ISSN: 2278 – 1323
International Journal of Advanced Research in Computer Engineering & Technology
Volume 1, Issue 5, July 2012

A Failure Recovery Scheme in Mobile Computing System Based on Checkpointing and Handoff Count

Anurag Sachan , Prachi Maheshwari

Abstract— In Mobile hosts failure probability is high. An efficient checkpointing technique and a failure recovery scheme together can make a mobile computing system fault-tolerant. For efficient recovery, information of a mobile host should be kept in an organized manner. Efficiency of a recovery scheme can be measured in terms of time and cost. Mobile hosts move randomly and handoff occurs. Information of a single mobile host gets scattered over a number of mobile support stations that can be at closer or further distance. Recovery time and cost primarily depend on number of mobile support stations from which information to be collected as well as distance among them. Larger the distance, longer the time for communication through message passing. Number of mobile support stations from which information to be recovered and distance among them can be delimited by keeping a handoff threshold value in each mobile host. Recovery scheme proposed here applies both the measures. Our work optimizes both failure-free and failurerecovery operation costs.

Index Terms— coordinated checkpointing, communication induced checkpointing, domino effect, minimal checkpoint, non-blocking checkponting, optimistic logging, message passing systems

1 INTRODUCTION

DISTRIBUTED systems today are ubiquitous and enable many applications, including client-server systems, transaction processing, World Wide Web, and scientific comping, among many others. The vast computing potential of these systems is often affected by their chances of failures.

Therefore, many techniques have been developed to add reliability and high availability to distributed systems. These techniques include transactions, group communication, and rollback recovery, and have different tradeoffs and focuses. For example, transactions focus on data-oriented applications, while group communication offers an abstraction of an ideal communication system that simplifies the development of reliable applications. This survey covers transparent rollback-recovery, which focuses on long running applications such as scientific computing and telecommunication application.

Rollback-recovery treats a distributed system as a collection of application processes that communicate through a network. The processes have access to a stable storage device that is unaffected of all possible failures. Processes achieve fault tolerance by using this device to save recovery information periodically during failure-free execution. Upon a failure, a failed process uses the saved information to restart the computation from an intermediate state, thereby reducing the amount of lost computation. The recovery information includes, at a minimum, the states of the participating processes, called checkpoints. Other recovery protocols may require additional information, such as logs of the interactions with input and output devices, events that occur to each process, and messages exchanged among the process.

Message-passing systems complicate rollback-recovery because messages induce inter-process dependencies during failure-free operation. Upon a failure of one or more processes in a system, these dependencies may force some of the processes that did not fail to roll back, creating what is commonly called rollback propagation. Under some scenarios, rollback propagation may extend back to the initial state of the computation, losing all the work performed before a failure. This situation is known as the domino effect.

The domino effect may occur if each process takes its checkpoints independently—an approach known as independent or uncoordinated checkpointing. It is obviously desirable to avoid the domino effect and therefore several techniques have been developed to prevent it. One such technique is to perform coordinated checkpointing in which processes coordinate their checkpoints in order to save a system-wide consistent state [Chandy and Lamport 1985].

The entire process of reconfiguring communication, at the MH, wireless network and backbone wired network, is known as handoff. The objective of handoff is to maintain end-to-end connectivity in the dynamically reconfigured network topology. During a handoff, the route of data through the wired network to the MH must be updated to pass through this new BS. In addition any state in the old BS associated with the MH must somehow be transferred to the new BS and the radio communication may also need to be reconfigured.

2 PREVIOUS WORK

A. checkpoint-based rollback recovery

Upon a failure, checkpoint-based rollback recovery restores

- Anurag sachan is currently pursuing masters degree program in computer science and engineering engineering in Mahamaya Techinal University, India, PH-919990385346. E-mail: anuragsachan2@gmail.com
- Prachi Maheshwari is currently working as asst. prof. in computer science and engineering in Mahamaya Technical University, India, PH-919654207157. E-mail: prashi0110@gmail.com

the system state to the most recent consistent set of checkpoints, that is the recovery line It does not rely on the PWD assumption, and so does not need to detect, log, or replay nondeterministic events. Checkpoint-based protocols are therefore less restrictive and simpler to implement than log-based rollback-recovery. But checkpoint-based rollback-recovery does not guarantee that pre-failure execution can be deterministically regenerated after a rollback.

