http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 12 (14): 2690-2696, 2013
ISSN 1812-5638 / DOI: 10.3923/itj.2013.2690.2696

Central Causal Logging Rollback Recovery Scheme for Mobile Computing

[1]Zhenpeng Xu, [2]Yu liu and [1]Weiwei Li
[1]Jiangsu Automation Research Institute, No.18, RD.Shenghu, Xinpu District, Lianyungang, Jiangsu, China
[2]Department of Computer, Weifang University, NO. 5147, RD. East Dongfeng, Weifang, Shandong, China

Abstract: Checkpointing and rollback-recovery is one of the most important techniques for fault-tolerance in mobile computing. Different from the traditional wired distributed computing, mobile computing puts forth some new high requirements for fault tolerance. Therefore, various log-based fault tolerant schemes were proposed to accommodate its characteristics. However, these schemes may still lead to dramatic loss of system performance or inconsistent recovery. In this paper, a central causal logging rollback recovery scheme is proposed combining checkpointing with message logging based on perfect PWD assumption. The checkpoint, message logs and happened-before relations are all recorded in the antecedence graph maintained by the local mobile support station. The antecedence graph is logged synchronously into volatile memory and asynchronously into the persistent storage upon the specific event and the proposal supports the independent consistent recovery with the complete log and propagated consistent recovery without the complete log. The performance analysis shows that the proposal incurs a low failure-free overhead.

Key words: Mobile computing, fault tolerant, checkpoint, rollback

INTRODUCTION

In log-based rollback recovery schemes, the events experienced is also recorded into a location that will survive the process fault. That action is called logging (Chen *et al*., 2005; Li *et al*., 2005; Wang and Shao, 2003; Yang *et al*., 2006).

Many new characteristics are introduced in mobile computing, such as mobility, disconnections, finite power source, vulnerable to physical damage, lack of stable storage (Brzezinski *et al*., 2006; Elnozahy *et al*., 2002; Gupta *et al*., 2008; Li and Wang, 2005; Ono *et al*., 2004; Pradhan *et al*., 1996). Therefore, the wireless network connection is more fragile and mobile host is much less reliable than the traditional wired distributed computing. Mobile hosts may disconnect from the rest of the network due to doze mode, abrupt power off or permanents damage.

Research on rollback recovery fault tolerant scheme for mobile computing systems has received tremendous interests in recent years. Various schemes have been presented to accommodate the characteristics of mobile computing (Cao and Singhal, 2001; Men *et al*., 2006; Park *et al*., 2002; 2003b). However, how to minimize the failure-free overhead incurred and to ensure the consistent recoverability, requires to be investigated further.

PRELIMINARY

As shown in Figure 1, mobile computing system, MCS = <N, C>contains a set of nodes N and a set of channels C. The set of nodes N = M?S can be divided into two types, M={MH_1, MH_2, ..., MH_n} is the set of Mobile Hosts (MH), which are able to move around while retaining wireless network connections, while S = {MSS_1, MSS_2, ..., MSS_m} is the set of static nodes acting as the

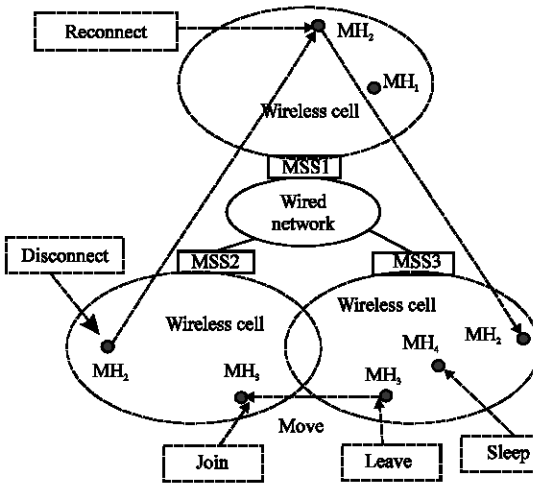

Fig. 1: Mobile computing system

Corresponding Author: Zhenpeng Xu, Jiangsu Automation Research Institute, No.18, RD.Shenghu, Xinpu District, Lianyungang, Jiangsu, China

