An Enhanced Coordinated Checkpointing Scheme for Fault Recovery in Wireless Mobile Systems

Nomica Imran[1], Imran Rao[2], Young-Koo Lee[3,*], Byeong-Soo Jeong[3], Sungyoung Lee[1]

Dept. of Computer Engineering, Kyung Hee University, South Korea
[1]{nomica, sylee}@oslab.khu.ac.kr, [3](yklee, jeong)@khu.ac.kr

Dept. of Computer Science and Software Engineering, University of Melbourne, Australia
[2]imran@csse.unimelb.edu.au

Abstract. Mobile computing is an emerging and prospering field of distributed computing where wireless mobile devices let consumers do their job on the move. However, there are challenges such as unpredictable network quality, lower trust, limited resources (battery power, network bandwidth, storage, processing power, etc) and extended periods of disconnections which hinder to make this inspiration a reality. A major issue is the appropriate handling of such failures with minimal processing and storage overhead on mobile hosts. To meet these goals, we propose a proxy-based coordinated checkpointing scheme. In this scheme mobile hosts seamlessly store checkpoints on their respective proxies running on the middleware. Our approach makes it efficient to rollback to the latest consistent global snapshot, without direct involvement of the mobile hosts, which results in less processing and storage overhead on mobile devices as compared to existing schemes.

1 Introduction

Even though wireless mobile devices support mobile communication and flexible usage, at the same time they have some shortcomings that include: unpredictable network quality, lower trust, limited resources (power, network bandwidth etc) and extended periods of disconnections [3]. A major issue is the appropriate handling of such failures with minimal processing and storage overhead on mobile hosts. Traditional fault tolerance techniques are inadequate and unfeasible to meet the mobile computing challenges, as explained in next Section. Special measures are required to be taken to ensure fault tolerance of the mobile devices and once the fault occurred, to roll back to the last correct state.

Different techniques have been purposed in literature regarding coordinated [4], [8], [16] and uncoordinated checkpointing [13]. Some researchers have purposed non blocking coordinated checkpointing algorithms in which a checkpointing sequence number is being used to avoid inconsistencies. Nevertheless, these algorithms necessitate all processes in the computation to take checkpoints during the checkpointing, although many of them possibly will not be necessary. First algorithm to purpose a

* Corresponding author.

non-blocking coordinated checkpointing with minimal number of synchronization messages was [4]. G. Cao, et. al uncovered that this algorithm can result in an inconsistency in some situations and show that there does not exist a non-blocking algorithm which forces only a minimum number of processes to take their checkpoints [1]. G. CAo, et. al in [9] propose a non-blocking fault recovery technique to minimize the number of checkpoints by introducing the concept of mutable checkpoints.

In this paper, we present an enhanced and corrected proxy based coordinated checkpointed scheme based on our previous work [2]. Our contribution includes fixing the errors in our previous scheme; including more generalize cases in our algorithm such as mobility management, multiple instances of MSS in the system and a more detailed survey and comparison with related work. Our proxy-based coordinated checkpointing scheme takes storage and processing overhead from low-power mobile devices and delegates it to their respective proxies running on mobile service station (MSS).

The rest of this paper is organized as follow: Section 2 surveys the existing work in the field of fault tolerance, in general and in mobile systems in specific, and gives a comparison with our work. Section 3 elaborates our proposed proxy-based checkpointing scheme in details. In section 4, we present the simulation results and analyze our work. Conclusion and directions for future works follows.

2 Related Work

Acharya, et. al was the pioneers in presenting a checkpointing algorithm for mobile computing systems [15]. They use uncoordinated checkpointing technique in which a MH takes a local checkpoint whenever a message reception is preceded by a message sent at that MH. G. Cao, et. al [5] introduces the concept of mutable checkpoint, which is neither a tentative checkpoint nor a permanent checkpoint, to design efficient checkpointing algorithms for mobile computing systems. Mutable checkpoints need not be saved on the stable storage and can be saved anywhere, e.g., the main memory or local disk of MHs. This scheme, however, fails to overcome the storage overhead on mobile devices. Our scheme overcomes this drawback by delegating resource intensive tasks to the MHPs residing on the resource rich MSS.