a. Uncoordinated Checkpointing

Uncoordinated checkpointing allows each process the maximum autonomy in deciding when to take checkpoints. The main advantage of this autonomy is that each process may take a checkpoint when it is most convenient. For example, a process may reduce the overhead by taking checkpoints when the amount of state information to be saved is small [Wang 1993]. But there are several disadvantages. First, there is the possibility of the domino effect, which may cause the loss of a large amount of useful work, possibly all the way back to the beginning of the computation. Second, a process may take a useless checkpoint that will never be part of a global consistent state.

b. Coordinated Checkpointing

Checkpointing requires processes to broadcast their checkpoints in order to form a consistent global state. Coordinated checkpointing simplifies recovery and is not susceptible to the domino effect, since every process always restarts from its most recent checkpoint. Also, coordinated checkpointing requires each process to maintain only one permanent checkpoint on stable storage, reducing storage overhead and eliminating the need for garbage collection. Its main disadvantage, however, is the large latency involved in committing output. A straightforward approach to coordinated checkpointing is to block communications while the checkpointing protocol executes.

c. Minimal Checkpoint Coordination

Coordinated checkpointing requires all processes to participate in every checkpoint. This requirement generates valid concerns about its scalability. It is desirable to reduce the number of processes involved in a coordinated checkpointing session. This can be done since the processes that need to take new checkpoints are only those that have communicated with the checkpoint initiator either directly or indirectly since the last checkpoint [Koo and Toueg 1987].

The following two-phase protocol achieves minimal checkpoint coordination [Koo and Toueg 1987]. During the first phase, the checkpoint initiator identifies all processes with which it has communicated since the last checkpoint and sends them a request. Upon receiving the request, each process in turn identifies all processes it has communicated with since the last checkpoints and sends them a request, and so on, until no more processes can be identified. During the second phase, all processes identified in the first phase take a checkpoint. The result is a consistent checkpoint that involves only the participating processes. In this protocol, after a process takes a checkpoint, it cannot send any message until the second phase terminates successfully, although receiving a message after the checkpoint has been taken is allowed.

d. Communication-induced Checkpointing

Communication induced checkpointing (CIC) protocols avoid the domino effect without requiring all checkpoints to be coordinated. In these protocols, processes take two kinds of checkpoints, local and forced. Local checkpoints can be taken independently, while forced checkpoint must be taken to guarantee the eventual progress of the recovery line. In particular, CIC protocols take forced checkpoints to prevent the creation of useless checkpoints, that is checkpoints that will never be part of a consistent global state. Useless checkpoints are not desirable because they do not contribute to the recovery of the system from failures, but they consume resources and cause performance overhead.

B. LOG-BASED ROLLBACK-RECOVERY

As opposed to checkpoint-based rollback recovery, log-based rollback-recovery makes explicit use of the fact that a process execution can be modeled as a sequence of deterministic state intervals, each starting with the execution of a nondeterministic event [Strom and Yemini 1985].

Log-based rollback-recovery assumes that all nondeterministic events can be identified and their corresponding determinants can be logged to stable storage. During failure-free operation, each process logs the determinants of all the nondeterministic events that it observes onto stable storage. Additionally, each process also takes checkpoints to reduce the extent of rollback during recovery. After a failure occurs, the failed processes recover by using the checkpoints and logged determinants to replay the corresponding nondeterministic events precisely as they occurred during the pre-failure execution.

a. Pessimistic Logging

Pessimistic logging protocols are designed under the assumption that a failure can occur after any nondeterministic event in the computation. This assumption is "pessimistic" since in reality, failures are rare. In their most straightforward form, pessimistic protocols log to stable storage, the determinant of each nondeterministic event before the event is allowed to affect the computation.