access point with extra processing power and storage capabilities, static backbone node named Mobile Support Station (MSS) (Elnozahy *et al*., 2002; Gupta *et al*., 2008). The set of channels C = W?W' can also be divided into two disjoint sets, the set of high-speed wired channels W and the set of low bandwidth wireless channels W'. W = S×S, that is the static MSSs are connected by wired channels, while W' = S×M, means that MHs communicate with the network through the wireless channel existing between them and MSSs, by using a wireless LAN protocol like IEEE 802.11 (Ono *et al*., 2004). All the channels W and W' provide reliable, sequenced FIFO delivery of messages, with finite but arbitrary message latency (Gupta *et al*., 2008). A cell is a geographical area covered by the transceiver of a MSS. A MH within the radio range of the cell of MSS_p can directly communicate with MSS_p through the wireless channel. Logically a MH belongs to only one cell at any given instant. This implication can be drawn by assuming that the geographical cells around each of MSS do not overlap.

The cell of MSS_p, $Cl_p = \{MH_i \mid MH_i \in MSS_p, 0<i<n+1\}$ is the set of the active nodes and sleeping nodes identified by $Active\_MH\_List_p$ and $Disconnected\_MH\_List_p$ respectively, then there exists a channel $<MSS_p, MH_i> \in W'$ only if $MH_i \in CL_p \Rightarrow MH_i \notin CL_q$, $\forall q \neq p$. Due to the mobility, an active MH can move freely from one cell to another and the local MSS responsible for the MH is changed correspondingly. This process is called handoff.

For simplicity, only one process running on a MH is assumed. Thus, the terms 'MH' and 'process' can be used interchangeably. A MCS consists of a set of N processes denoted by P_1, P_2,... , P_n. Processes do not share a global memory or a global physical clock. To cooperate to complete a distributed computation, these distributed processes communicate each other only through exchanging computational messages and interact with the outside world only through input/ output commit (Elnozahy *et al*., 2002).

The execution of the process is assumed to follow Piece-wise Deterministic (PWD) model, in which a process experiences a sequence of state transitions for its execution and the atomic action caused by the event experienced. The underlying computation is asynchronous (Elnozahy *et al*., 2002). PWD states that a process always produces the same sequence of states in its execution for the same sequence of events and the system can detect and capture sufficient information about the nondeterministic events that initiate the state intervals. The event experienced by processes can be classified into two categories: nondeterministic event and deterministic event.

The transient independent fault of the process follows fail-stop model (Elnozahy *et al*., 2002). Upon a fault, the process is able to stop its execution and does not perform any malicious action.

RELATED WORKS

Commonly, the fault tolerant techniques for mobile computing, that do not require user interaction can be classified into two categories: checkpoint- based and log-based rollback recovery scheme (Elnozahy *et al*., 2002). So many coordinated checkpoint-based variations for fault tolerance are proposed to reduce the number of participating process required to checkpoint, overhead in terms of synchronization messages required to coordinate and avoid the domino effect (Li and Wang, 2005). However, an abrupt disconnection or doze mode of a single MH may lead to fail to achieve the entire coordinated checkpointing or rollback recovery in mobile computing. Therefore, log-based rollback recovery approach based on independent checkpointing, is preferable to checkpoint-based approach for fault tolerance of mobile computing (Gupta *et al*., 2008).

Log-based rollback recovery scheme combines independent checkpointing with message logging techniques. Depending on how the determinants are logged to stable storage, log-based rollback recovery scheme is classified into three types: pessimistic logging, optimistic logging and causal logging (Elnozahy *et al*., 2002).

In pessimistic logging (Park *et al*., 2003b), the process has to block waiting for the determinant of each nondeterministic event to be stored on stable storage before the effects of that event can be seen by other processes or the outside world. Pessimistic logging simplifies recovery and garbage collection but hurts failure-free performance due to synchronous logging.