First coordinated checkpointing algorithm was presented by G. Barigazzi, et.al [10]. According to their assumption, all communications between processes are atomic, that again is too restrictive. Algorithm purposed by R. Koo, et. al.[8] gives more flexibility to there assumption by allowing only those processes to take the checkpoint that have communicated with the checkpoint initiator either directly or indirectly, consequently minimizing the number of synchronization messages and the number of checkpoints. The entire checkpointing process is aborted if any of the involved processes is not able to or not willing to take a checkpoint.

All of the above coordinated checkpointing algorithms are blocking in nature as they require processes to be blocked during checkpointing. Checkpointing includes the time to trace the dependency tree and to save the state of processes on the stable storage, which may be long. Therefore, blocking algorithms may noticeably reduce the performance of the system and are considered inappropriate for mobile devices. One of the earliest works done in the category of non-blocking checkpointing was by

K. Chandy, et. al. [11], which deals with static nodes and system messages are sent along all the links in network during snapshot collection. This leads to a message complexity of $O(n^2)$. E. Elnozahy, et. al used the checkpoint sequence number to identify orphan messages [12]. This sequence number avoids the need for processes to be blocked during checkpointing. However, this approach is centralized in nature as it requires the initiator to communicate with all processes in the computation. L.M. Silva, et. al in [13] used a similar idea as [14] with an exception that the processes which did not communicate with others during the previous checkpoint interval change do not need to take new checkpoints. Both algorithms [13] and [14] suffer because of there centralized natures which assume that a distinguished initiator decides when to take a checkpoint.

3 Proposed Solution

We present a technique based on similar ideas as [5] in which purposed that the checkpoint can be stored on any storage media, either on the MH or on MSS. However, mobile devices, being light in processing and having limited storage, cannot efficiently store and bear the calculation overhead of constructing the dependability checkpointing tree as suggested by [5]. To resolve these issues, we introduce the concept of mobile host proxy (MHP), which resides on MSSS and seamlessly communicates to their respective MH. MSS coordinates among all the MHPs to calculate globally consistent snapshot. We propose that rather than calculating and storing the checkpoints at the mobile device, MH delegates this task to its respective MHP. As MHP is a static host and resides on the MSS, which is rich in resources, this delegation results in better performance and reliability as compared to existing techniques.

Our proxy based checkpointing algorithm uses two-phase commit (2PC) protocol in which we saves two types of checkpoints on MHP; permanent and tentative. A permanent checkpoint can not be undone, while a tentative checkpoint can be undone or changed to permanent. In first phase MHP takes its latest local snapshots and mark it as tentative. In tentative phase, MHP refrains from communication with other processes. Moreover, after receiving tentative checkpoints from all relevant processes, MSS will convert the tentative checkpoints to permanent and store in the stable storage to rollback for fault recovery. If any relevant process refuses to take tentative checkpoint, it will notify back to MSS which will notify all the participating processes to rollback the checkpointing activity and discard their tentative checkpoints. By relevant process we mean a process which received or sent a message from/to the checkpoint initiator after taking its last permanent checkpoint. So only affected processes are involved in the checkpointing process, which save the considerable overhead on the system. In addition to the tentative and permanent checkpoints, we employ the concept of mutable checkpoints [5] in our proxy based checkpointing scheme to avoid the checkpointing inconsistencies [6], [1]. Mutable checkpoints are neither tentative and nor permanent. When a MH takes a mutable checkpoints it doesn't send the checkpoint requests to other MHs and don't need to save the checkpoint on stable storage.

MHP being the gateway to MH, log and number all the messages sent and received by MH. Moreover, MHP after adjustable time quanta, request its corresponding MH to take a local snapshot of its processes (which includes process states, function stack). Moreover every MH may have its own snapshot frequency. This process of taking local snapshot is an offline activity and is not synchronized with the global checkpointing process. After taking its local snapshot, MH sends it back to MHP. Subsequently MHP stores this local checkpoint in its personal storage which is readily available to it. After time quanta expire, MHP will repeat these steps and will update the local snapshots. Fig. 1 explains the process.

