CHAPTER 4
DESIGN OF STATIC TOKEN RING CHECKPOINTING (STRC)

4.1 Introduction

The introduction of token ring in the checkpointing process gives a new way of designing the checkpointing algorithm. The non-blocking makes the algorithm more suitable for mobile environment. The proposed DTRC model of the checkpointing algorithm overcomes the constraints of the checkpointing algorithms and produces better outcome. In some practical applications, difficulties take place when passing the dependency array in the wireless network. In that situation, the dependency array can be stored in one common place and can be updated by the process. Storing the dependent details in MSS reduce much of the overhead of mobile host. As the dependency array is stored in a place, this protocol is called static protocol. This chapter discusses the design of the proposed Static Token Ring checkpointing algorithm. The pseudo code of the proposed algorithm is also presented in this chapter. This chapter also illustrates the correctness of the STRC algorithm.

4.2 Static Token Ring Checkpointing (STRC)

The second version is the Static Token Ring Checkpointing algorithm. In STRC algorithm the dependency array is stored in the MSS. As every mobile host communicates through MSS, the dependency array is updated in the MSS. There is no need to pass the dependency array in the wireless network. The coordinator initiates the checkpointing process by creating the dependency array and stores it in the MSS. The MSS maintains a list of processes (LS) that had taken checkpoints. During checkpointing, the MSS sends the checkpoint
request to the least priority process in Dp. When a process receives the 
checkpoint request, it will take a checkpoint and send the local dependency 
information Dcp to the MSS. After receiving the Dcp it will update the Dp by 
adding the process ids of Dcp if they are not present in LS and Dp. And then 
the MSS adds the current process id to LS. When an empty Dp is found by 
MSS, it deletes LS and Dp and it sends a finish signal to the coordinator. After 
receiving the finish signal, the coordinator knows the end of the checkpointing 
process. If needed, it initiates the second checkpointing process.

As in the case of DTRC protocol, the problem may arise because of the 
two possibilities.

Case (i): After taking a checkpoint a process sends a message to a 
process that has not yet participated in the checkpointing process

Case (ii): Before taking a checkpoint a process sends a message to a 
process that has already participated in the checkpointing process. These cases 
can be effectively handled by the MSS.

In STRC algorithm, the MSS maintains the list of processes that had 
taken a checkpoint. When processes in different states try to communicate, the 
MSS sends a request to the process that has not yet participated in the 
checkpointing process to take a checkpoint. If a process in the LS sends a 
request to communicate with a process not in the list, the MSS sends a request 
to that process to take a checkpoint. When a process receives a checkpoint 
request from MSS, it takes a checkpoint and sends its dependent information as 
Dcp. After receiving Dcp from the process, the MSS updates LS by adding the 
participated process id, say the process P_i’s id and updates Dp by deleting P_i 
and adding P_i’s dependent process ids that are not present in LS and Dp. If an
empty token is found by the MSS, it informs the coordinator with a finish signal. According to the necessity, the coordinator initiates the next checkpointing process.

**Data structures used**

- **Dp** - A dependent array created and passed as a token and updated by the process.
- **LS** - A dependency array used by MSS to store the ids’ of the processes participated in the checkpointing process.
- **F_{pt}** - Trigger used to indicate the end of checkpointing process.
- **Dcp_{i}** - A local dependent array maintained by process P_{i} during the checkpointing process.
- **Cpt** - Checkpointing request message.

### 4.2.1 Static Token Ring Checkpointing Algorithm (STRC)

**Algorithm**

**At the Coordinator process P_{cor}**

Step 1: Initiate the checkpointing process

(i) Take a checkpoint

(ii) Pass the checkpointing request to MSS with Dcp_{cor}

Step 2: Do the normal Computation

Step 3: if receives a finish signal from MSS

Initiates the next checkpointing process if necessary

**At the Mobile Support Station (MSS)**

Step 1: During checkpointing Request from P_{cor}

(i) Create dependency array Dp with Dcp_{cor}

(ii) Create list LS with P_{cor}
(iii) Sort Dp
(iv) Send checkpoint request Cpt to the first process of Dp

Step 2: If receive Dcp\textsubscript{i} for any process P\textsubscript{i}
   (i) if P\textsubscript{i} is present in Dp
       Delete P\textsubscript{i} from the list Dp
       Add P\textsubscript{i} to the list LS
   (ii) If an id in Dcp\textsubscript{i} is not in LS and Dp
       Add that id to Dp
   (iii) Send the checkpoint request Cpt to the least process in Dp

Step 3: If receives communication request from a process P\textsubscript{i} to P\textsubscript{j}
   If P\textsubscript{i} and P\textsubscript{j} are in LS or both are not in LS
       Allow the communication
   Else
       If either P\textsubscript{i} or P\textsubscript{j} not in LS
       send a Cpt to the process which is not in LS

Step 4: If found an empty Dp
   (i) Send finish signal F\textsubscript{pt} to the coordinator.
   (ii) Delete Dp and LS

At any processes P\textsubscript{j} in the system

Step1: If checkpoint request receives from MSS
   (i) Take a checkpoint
   (ii) Send Dcp\textsubscript{i} to MSS
       Else if it is normal computation request
           Update Dcp\textsubscript{i} and do the normal computation
Algorithm Explanation

The static algorithm divides the functionalities into three different parts namely, the functions of the coordinator, functions of MSS, and functions of a process on receiving a checkpoint request; in the first part of the algorithm, the functions of the coordinator are stated. The coordinator initiates the checkpointing process by taking a checkpoint and sends its dependency details to the MSS. Then it continues its normal computation. When it receives a finish signal from the MSS, it becomes ready to initiate the next checkpointing process if necessary.