In optimistic logging (Park *et al*., 2002), the process does not block and determinants are transferred to stable storage asynchronously. Thus, optimistic logging does not require the application to block waiting for the determinants to be actually written to stable storage and only record the nondeterministic event and therefore incurs little overhead during failure-free execution. However, this advantage comes at the expense of more complicated recovery, garbage collection and propagated rollback. In case of some loss of volatile logs, it may lead to unrecoverable rollback without well consideration of the input/output commit problem, as the input/output devices that cannot roll back.

Causal logging in (Zhang *et al*., 2008) satisfies always-no-orphans property by ensuring that the

determinant of each nondeterministic event that causally precedes the state of a process is either stable or it is available locally to that process (Alvisi and Marzullo, 1998). Therefore, it gets a balance between optimistic and pessimistic logging to maintain the dependency relations through each common computing message. However, many records of the dependency relations require to be exchanged among the process. In practice, carrying the dependency relation graph information on each application message may lead to an unacceptable overhead, since each MH corresponds to one antecedence graph.

CAUSAL LOGGING ROLLBACK-RECOVERY

The proposed central causal logging scheme is based on message logging, independent checkpointing and PWD assumption. Focusing on the logging rollback recovery fault tolerant scheme for mobile computing, the handoff management is ignored in this paper. In our model, the nondeterministic event of a process includes the message and input receipt from the other processes or outside world, while the message sending, output and internal events are assumed as the deterministic event of a process.

Determinant and related denotations: To implement the proposal, for each message m experienced by the process, the tuple ⟨type, source, dest, ssn, dsn, data⟩, is the determinant of message m. Specifically, m.source, m.dest and m.ssn denote the identifiers of the sending, receiving process and the unique identifier assigned to m by the sender respectively. The latter, m.ssn may be just a sequence number. m.data is the content carried by message m. m.type records the type of m and m.type∈{Normal, Checkpoint, Input, Output, Connecting, Recovery}. 'Normal' indicates that m is common communicating message; 'Input' indicates that an input from outside world is packed in m; 'Output' indicates that an output commit to outside world packed in m; 'Connecting' indicates that m is a connecting or disconnecting request; 'Recovery' indicates that m is a recovery request; and 'Checkpoint' indicates that a checkpoint of a process is recorded in m. The deliver sequence number, m.dsn encodes the order in which m is delivered by the receiving process. Thus, if process P_i has delivered m and m.dsn =θ, then m is the θth event that P_i has delivered. For simplicity, we refer to #m the determinant of m. This tuple #m determines message m and related information, which is useful for the rollback recovery phase.

Antecedence graph: To record the process state and happened-before relation exactly, the antecedence graph is introduced based on the state transition advanced by the nondeterministic message. Each MSS maintains only one antecedence graph for all the MHs in the local cell to support the rollback recovery. The antecedence graph contains a summary of the local cell's execution. The antecedence graph <E, V>, contains the node set V including the logged determinant of key messages experienced by each process and the edge set E representing happened-before relations among the nodes in V.

Specifically, in the antecedence graph, each node σ corresponds to one or more logged determinants (σ∈V). Let σ_θ^i indicates the θth node of P_i in the antecedence graph. The property of a graph node depends on the type of the corresponding recorded determinant of the message (Gupta *et al*., 2008).

Different from the antecedence graph in Zhang *et al*. (2008), only one entire antecedence graph of all the local MHs is managed and maintained by MSS. Furthermore, the checkpoint and message logs are also recorded in the graph node set V, in addition to the happened-before relations recorded in the edge set E. That means no dependency relations require to be exchanged among the process during failure-free execution.

Checkpointing and input/output: For checkpointing, the local checkpointable interface is inhabited at each MH. P_i takes a checkpoint with a fixed interval. Let C(i,α) denotes the αth checkpoint of P_i at MH_i.

The process takes periodic checkpoint to limit the amount of work that has to be repeated in execution replay upon recovery. The checkpoint period between two consecutive checkpointing of P_i is determined by itself. That means the process takes local checkpoints asynchronously.