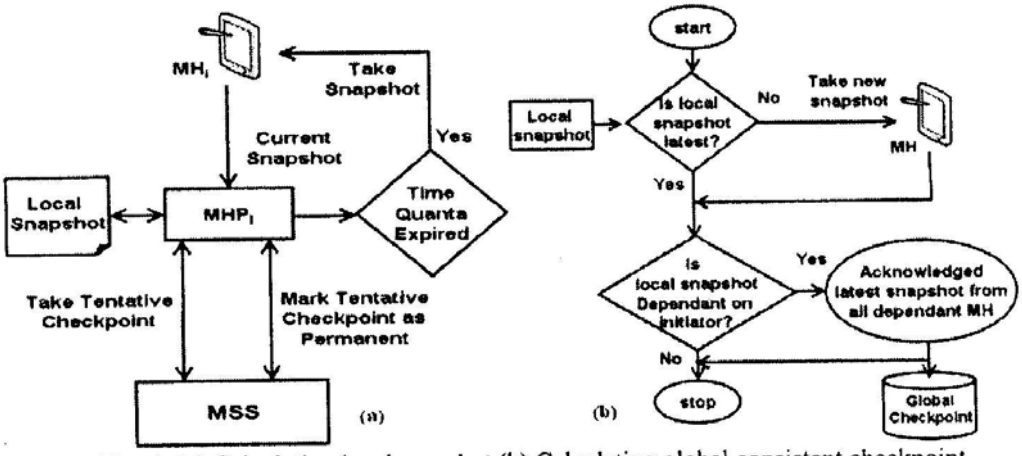

Fig. 1. (a) Calculating local snapshot (b) Calculating global consistent checkpoint

3.1 System Model

Any MH in the system can initiate the checkpointing process by sending a request to MSS through its MHP. If there are n mobile hosts, system can be modeled as;

$$\{(MH_p, MHP_p), MSS\} \forall p \in \{1,2,3,...,n\}$$

Let MH_p^a has its mobile host proxy MHP_p^a residing on the resourceful fixed MSS. MHP_p^a maintain two array R_p^a and csn_p^a of n bits where $R_p[j]=1$ means that MHP_p^a receive a message from MHP_j in the current checkpoint interval and $csn_p[j]$ represents the checkpoint sequence number of MHP_j known at MHP_p^a. Besides that MHP_p^a maintains two boolean variables cp_state and $sent_p$ to indicate if MHP_p^a is in checkpointing state and if it has sent a message during its current checkpointing interval respectively. MHP_p^a uses variables old_csn_p to save the csn of the current tentative checkpoint. It also uses a tuple $trigger(pid, inum)$ of checkpointing initiator identifier pid and csn number at pid. In our algorithm we use a non-negative variable ω which is used to detect the termination of checkpointing

algorithm. Initially array $csn_p[j]$, cp_state, ω are all initialized to 0s. $trigger(pid, inum)$ is initialized to $(p,0)$ at MHP_p^a. When MHP_p^a sends a message, it appends its $csn_p[p]$ to the message. Also MHP_p^a checks if cp_state is equal to one. If so, MHP_p^a also piggybacks its trigger with the message. If MHP_p receives a message from MHP_p^a, MHP_p^a takes a tentative checkpoint if and only if $old\_csn_p \le req\_csn$ (where req_csn is appended with the checkpointing request). Note that old_csn_p is literarily used to instead of $csn_p[p]$.

We also define three control messages MH_p^a may send for connection management with MSS. These messages are msg_join, msg_recon and msg_disc. Message msg_join is sent for a new connection with a MSS, msg_disc is for a graceful disconnection from its current MSS and msg_recon is for a reconnection after a graceful disconnection or failure with the same or a different MSS. Fig. 2 shows the process of creation of a new proxy for a newly connected MH. Note the difference between msg_join and msg_recon. msg_recon is meant to reconnect its processing from where it left whereas msg_join is used to make a new connection. Moreover, msg_recon and msg_disc messages sent by MH_p^a will piggyback the address pointer of the last MHP it corresponded with.