The MSS plays a very important role in the static algorithm. The role of MSS is stated in the second part of the algorithm. When the MSS receives a request from the coordinator it creates a dependency array Dp. And also it creates another list LS to store the ids of processes that have already participated in the checkpointing process. At first, it includes the coordinator id in LS and the coordinator’s dependent details to Dp. Then, it finds the least element in the Dp and sends the checkpoint request to that least process in Dp. When it receives a dependency array as a reply for checkpoint request, it checks both LS and Dp. If an id is not present in both the lists, it enters that id into the Dp. It deletes the current process id from Dp and adds it to LS.

It also handles request from a process P_i to P_j. If P_i and P_j are both in the list it allows them to communicate. Similarly if they are not in the list LS it allows them to communicate. But if any one of these two processes is not in LS, it asks that process to take a checkpoint and updates the Dp.
The third part explains the functionalities of a process $P_i$. When a checkpoint request is received from MSS, it takes a checkpoint and sends its local dependent details as a parameter. In this way, the static algorithm reduces the size of the dependency array passed in the network.

4.2.2 Example for STRC Algorithm

The Static Token Ring Checkpointing Algorithm can be explained clearly with an example shown in Figure 4.1. Let $P_1$, $P_2$, $P_3$, $P_4$ and $P_5$ be five processes in the distributed system. They connect and communicate through the MSS. Let $P_1$ be the coordinator. The process $P_1$ sends message to $P_3$ and $P_5$. The process $P_2$ sends message to $P_5$ and $P_1$. After receiving the message from the process $P_2$, $P_5$ sends a message to $P_1$. The coordinator $P_1$ takes a checkpoint and sends the checkpointing request to MSS with its own dependents $P_3$ and $P_5$. On receiving the checkpointing request, the MSS starts the checkpointing process. The new dependency array $Dp$ is created with $P_3$ and $P_5$ because $P_1$ depends on $P_3$ and $P_5$. And $P_1$ is included in the list LS. MSS sorts the $Dp$ and finds out the next lower priority process which is $P_3$. It sends the request to $P_3$. On receiving the checkpoint request, $P_3$ takes a checkpoint and sends its dependent process id say $P_1$ and $P_2$.

On receiving the reply from $P_3$, the MSS checks both LS and $Dp$. Any process id that is not present in both of them, is included in $Dp$. In this case $P_2$ is not present, so it includes $P_2$ in $Dp$. Then, it includes $P_3$ in LS and deletes $P_3$ from $Dp$. Next the MSS sends the checkpointing request to $P_2$, the next lower priority process in $Dp$. After receiving the checkpointing request, $P_2$ takes checkpoint and sends the dependent information $P_3$ and $P_5$ to the MSS. As $P_3$ is in LS and $P_5$ is in $Dp$, no updating takes place in $Dp$. $P_3$ is included in LS. $P_2$ is
deleted from Dp. Next the MSS sends the checkpoint request to P5. P5 sends the process id of P1 and P5 to MSS. In MSS, P1 is present in Dp. So the process id P5 is deleted from Dp. Dp now becomes empty. The MSS informs it to the initiator with a finish signal and the checkpointing process ends. P4 does not participate in the process and it is not included in the checkpointing process. So the number of processes taking checkpoint is minimized and the size of the dependency array is diminished.

![Diagram](attachment:image.png)

**Figure 4.1 Example for Static Token Ring Checkpointing Algorithm**

### 4.2.3 Mobility and STRC Algorithm

The STRC algorithm also handles the mobility issues. In this algorithm, the dependent information is stored in one place. If any host moves from one cell to another, the current MSS forces that process to take checkpoint and updates the dependent information in MSS. The leaving process must be a
checkpointed process. If a process in the MSS sends a computation message to a process out of a cell, the MSS forces the process to take checkpoint if it has not yet participated in the checkpointing process. During the second checkpointing process, the home MSS is used to find out the location of the mobile host. The MSS of the coordinator process is used to store the dependency information and it sends the checkpoint request to the MSS where the mobile hosts are currently present.

4.2.4 Proof of Correctness of STRC Algorithm

**Finite termination of static algorithm:** The static algorithm is terminated at a finite time.

According to the algorithm, after taking a checkpoint the processes delete their information from Dp and add it to LS. The process ids of dependent processes that are not in LS are included in the Dp. So during checkpointing the possible size of Dp is decreased and the LS is increased. When LS includes all the elements in the system, the Dp must become empty. Then there is no other element in the Dp to send the checkpointing process. This will terminate the algorithm at a finite time.

**Creation of consistent Global checkpoint:** The static algorithm creates the consistent global checkpoints.

According to the algorithm, the message is accepted by MSS only if both the processes P_i and P_j are either present or not present in the LS. If any process not in LS sends a message to a process in the LS, the MSS sends a checkpoint request to that process. This will create the consistent global checkpoint. Thus the STRC always creates consistent global checkpoint.
Recovery

The Recovery algorithm used to recover the consistent system state in STRC after failure is the same as in DTRC algorithm explained in Chapter 3.

4.3 Conclusion

The algorithms proposed in this work overcome the difficulties of checkpointing algorithm in many ways. The storage utilization or the amount of information passed along the communication media during checkpointing is diminished. The number of processes participating in the checkpointing is reduced using the dependency array. Both DTRC and STRC are single phase algorithms and are terminated at a finite time. These algorithms always produce consistent global checkpoint. They can handle the mobility of the nodes very effectively. These algorithms are well suitable for mobile environment. The implementation of the algorithms and their results are discussed in the following chapter.