Time to take checkpoint, the replicated process state of P_i is created by MH_i using the checkpointing operation of the local checkpointable interface. The new checkpoint is encapsulated into a message and transferred to the local connected MSS_p for logging.

Upon a user input from outside world, a copy of the input is firstly encapsulated and forwarded to the local MSS_p for logging. On receipt of the acknowledgment from MSS_p, MH_i starts to process the input. Similarly, before interacting with the outside world to show the outcome of, a copy of the outcome is also forwarded to MSS_p for logging firstly. The pseudo-code of the procedure Checkpointing() is shown in Fig. 2.

```
/* when the checkpoint peropd pf MH_i expires */
Procdure checkpointing ()
/* saves the state of P_i as a checkpoint. */
task a new checkpoint C(i, α);
send C(i, α) to the local connected MMS_p;
```

Fig. 2: The checkpointing

Central causal logging: Since MSS is on the border between wireless and wired network in mobile computing. All the messages to and from the local MHs are traversed through its connected MSS locally. Thus, the storage available at MSSs is employed to save the entire antecedence graph on behalf of the local mobile hosts.

Each MSS_p manages and maintains a volatile version antecedence graph AG_T_p, to record the logs and happened-before relations in the local cell on behalf of the local mobile hosts. The message determinants and checkpoints of each local MH are positioned logically in the antecedence graph AG_T_p, according to the happened-before relations.

Considering the reliability of the volatile logs stored in the volatile memory or cache at MSS, the volatile version antecedence graph, AG_T_p, requires to be updated into the persistent version, AG_P_p, upon the specific events, to ensure the consistent recoverability and free the volatile memory space of MSS_p. The persistent version AG_P_p, is recorded on stable storage for surviving failures. The specific special events include the receipt of the checkpoint, input, output, connecting message from any local MH. It is also assumed that updating the volatile version is much faster than updating the persistent one. Thus, the volatile space of a MSS is the preferred storage for recording the antecedence graph and performance considerations prohibit updating the persistent version without the occurrence of the specific event.

The proposed central causal logging scheme in pseudo-code is described in Fig. 3. During the normal failure-free execution, each message m, received or relayed by the local MSS_p, is passed to the logging mechanism of MSS_p, packed into the determinant #m, through Message_logging (m).

If m is a nondeterministic message ($m.dest \in CL_p$), a new graph node is created and added into the local antecedence graph AG_T_p at local MSS_p through Create_node_AG_T (AG_T_p, #m).

If m is a deterministic message ($m.source \in CL_p$ and and $m.dest \notin CL_p$), no new graph node is created, but #m is appended to the latest node σ of $P_{m.source}$ in the local antecedence graph AG_T_p, through procedure Update_AG_P (AG_T_p, #m).

```
Procedure message_loffing (m)
/* when MMSp receives a message m*/
identify the related values of #m;
pack the determinant #m;
if (m.source∈CL,&& m.dest∉CL_p)
{update_AG_p (AG_T_p, #m;}
if (m.dest∈CL_p)
{invoke create_node_AG_T (AG_T_p, #m);
rely m to the destination m dest;
if (m.type∈{checkpoint, input, output, connecting})
{incoke presistent_AG_P (AG_T_p, AG_P_p);
if m.type∈{checkpoint}
invoke Grabaege_collection (AG_T_p);
}
```

