IMPLEMENTATION OF CHANDY LAMPORT'S GLOBAL SNAPSHOT ALGORITHM USING MPI

BY

SAURA MANDAL
&
TARUN NASKAR

UNDER THE GUIDANCE OF

MR. UTPAL RAY
ASSISTANT PROFESSOR
DEPARTMENT OF INFORMATION TECHNOLOGY
JADAVPUR UNIVERSITY

OUTLINE

- INTRODUCTION
- MULTICOMPUTING AND MPI BASICS
- MONEY TRANSACTION SYSTEM
- CHANDY LAMPORT'S GLOBAL SNAPSHOT ALGORITHM
- IMPLEMENTATION OF CHANDY LAMPORT'S GLOBAL SNAPSHOT ALGORITHM USING MPI
- CONCLUSION
- REFERENCES

INTRODUCTION

- The global state of a distributed system is a collection of the local states of its components.
- Recording the global state of a distributed system is an important paradigm.
- A continuously running distributed system(Money Transaction System) is required to take global snapshots.
- The Global snapshot model is implemented using MPI in a multicomputing environment.

MULTICOMPUTING AND MPI BASICS

- Distributed memory is used in massively parallel multicomputers and provides high levels of performance.
- Multicomputers solve the Grand Challenge computational science problems.
- Multicomputers communicate by message passing.
- MPI stands for Message Passing Interface.
- MPI is a library specification for message-passing, proposed as a standard by a broadly based committee of vendors, implementers, and users.

BASIC MPI ROUTINES

- **MPI_Init()** starts the MPI runtime environment.
- **MPI_Comm_size()** gets the number of processes, N_p.
- **MPI_Comm_rank()** gets the process ID of the current process which is between **0** and $\mathbf{N_p - 1}$, inclusive. (These last two routines are typically called right after MPI_Init()).
- **MPI_Send()** sends a message from the current process to another process (the destination).
- **MPI_Recv()** receives a message on the current process from another process (the source).
- **MPI_Finalize()** shuts down the MPI runtime environment.

COMPILING AND RUNNING A MPI PROGRAM

- **Compiling a MPI program**

  mpicc <filename> -o <objectfilename>

- **Running a MPI program**

  mpirun -np n <objectfilename>

  n= The number of processors.

  - Run an object file on multiple machines using a command:

  mpirun -np n -pernode --hostfile my_host <objectfilename>

  pernode ensures a single process executes in each node.
  hostfile contains the IP addresses of the nodes. The first IP address of the hostfile is the master processor's IP address.

MONEY TRANSACTION SYSTEM

- A money transaction system is a system of a number of nodes (N), implemented to transfer a random amount of money from any node to any other node in a distributed environment.

- Checks data consistency in global snapshot algorithm.

CHANDY LAMPORT'S GLOBAL SNAPSHOT ALGORITHM

Algorithm:

Marker Sending Rule for process *i*:

begin

(i) Process ***i*** records its state.

(ii)For each outgoing channel C on which a marker has not been sent, ***i*** sends a marker along C before ***i*** sends further messages along C.

end

Marker Receiving Rule for process *j*:

On receiving a marker along channel C:

If *j* has not recorded its state **then**

begin

(i)Record the state of C as the empty set.

(ii)Follow the '**Marker Sending Rule**'.

end

else

Record the state of C as the set of messages received along C after ***j's*** state was recorded and before ***j*** received the marker along C.

IMPLEMENTATION OF CHANDY LAMPORT'S ALGORITHM USING MPI

Assumptions: ***Recording State*** and ***Sending Marker*** are ***atomic*** operations.
Each processor is initialized with $10,000.
The below code snippet does the money transaction system.

```
for (int i=0;i<size;i++) {
        if(rank==i)
            continue;
            int d=rand()%10+1;
            MPI_Send(&d,1,MPI_INT,i,0,MPI_COMM_WORLD);
            data-=d;
        g_node.channel_record[i]=data;
            g_node.channel_trans[i]=d;
                MPI_Recv(&d,1,MPI_INT,i,0,MPI_COMM_WORLD,&status);
            data+=d;
}
```

IMPLEMENTATION OF CHANDY LAMPORT'S ALGORITHM USING MPI

```
//initialize the channel buffer, state and status of channel
for(int i=0;i<size;i++) {
	g_node.channel_status[i]=RED;
	g_node.channel_record[i]=null;
}
g_node.state=data;
srand(time(NULL));
```

IMPLEMENTATION OF CHANDY LAMPORT'S ALGORITHM USING MPI

```
while (true) {
	perform_money_system_transaction();	//running indefinitely
	scanf("%c %d",&MARKER,&initiator);
	if(initiator > size-1) {
			printf("Enter a valid node!\n");
			MPI_Abort(MPI_COMM_WORLD,1);
	}
	//SENDING MARKER
	for(int i=0;i<size;i++) {
		if(rank==i)
			continue;
		else if(rank==initiator) {
	//start recording on incoming channels  and make incoming channels wideopen
			for(int j=1;j<size;j++)
				g_node.channel_status[j]=RED;
		}
```

IMPLEMENTATION OF CHANDY LAMPORT'S ALGORITHM USING MPI

```
    //send markers to other processors
ierr=MPI_Isend(&MARKER,1,MPI_CHAR,i,0,MPI_COMM_WORLD,&send_request);
            g_node.state=data;      //record own state
            ierr=MPI_Wait(&send_request,&status);
    }
```