In a pessimistic logging system, the observable state of each process is always recoverable. This property has four advantages:

1. Processes can send messages to the outside world without running a special protocol.
2. Processes restart from their most recent checkpoint upon a failure, therefore limiting the extent of execution that has to be replayed.
3. Recovery is simplified because the effects of a failure are confined only to the processes that fail.
4. Garbage collection is simple.

b. Optimistic Logging

In optimistic logging protocols, processes log determinants asynchronously to stable storage [Strom and Yemini 1985]. These protocols make the optimistic assumption that logging will complete before a failure occurs. Determinants are kept in a volatile log, which is periodically flushed to stable storage. Thus, optimistic logging does not require the application to block waiting

for the determinants to be actually written to stable storage, and therefore incurs little overhead during failure-free execution. However, this advantage comes at the expense of more complicated recovery and garbage collection, and slower output commit, than in pessimistic logging.

c. Causal Logging

Causal logging has the failure-free performance advantages of optimistic logging while retaining most of the advantages of pessimistic logging [Alvisi 1996; Elnozahy 1993]. Like optimistic logging, it avoids synchronous access to stable storage except during output commit. Like pessimistic logging, it allows each process to commit output independently and never creates orphans, thereby isolating each process from the effects of failures that occur in other processes. Furthermore, causal logging limits the rollback of any failed process to the most recent checkpoint on stable storage. This reduces the storage overhead and the amount of work at risk. These advantages come at the expense of a more complex recovery protocol.

C. ADAPTIVE CHECKPOINTING

Our approach uses time & leasing to coordinate checkpoint creation adaptively and indirectly. It says that time can be used to implement coordinated checkpointing efficient. Our storage manager uses the leasing mechanism. A 3-level storage hierarchy is used to save checkpoints.

a. Checkpoint Creation

When the application begins, the protocol sets the checkpoint timers in all processes with a value equal to the chcckpoint interval. Whenever a timer expires, a. process takes a checkpoint and resets the timer. The protocol uses a simple re-synchronization mechanism to roughly synchronize the checkpoint intervals of the processes, even if drift rates of clocks are different.

b. Hierarchical Storage Management

The protocol uses a 3-level storage hierarchy to save checkpoints Checkpoints stored in level #1 called soft checkpoints (SC), they are saved in the mobile host. Level #2 is the stable storage available in the base stations, level #3 corresponds to the home host. Level #2 and level #3 are both referred to as hard checkpoints (HC). Soft checkpoints are less reliable than hard checkpoints because they will be lost if the mobile host fails permanently. Hard checkpoints can survive mobile-host permanent failures but have higher overheads since they must be transmitted through the wireless channels.

c. Hand-Off Procedures

Before moving to another cell, the process notifies the storage manager at the current base station. The manager then forwards the hard checkpoints of the process to the home host. After the checkpoint is saved safely by the home host, the checkpoint on the base station is removed. f the new cell provides storage service, and the process got a lease, then the hard checkpoints can alternatively sent to the new base station. This hand-off procedure simplifies garbage collection on base stations. When the mobile host leaves the current cell, the space occupied by its checkpoints becomes available for reallocation. This feature avoids having checkpoints scattered throughout the network while the mobile host moves around. The mobile host also does not have to maintain extra links to locate previous checkpoints.

3 PROBLEM FORMULATION

MHs move randomly causing handoff. Hence checkpoints of a single MH are scattered in different MSSs. If an MH fails, recovery information are to be collected from different MSSs. If di tance between MSSrecovery and the MSSstart is large then recovery time will be more. In region based recovery scheme, region manager is the centralized control. Hence it may get overloaded. Moreover if it fails, recovery is not possible.

4 ALGORITHM AND IMPLEMENTATION

ALGORITHM:

In first step connection is initiated and MH stores the id of current MSS.
A link is created in the current MSS structure
CurrentMSSid field of MH is updated to contain the current MSSid.
Take a checkpoint of present state of the MH.

HANDOFF PART
New connection is setup as soon as MH get attached to it.
A link is created in the MSS structure which contains the MH after it attaches to it.
CurrentMSSid field of MH is updated to contain the present MSSid.
Hcount = Hcount + 1

Here decision is taken on the basis of predefined conditions on parameters. On success checkpoint is initiated.
if(Hcount > Handoff Threshold or Distance Threshold
between MSS > Threshold value)&&(RSS>RSS_threshold)
Take a checkpoint of present state of the MH.
Hcount = 0 .
Update the Number of the MH.
Update the checkpoint_location with the current MSS_id.