Fig. 3: The pseudo-code of the causal logging

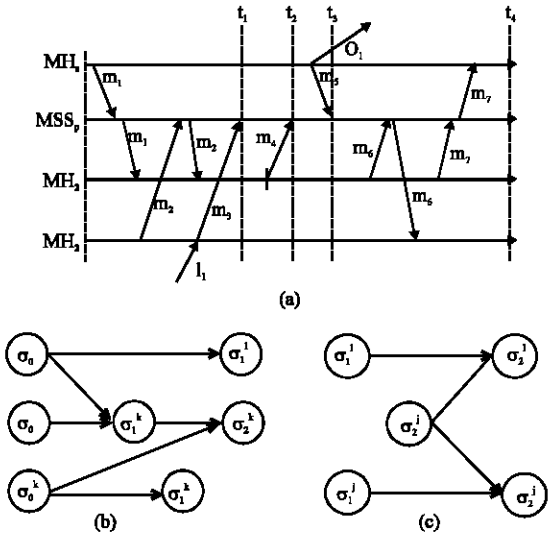

Fig. 4: An example of the antecedence graph

If m is the specific event with $m.type \in$ {checkpoint, input, output, connecting}, the volatile version antecedence graph, AG_T_p, is updated into the persistent version, AG_P_p, through Persistent_AG_P (AG_T_p, AG_P_p). Furthermore, if the content of m is a new checkpoint of a local process, the procedure Garbage_Collection (AG_T_p) is invoked to delete the nodes in AG_T_p, happened-before the new checkpoint to free the volatile memory space.

Figure 4a presents a simple scene of mobile computing in which all the messages to and from the local MHs are traversed through its connected MSS locally.

The simple system contains three mobile hosts, MH_1, MH_2 and MH_3. All the MHs reside in the geographical cell of MSS_p, CL_p. In practice, a typical system may contain many such components of MH and MSS.

According to the proposed central causal logging scheme, the copy of P_k's input I_1 in m_3, the checkpoint of P_j in m_4 and the copy of output O_1 in m_5, are received by

```
/* when MSS0 recevives a recovery request from MH */
Procefure extract_AG_Fr (MHi)
If (AG_Tp∪AG_Pp is complete)
extract AG_Fr from AG_Tp∪AG_Pp;
else
extract AG_Fr from AG_Pp and other MSSs;
broadcast AG_Fr to all the MH in the local cell;
/* when MHi receives AG_Fr from the local MSS0. */
Procedure Rollback_recovery (AG_Fr)
reload the checkpont in AG_Fr;
replay the logs in partial order in AG_Fr
```

Fig. 5: The recovery algorithm

the local MSS_p for logging at time t_1, t_2 and t_3, respectively, as shown in Fig. 4a.

According to the proposal, Fig. 4b and c show the recorded volatile and persistent antecedence graph of a cell at time t_4, at MSS_p. Specifically, in persistent AG_P_p, node σ_2^i records $\#m_7$ while node σ_2^j records $\#m_5$. In volatile AG_T_p, node σ_2^i records $\#m_7$, while node σ_2^j records $\#m_2$, $\#m_4$, $\#m_6$ and $\#m_7$.

Recovery: During the recovery, the logged messages in the antecedence graph require to be replayed in the original irreflexive partial order.

For process P_i, a complete log consists of the latest checkpoint and logged determinants of all the experienced nondeterministic events after the checkpoint. The proposed central causal logging scheme supports independent rollback-recovery with the complete log and propagated rollback-recovery without the complete log.

Consider loss of the volatile AG_T_p, the rollback may require to be propagated to all the MHs in the local cell for consistent recoverability. Without the complete log, the lost nondeterministic messages in AG_T_p, the across-cell messages, can be retrieved from the antecedence graphs at other MSSs, since the proposal the antecedence graphs also records the local deterministic sending events, which is the nondeterministic events for the failure destination MH in other cell. Under PWD assumption, the lost computation within the cell can be deterministically recreated during the recovery.

The recovery algorithm in pseudo-code is described in Fig. 5.

Upon a process fault, MH_i sends a recovery request, to its local MSS_p. Upon the receipt of the recovery request, Extract_AG_F_r (MH_i) is invoked at MSS_p, in which the sub-antecedence graph AG_F_r is extracted from volatile AG_T_p and persistent AG_P_p. if AG_T_p is unavailable, AG_F_r is required retrieving from the antecedence graphs at other MSSs combining with AG_T_p and the rollback has to be propagated to all the MHs in the local cell for consistent recoverability. At last, AG_F_r is sent or broadcasted to the rollback MH. AG_F_r contains the latest checkpoint, nondeterministic message logs and related happened-before relations, which are useful to recover the failure process.