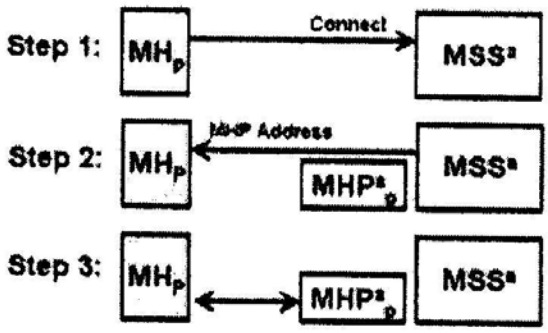

Fig. 2.A new MH joining the system

3.2 Checkpointing Scheme

Assume the checkpointing process is initiated by MH_q^b whose proxy is MHP_q^b, at time t_2 such that $t_2 < t_1 + \Delta_p$. The initiation process for checkpointing includes incrementing its $csn_q[q]$, setting its ω to 1, setting its cp_state_q to 1 and storing its own identifier, the new $csn_q[q]$ in its trigger along with sending this information

along with its latest snapshot to its MHP, that is MHP_q^b in this case. Once this information is received from MH_q^b, MHP_q^b marks the received snapshot as tentative and is saved on its personal local storage. Subsequently the MHP_q will report MSS for its readiness to take a checkpoint via sending checkpointing request. The request carries the trigger of the initiator, R_q and ω. MSS multicasts this request to all MHP_i connected such that $R_q[i] = 1$ and where $i \neq q$.

Upon receipt of the checkpointing request from MHP_q^b, MHP_p^a will decide should it need to take a checkpoint. Checkpoint is inevitable if and only if $old\_csn_p \leq req\_csn$. MHP_p^a on behalf of MH_p, update its csn and cp_state and compares checkpointing request message trigger with its own trigger. If message trigger is equal to its own trigger (implying that MHP_p^a has already taken a checkpoint for this checkpointing), MHP_p^a further checks if it has a mutable checkpoint which has a trigger identical to message trigger. If not, MHP_p^a sends the appended ω to the initiator. In all other cases MHP_p^a turns the status of mutable checkpoint as tentative and then propagates the request.

If MHP_p^a propagates the request to all MHs on which it depends, it may result in a large number of redundant system messages since some MH on which MHP_p^a depends may have received the request from other MH. The [8] algorithm uses this approach and its system message overhead can be as large as $O(N^2)$, where N is the number of MH in the system. On the other hand, only propagating the request to MHs on which MHP_p^a depends, but MHP_q (the sender) does not, may not work since receiving a request does not necessarily mean that the MH inherits the request. We solve this problem by attaching some information (csn and R which are saved in a structure called MR in the algorithm) to the request as does [9]. MHP_p^a only propagates the request to each MHP_k on which MHP_p^a depends, but MHP_k may not have inherited the request; that is, if MHP_p^a knows (by MR) some other process has sent the request to MHP_k with $req\_csn \geq scn_p[k]$ (req_csn is appended with the request and saved in $MR[k].csn$), it does not need to send the request to MHP_k; otherwise, it has to send the request since MHP_k may inherit the request from MHP_p^a. Furthermore, MHP_p^a appends the initiator's trigger and a portion of the received weight to all those requests. Then, MHP_p sends a *reply* to the

initiator with the weight equal to the remaining weight and resumes its underlying computation.

If $msg\_trigger \neq own\_trigger$, MHP_p^a takes a tentative checkpoint, increases its csn_p, as well as propagates the request as above. Eventually, MHP_p^a clears R_p and $sent_p$, sends a reply to the initiator by means of the remaining weight, and then recommence its basic computation.

3.3 Mobility Management

Duration MH movement, the cost of carrying the checkpoint and logs can be a significant communication overhead. In our proposed scheme, due to mobility of MH_p, there can be n number of MHPs, each on a different MSS it has visited. These distributed MHPs not only take space but are also difficult to manage. Moreover, after recovery from failure, they make recovery process slow due to delay in collaboration among these distributed proxies.

