Votintseva, "Experience report: Formal verification and testing in the development of embedded software," in *Proceedings of the IEEE 26th International Symposium on Software Reliability Engineering*, ser. ISSRE '2015. IEEE, 2015, pp. 293–302.

[10] L. Shan, Y. Wang, N. Fu, X. Zhou, L. Zhao, L. Wan, L. Qiao, and J. Chen, "Formal verification of lunar rover control software using uppaal," in *Lecture Notes in Computer Science*, vol. 8442. Springer, Cham, 2014, pp. 718–732.

[11] R. Marinescu, H. Kaijser, M. Mikucionis, C. Seceleanu, H. Lönn, and A. David, "Analyzing industrial architectural models by simulation and model-checking," in *Formal Techniques for Safety-Critical Systems - Third International Workshop, FTSCS 2014, Luxembourg, November 6-7, 2014. Revised Selected Papers*, 2014, pp. 189–205. [Online]. Available: https://doi.org/10.1007/978-3-319-17581-2_13

[12] M. Hammad and J. Cook, "Compositional verification of sensor software using uppall," in *Proceedings of the IEEE 23rd International Symposium on Software Reliability Engineering*, ser. ISSRE '2012. IEEE, 2012.