Upon the receipt of the extracted AG_F_r, Rollback_recovery(AG_F_r) is invoked at the rollback MH. According to AG_F_r, the process starts to recover the lost computation, through loading the checkpoint and replaying nondeterministic message logs in the original irreflexive partial order. During the rollback recovery, new message sent to the rollback MH is also recorded in the antecedence graph at its local MSS_p and will be forwarded to the rollback MH, in order, after the recovery.

Garbage collection: The central causal logging rollback recovery scheme establishes a recovery line beyond which no rollback is necessary. Therefore, recovery information describing events that occurred before the latest checkpoint is discarded, since it is not needed by the recovery scheme. This information includes the nodes of the antecedence graph belonging to events that occurred before the checkpoint, message sent and received before the checkpoint and older checkpoints. Therefore, in the procedure of Garbage_Collection (AG_T_p), the recovery information happened-before the new checkpoint is discarded automatically.

PERFORMANCE

For performance analysis, the metrics specified in (Alvisi and Marzullo, 1998) are summarized. The proposed central causal logging rollback recovery satisfies the optimal definition according to the metrics specified, since the entire antecedence graph is centralized managed and maintained in the local MSS, rather than each MH corresponding to one antecedence graph. Furthermore, the proposal ensures consistent recoverability of the failure process, since it maintains stable information about the external interactions with the outside world specifically, different from the internal interactions among processes.

Park *et al.* (2002) also satisfies the optimal definition according to the metrics specified (Alvisi and Marzullo, 1998), since it does not block the application totally and all the logged determinants are transferred to stable storage asynchronously in a fixed period. However, optimistic logging may lead to an inconsistent global state when the logged determinant of the input or output in the volatile log-space has been lost. Therefore, optimistic logging is ignored in following performance analysis.

A simple mobile computing system with 10 circular cells was simulated through the improved GloMoSim 2.02.

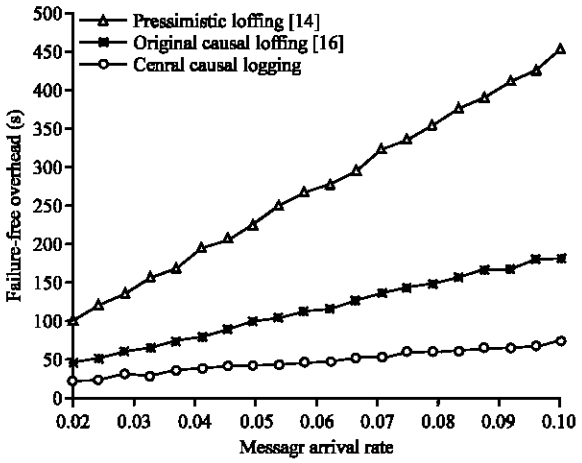

Fig. 6: Average logging overhead incurred

The sequence of the discrete event is generated based on the system parameter model (Pradhan *et al*., 1996). The fixed checkpoint interval of MH_i is T_i. Log arrival rate of MH_i follows a Poisson process with rate λ_i per time unit. Let α denote the ratio of the number of input/output to the total experienced messages. Let β denote the ratio of the number of connecting message to the total messages. Let θ denote the ratio of the size of a checkpoint to a normal messages.

The average logging overhead during the failure-free incurred by various logging schemes with the varying message rate, is shown in Fig. 6, where α = 3%, β = 2.5% and θ = 30. The y-axis indicates the average logging overhead incurred by access to volatile and stable storage for logging and corresponding operations, while the x-axis denotes the message arrival rate of a process.

As shown in Fig. 6, the proposed central causal logging scheme incurs a lower average logging overhead than pessimistic logging and original causal logging. The reason is that proposed central causal logging does not require the application to block but in case of the specific event, which reduced the number of stable storage accessing. However, pessimistic logging requires the determinant of each message to be logged synchronously into the stable storage and leads to a high overhead of frequent connection to the stable storage and relay blocking. In the original causal logging, much overhead is incurred by updating dependency relations among the antecedence graphs of mobile hosts, since each mobile host holds one antecedence graph.