IMPLEMENTATION OF CHANDY LAMPORT'S ALGORITHM USING MPI

```
//RECEIVING MARKER
for(int i=0;i<size;i++) {
        if(rank==i)
                continue;

        //received marker for the first time

        ierr=MPI_Irecv(&MARKER,1,MPI_CHAR,i,0,MPI_COMM_WORLD,&recv_request);
        ierr=MPI_Wait(&recv_request,&status);
        //if marker is received for the first time, record own state
        if(g_node.channel_status[i]==RED) {
                g_node.state=data;            //record state
                g_node.channel_record[rank]=0;   //intiate NULL
                g_node.channel_status[i]=GREEN;  //close the channel
        }
```

IMPLEMENTATION OF CHANDY LAMPORT'S ALGORITHM USING MPI

```
else { //if received earlier

        if(rank==initiator) {
                if(g_node.channel_status[i]==GREEN)
                        ;
                //check if all the incoming channels are recorded GREEN
                for(int j=0;j<size;j++) {
                        if(j==initiator)
                                continue;
                        if(g_node.channel_status[j]==RED) {
                                g_node.allgreen=0;
                                break;
                        }
                }
```

IMPLEMENTATION OF CHANDY LAMPORT'S ALGORITHM USING MPI

```
//if all are green we are done
            if(g_node.allgreen)
                break;
            else {
                g_node.allgreen=true;
                break;
            }
        }
```

IMPLEMENTATION OF CHANDY LAMPORT'S ALGORITHM USING MPI

```
else {
//for non-initiator processes check if all are green
for(int j=0;j<size;j++) {
        if(rank==j)
                continue;
        if(g_node.channel_status[j]==RED) {
                g_node.allgreen=0;
                break;
        }
}
if(!g_node.allgreen)
        continue;
else {
        g_node.allgreen=true;
        break;
}
}}}
```

IMPLEMENTATION OF CHANDY LAMPORT'S ALGORITHM USING MPI

```
    for(int i=0;i<size;i++)
        g_node.state-=(g_node.channel_trans[i]);
    sum+=g_node.state;

    array[0]=g_node.state;
    for(int i=1;i<(size+2-1);i++)
        array[i]=g_node.channel_trans[i-1];
    array[size+2-1]=sum;
    MPI_Send(&array,size+2,MPI_INT,0,0,MPI_COMM_WORLD);

}
```

IMPLEMENTATION OF CHANDY LAMPORT'S ALGORITHM USING MPI

```
if(rank==0) {

        for(int i=0;i<size;i++) {
                sum+=g_node.channel_trans[i];
            }

        for(int i=0;i<size;i++)
                g_node.state-=(g_node.channel_trans[i]);
            sum+=g_node.state;
```

IMPLEMENTATION OF CHANDY LAMPORT'S ALGORITHM USING MPI

```
int f=0;
f+=sum;
int i;

for(i=1;i<size;i++) {
        MPI_Recv(&array,size+2,MPI_INT,i,0,MPI_COMM_WORLD,&status);
        printf("\t%d",i);
        for(int j=1;j<(size+2);j++)
        {
                //info[i][j-1]=tarray[j-1];
                printf("\t%d",array[j-1]);
        }
```

IMPLEMENTATION OF CHANDY LAMPORT'S ALGORITHM USING MPI

```
    f+=array[size+2-1];
        if(i==size-1) {
            printf("\n\n\n\n   INITIATOR PROCESSOR: %d",initiator);
            printf("\n\n   TOTAL SUM: %d",f);
            totaltime = ((double) (final_time - initial_time));
            printf("\n\n   TIME TAKEN : %f sec\n\n",totaltime);
        }
        else
            printf("\n");
    }

}
```

OBSERVATIONS

```
m 0

-----------------------------------------------------------------
   CHANDY     LAMPORTS      GLOBAL     SNAPSHOT
-----------------------------------------------------------------
   PROCESSOR     STATE   P0        P1        P2        P3

        0        9975    0         9         10        3
        1        9980    4         0         6         3
        2        10010   1         3         0         5
        3        9974    7         2         8         0


   INITIATOR PROCESSOR: 0

   TOTAL SUM: 40000

   TIME TAKEN : 0.000082 sec
```

The final sum is always consistent.

CONCLUSION

- The basic idea is to use two colors viz. **RED** and **GREEN** which indicate whether a process has already taken its local snapshot and whether a message was sent before or after the local snapshot of a process.
- Messages which would make a snapshot inconsistent can easily be recognized and avoided, and messages which are in transit can be caught by the receiving process.
- The total amount of money was always consistent.

FURTHER WORK

- There are several variants of the Chandy Lamport's snapshot algorithm. Like **Spezialetti** , **Venkatesan's** and **Kearns** algorithm.
- More distributed systems like the money transaction system can be developed to check these algorithms.

References

- ONLINE RESOURCE: MPI-1 standard, http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html

- Chandy K M and Lamport L "Distributed snapshots: determining global states of distributed systems" ***ACM Trans. Computer Systems 3*** 1985 ,pp 63-75

- Acharya **A** and Badrinath B R "Recording distributed snapshots based **on** causal order of message delivery ***Information Processing Lett"*** 1992, pp 317-21

- Spezialetti M and Kearns P "Efficient distributed snapshots ***Proc.*** 6th ***Int. Conf. on Distributed Computing Systems"***, 1986, pp 382

THANK YOU