If RSS condition is not met, RSS is incremented to avoid the condition of starvation of a MH due to present weak strength of received signal.

A coordinated approach is followed to make system more Lossless.
Check the dependency list of the MH.
Send checkpoint requests to all MHs that are in the dependency list.
After that take a checkpoint of the present state of MH.

If any MH doesn't reply in accordance with sender, it means that may be executing some process. The sender waits for a

defined limit.
if(difference of current time and previous checkpoint time > Threshold value)
Take checkpoint.

A log file is created for making to support recovery.
Open the current MSS's log file.
Sequence_number = Sequence_number + 1.
Save data, MH_id and number of the MH in the log file.

For saving space, the unnecessary information is cleared time by time.
if(coordinated checkpoint is been taken) Clear dependency list.
Save the end checkpoint timestamp in local buffer.
Update the Tstamp to reflect the current time.

MH RECOVERY PART
First of all a request is done.
Get the MSS_id from the checkpoint_location field of MH where the latest checkpoint was taken.
Send the Seq_number and the checkpoint_location values to the current MSS.
Request current MSS to send these values to the checkpoint_location MSS.

With the help of checkpoint_location MSS log file is accessed.
Select the log entry corresponding to the MH_id and Seq_number provided.
Retrieve the data from this log file entry.
Send the data to the MH via the current MSS get its last stable state back.

IMPLEMENTATION CODE ON MATLAB

Best Distance Calculating Function

```
function best_node_inx = best_dist(pnt1,nodes)

for i = 1:length(nodes)
   x1 = nodes(i).x;
   y1 = nodes(i).y;
   pnt2 = [x1 y1];
   dist_val(i,1) = node_distance(pnt1,pnt2);
end

[sval sinx] = sort(dist_val);

best_node_inx = sinx(1);
```

Mobile Host Representation By Circle

```
function H=circle(center,radius,NOP,style)

if (nargin <3),
 error('Please see help for INPUT DATA.');
elseif (nargin==3)
   style='b-';
end;
THETA=linspace(0,2*pi,NOP);
RHO=ones(1,NOP)*radius;
[X,Y] = pol2cart(THETA,RHO);
X=X+center(1);
Y=Y+center(2);
H=plot(X,Y,style);
% axis square;
```

Failure Free Operation Cost

```
Function oper_cost = failure_free_operation_cost(hT,Cmg_ch,Cmg_ch_transfer,Cper_ch)

oper_cost = (hT * Cmg_ch) + (hT * Cmg_ch_transfer) + Cper_ch;
```

Recovery Cost

```
function recov_cost = Recovery_Cost(hT,Cmg_ch_transfer,Cper_ch_transfer)

recov_cost = hT * Cmg_ch_transfer + Cper_ch_transfer;
```