CONCLUSION

A central causal logging rollback recovery scheme is proposed. In the local cell, the checkpoint of the computing process and all the nondeterministic messages, to be executed by each process, are encoded in the tuple called determinants, recorded in one entire antecedence graph maintained by the serving mobile support station. Considering the overhead incurred by the logging during failure-free execution, each determinant is first logged in the volatile version synchronously. The volatile version is asynchronously transferred to the stable storage upon the specific event. The proposed logging scheme enables the recoverability of the failure process by the independent rollback style with the complete log and by the cell propagated rollback style without the complete log. The performance of the proposal is evaluated by the extensive simulation. By contrast, the results show that the proposal incurs a lower failure-free overhead on the premise of the consistent recoverability. It is therefore particularly attractive for the fragile mobile computing system.

ACKNOWLEDGMENT

The study is supported by Natural Science Foundation of Jiangsu (Project No. BK2012237).

REFERENCES

Alvisi, L. and K. Marzullo, 1998. Message logging: Pessimistic, optimistic, causal and optimal. Trans. Software Eng., 2: 149-159.

Brzezinski, J., A. Kobusinska and M. Szychowiak, 2006. Checkpointing and rollback-recovery protocol for mobile systems with MW session guarantee. Proceedings of the 20th International Parallel and Distributed Processing Symposium, April 25-29, 2006, Rhodes Island, Greece,-pp: 8.

Cao, G. and M. Singhal, 2001. Mutable checkpoints: A new checkpointing approach for mobile computing systems. Trans. Parallel Distrib. Syst., 12: 157-172.

Chen, I.R., B. Gu, S.E. George and S.T. Cheng, 2005. On failure recoverability of client-server applications in mobile wireless environments. Trans. Reliab., 54: 115-122.

Elnozahy, E.N., L. Alvisi, Y.M. Wang and D.B. Johnson, 2002. A survey of rollback-recovery protocols in message-passing systems. ACM Comput. Surv., 34: 375-408.

Gupta, S.K., R.K. Chauhan and P. Kumar, 2008. Backward error recovery protocols in distributed mobile systems: A survey. J. Theor. Applied Inform. Technol., 4: 337-347.

Li, G.H. and H.Y. Wang, 2005. A novel min-process checkpointing scheme for mobile computing systems. J. Syst. Archit., 51: 45-61.

Li, Q.H., T.Y. Jiang and H.J. Zhang, 2005. A transparent low-cost recovery protocol for mobile- to-mobile communication. J. Software, 16: 135-144.

Men, C., N. Wan and Y. Zhao, 2006. Using computing checkpoints implement efficient coordinated check pointing. Chin. J. Electron., 15: 193-196.

Ono, M., T. Hirakawa and H. Higaki, 2004. Hybrid checkpoint protocol for cell-dependent infrastructured networks. Proceedings of the 18th International on Parallel and Distributed Processing Symposium, Volume, 2, June 28- July 1, 2004, IEEE., pp: 1006-1011.

Park, T., N. Woo and H.Y. Yeom, 2002. An efficient optimistic message logging scheme for recoverable mobile computing systems. IEEE Trans. Mobile Comput., 1: 265-277.

Pradhan, D.K., P. Krishna and N.H. Vaiday, 1996. Recoverable mobile environment: Design and trade-off analysis. Proceedings of Annual Symposium on Fault Tolerant Computing, Jun 25-27, 1996, Sendai, pp: 16-25.

Taesoon, P., N. Woo and H.Y. Yeom, 2003. An efficient recovery scheme for mobile computing environments. Future Gener. Comput. Syst., 19: 37-53.

Wang, D.S. and M.L. Shao, 2003. A cooperative checkpointing algorithm with message complexity O(n). J. Software, 14: 43-48.

Yang, J.M., D.F. Zhang and W.W. Li, 2006. A robust and efficient rollback recovery implementation scheme. Acta Electron. Sin., 34: 237-240.

Zhang, Z., D. Zuo, Y. Ci and X. Yang, 2008. A rollback recovery algorithm based on causal message logging in mobile environment. J. Comput. Res. Dev., 45: 348-357.