Handoff Procedure and Checkpoints in Network

```
clc
clear all
close all
warning off

net_width = 10;
net_height = 80;
check = 1;
tot_nodes = 20;
traves_MS = 10;
hT = 4;


% Form Network area....
node_count = 1;
all_x = [];
all_y = [];
for j = 0:9:net_height
    if check==1
        start = 0;
        check = 0;
    else
        start = 5;
        check = 1;
    end
    for i = start:10.3:net_height
        circle([j,i],6,7,'-');
        hold on
        drawnow
        plot(j,i,j,i,'.','color','green');
        nodes(node_count).x = j;
        nodes(node_count).y = i;
        nodes(node_count).id = node_count;
        node_count = node_count+1;
        all_x = [all_x;i];
        all_y = [all_y;j];
```

```
    end
end
axis square;

min_x = min(all_x);
max_x = max(all_x);

min_y = min(all_y);
max_y = max(all_y);


% Making Random nodes....
for i = 1:tot_nodes
    x = round(min_x+(max_x-min_x)*rand);
    y = round(min_y+(max_y-min_y)*rand);
    pnt1 = [x y] ;
    best_node_inx = best_dist(pnt1,nodes);
    best_x = nodes(best_node_inx).x;
    best_y = nodes(best_node_inx).y;
    best_pnt = [best_x best_y];

    plot(x,y,x,y,'*','color','red');
    all_x = [pnt1(1) best_pnt(1)];
    all_y = [pnt1(2) best_pnt(2)];
    plot(all_x,all_y);
    hold on
    drawnow
end



% Making Moving Nodes...
s_min_x = min_x+0.5;
s_min_y = min_y+0.5;
con_x = 1;
check = 1;
totms = 0;
traverse_node = [];

station_count = [];
station_data = [];
sel_nodes = [];
last_check_pnt = 1;
while(totms~=traves_MS)

    if s_min_x<=max_x&&check == 1
        check = 1;
    else
        check = 0;
        if s_min_x<=min_x && check == 0
            check = 1;
        end
    end
    if check ==1
        s_min_x = s_min_x+1;
        s_min_y = s_min_y+1;
    else
        s_min_x = s_min_x-1;
        s_min_y = s_min_y-1;
    end


    h1                                      =
plot(s_min_x,s_min_y,s_min_x,s_min_y,'*','c
olor','red');
    pnt1 = [s_min_x s_min_y] ;
    best_node_inx = best_dist(pnt1,nodes);
    best_x = nodes(best_node_inx).x;
    best_y = nodes(best_node_inx).y;
    best_pnt = [best_x best_y];
    hold on
    drawnow

%     checkpoint calculation....
    best_node_id = nodes(best_node_inx).id;
    if isempty(station_count)
        station_count = station_count+1;
        station_data     =     [station_data
best_node_id];
        traverse_node    =    [traverse_node
best_node_id];
        nodes(best_node_inx).data   =   sta-
tion_data;
        station_count                       =
length(traverse_node);
        sel_nodes        =        [sel_nodes
best_node_id];
        display(['Mobile  Node  Travel  on  '
num2str(traverse_node(station_count))  ' Mo-
bile Station....']);
    elseif                                tra-
verse_node(station_count)~=best_node_id
        station_count = station_count+1;
        station_data     =     [station_data
best_node_id];
        traverse_node    =    [traverse_node
best_node_id];
        nodes(best_node_inx).data   =   sta-
tion_data;
        sel_nodes        =        [sel_nodes
best_node_id];
        display(['Mobile  Node  Travel  on  '
num2str(traverse_node(station_count))  ' Mo-
bile Station....']);
%         permanent check point saving...
        if rem(station_count,hT)==0
            nodes(best_node_inx).data      =
station_data;
            last_check_pnt = best_node_inx;
            display('Saving  Data  on  Check
point...');
%             remove temp datas...
            for k = 1:length(sel_nodes)-1
                nodes(sel_nodes(k)).data   =
'';
            end
```

```
            sel_nodes      =      [sel_nodes
best_node_id];
        end
    end
    totms = station_count;

    all_x = [pnt1(1) best_pnt(1)];
    all_y = [pnt1(2) best_pnt(2)];
    h2 = plot(all_x,all_y);
    pause(0.1);
    delete(h1);
    delete(h2);
end



display(['Data       Faild       on       '
num2str(traverse_node(station_count)) ' Mo-
bile Station ...']);

% data revover from last check point...

chk_data = nodes(last_check_pnt).data;
chk_x = nodes(last_check_pnt).x;
chk_y = nodes(last_check_pnt).y;
all_x = [pnt1(1) chk_x];
all_y = [pnt1(2) chk_y];
plot(s_min_x,s_min_y,s_min_x,s_min_y,'*','c
olor','red');
for i = 1:10
    h = plot(all_x,all_y,'color','red');
    pause(0.1);
    delete(h);
    pause(0.1);
end
plot(all_x,all_y);
h  =  waitbar(0,['Data  Recover  From  Mobile
Station      -      Check      Point      '
num2str(last_check_pnt) ' ...']) ;
for i = 1:100
    pause(0.01);
    waitbar(i/100);
end
close(h);



% Result for graphs...

Cmg_ch = 1;
Cmg_ch_transfer = 1;
Cper_ch = hT;
Cper_ch_transfer = 1;
oper_cost               =               fail-
ure_free_operation_cost(hT,Cmg_ch,Cmg_ch_tr
ansfer,Cper_ch);

for i = 1:traves_MS
    recov_cost(i,1)         =         Recov-
ery_Cost(hT,Cmg_ch_transfer,Cper_ch_transfe
r);
    recov_cost_old(i,1)         =         re-
cov_cost(i,1)+(i-1);
end
```

4 SIMULATION AND PERFORMANCE ANALYSIS

Simulation of nodes is performed having handoff count(ht), total MSSs traversed(n) and speed(v) of the Mobile Host.Then on the simulation part, main handoff and checkpoint procedure code is embedded which can be viewed on the simulation window.

The simulation is possible for varying values of taken parameters.The results are plotted where the difference between region based and proposed approach can be observed.As for example, we can take some values of these parameter like ht, v and n. Let ht=3 and n=16 and v=4 thenaccording to proposed approach after every 3 MSS traversal the MS saves a checkpoint at that MSS.

Total cost = ht*migration checkpoint transfer cost + permanent checkpoint transfer cost of (i-1) interval

= k1+k2

Which will be a contant value.

But in case of region based approachs

Total cost = no. of MSS traversed*migration checkpoint transfer cost+ permanent checkpoint transfer cost

Which will be linearly increasing function of no.of MSSs.

6 CONCLUSION

Proposed recovery technique focuses on faster and efficient recovery information dispersed over a number of MSSs due to random movement of MHs. Our work limits the number of MSSs from which recovery information of a failed image to be collected. Our work also reduces number of communication messages, a significant overhead reduction in mobile computing sy tems. Recovery cost with failure is optimized. Failure free operation cost is also optimized because the recovery scheme described here works parallel table normal execution of mobile hosts without causing any blocking or delay effect.

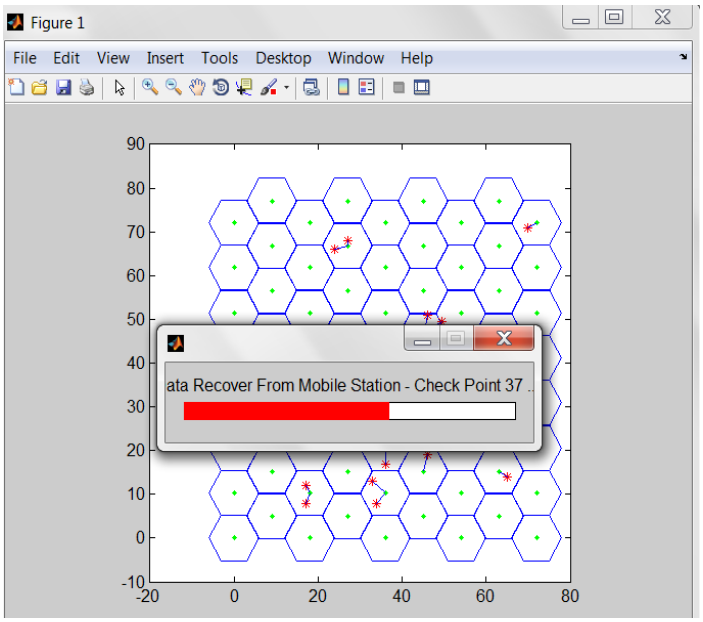
7 FUTURE SCOPE AND APPLICATION IN HANDOFF BASED SCHEME

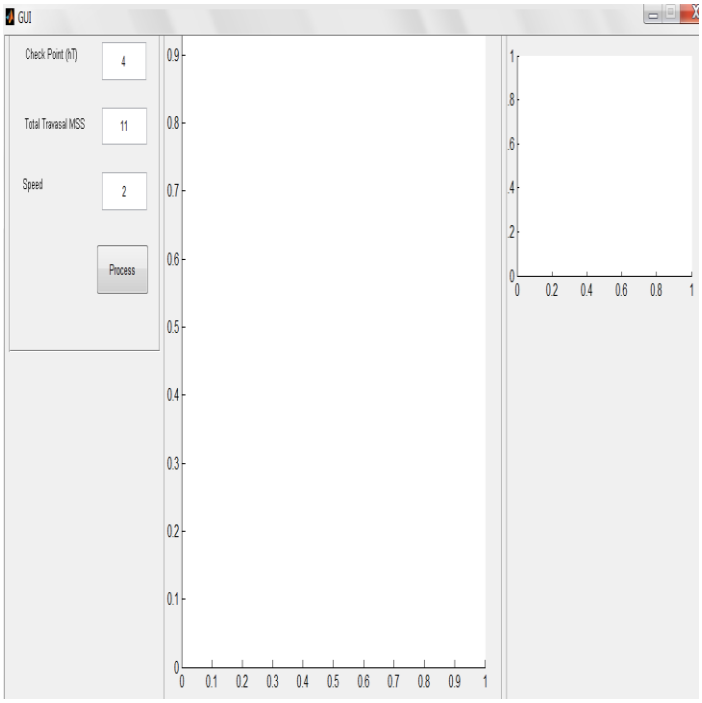
In next-generation wireless systems, it is important for Radio Resource Management (RRM) functionality to ensure that the system is not overloaded and guaranteeing the needed requirements. If the system is not properly planned, it leads to low capacity than required and the QOS degraded. The system became overloaded and unstable. Therefore, we need an algorithm which not only adapts itself according to the Received Signal Strength (RSS) but also on the load status of target station. In addition, the current works do not consider the load status of neighbouring

cell. This algorithm can be enhanced to differently set the handoff threshold also based on traffic of cells.

References

1. A book by Jochen Schiller, Mobile Communication
2. R.Koo, S.Toueg ”Checkpointing and Rollback-Recovery for Distributed Systems”, IEEE Transactions on software Engineering, 1987
3. Ravi Prakash and Mukesh Singhal **“**Maximal Global Snapshot with Concurrent Initiators” IEEE Transactions,pp. 344-351, 1994
4. Ravi Prakash, Michel Raynal, Mukesh Singhal “An Efficient Casual Ordering Algorithm for Mobile Computing “ IEEE Proceedings of the 16th ICDCS, pp. 744-751, 1996
5. Ravi Prakash and Mukesh Singhal” Low Cost Checkpointing and Failure Recovery in Mobile Computing Systems” IEEE Transactions on Parallel and Distributed Systems Vol. 7, NO. 10**,** October 1996
6. Ravi Prakash and Mukesh Singhal ”Modeling and Analysis of Channel Transferability in Mobile Computing Environments”, IEEE Transactions, pp. 198-205,1996
7. Guohong Cao, Mukesh Singhal, “Low-Cost Checkpointing with Mutable Checkpoints in Mobile Computing Systems”, 18th International Conference on Distributed Computing Systems, 1998
8. Guohong Cao and Mukesh Singhal “Efficient Distributed channel Allocation for Mobile Cellular Networks” IEEE Transactions, pp. 50-56, 1998
9. C. Guohong and M. Singhal, “On Coordinated Checkpointing in Distributed Systems”, IEEE Transactions on Parallel and Distributed Systems ,Volume 9 , Issue 12,pp. 1213 – 1225, ISSN:1045-9219, December 1998
10. T.Park, N.Woo, and H.Y.Yeon “A Region- based recovery Information Management Scheme for the Fault Tolerant Mobile Computing Systems ” Symp. on Fault Tolerant Computing Systems, pp. 294-301, Jun. 1999
11. E .N Elnozahy, L. Alvisi., W. David and J. David “A Survey of Rollback-Recovery Protocols in Message-Passing Systems”, Computing Surveys (CSUR), v.34 n.3, p.375-408, September 2002
12. Bidyut Gupta, Shahram Rahimi, and Ziping Liu “ A New High Performance Checkpointing Approach for Mobile Computing Systems” IJCSNS International Journal of Computer Science and Network Security, VOL.6 No.5B, May 2006
13. Sapna E. George,Ing-Ray Chen,Ying Jin, “Movement-Based Checkpointing and Logging for Recovery in Mobile Computing Systems”, 2006
14. S Biswas Saha, S Neogy "A Low Overhead Checkpointing Scheme for Mobile ComputingSystems," 15th International Conference on Advanced Computing and Communications, 2007
15. Rachit Garg and Praveen Kumar “A Review of Fault Tolerant Checkpointing Protocols for Mobile Computing Systems” International Journal of Computer Applications(0975-8887),Volume 3 – No.2, June 2010





Analysis Graphs