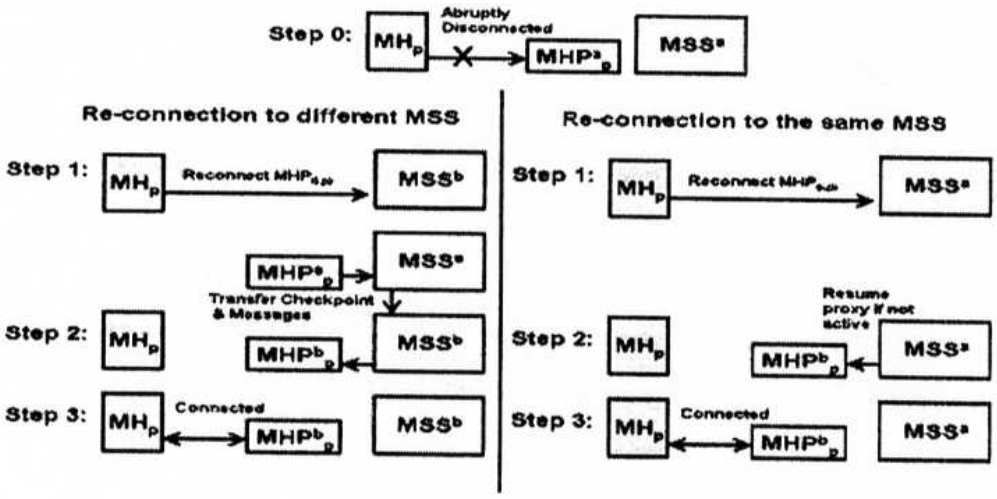

Fig. 3.A MH Reconnection and Hand-off management

We propose our mobility management scheme keeping above mentioned issues in mind. Let us suppose MH_p looses its connection from its current MSS_a due to mobility. Let it enters in the cell of MSS_b and send msg_recon to MSS_b with piggybacking $curr\_proxy_p^a$, the address of the last mobile host proxy it corresponded with. MSS_b will identify MH_p as a new MH in its cell and will create a new proxy MHP_p^b for it. MSS_b will also receive the last saved checkpoint ckp_p^j from $curr\_proxy_p^a$ at MSS_a. ckp_p^j will be copied into local stable storage of

MHP_p^b. There are also cases when a connection is lost due t poor network or failure. Suppose MH_p abruptly disconnects from its current MSS_a. At a later time, MH_p recovers from its failure and wants to reconnect to the same MSS_a. It will send a reconnection message msg_recon to MHP_p^a piggybacking $curr\_proxy_p^a$, the address of the last proxy it corresponded with. Finding $curr\_proxy_p^a$ as its own address, MHP_p^a will identify MH_p as its lost MH. So MHP_p^a will send the last checkpoint ckp_p^j to MH_p. Note that during the time period of abruptly disconnection of MH_p from its proxy, MHP_p^a will be in a suspended mode. For a considerably longer disconnection period, MHP_p^a can consider MH_p permanently disconnected or dead and can trigger MSS_a for a disconnection on behalf of its MH by sending msg_disc message. Fig. 3 explains these cases.

4 Simulation and Analysis of Results

We simulated our proposed scheme to evaluate the performance effects due to the inclusion of mobile host-proxies in the systems. We choose communication cost and time to recover as simulation metrics for 20 MHs with varying network bandwidths, number of messages exchanged and hand-offs.

Let MH_p takes its local checkpoint describing current state of MH_p at time T_p^{start} and sends to MHP_p. Let T_p^{ckp} is the average time to calculate local checkpoint at MH_p. If λ_p is the wireless network bandwidth between MH_p and MHP_p and θ_p is the data load associated with T_p^{ckp} then T_p^{transf}, time taken to transfer checkpoint from MH_p to MHP_p, can be calculated as follows; $T_p^{transf} = \theta_p / \lambda_p$ Note that Δ_p must be set greater than ($T_p^{transf} + T_p^{ckp}$). Let later at some time t_{cp}, MHP_p receive a checkpointing request from MH_q. The total time T_p taken by MH_p to take a checkpoint after receiving the checkpoint request from a peer MH is;

$$T_p = T_p^{transf} + T_p^{ckp} + \phi$$

We define ϕ as the time taken to participate in the checkpoint process. Now note that MHP_p will only request a new local snapshot from MH_p if there was some message exchange to and from MH_p otherwise MHP_p will use locally stored snapshot of MH_p which will save considerable time (Fig. 4). Every time the local snapshot is

used from the personal storage of the MHP_p, the total time taken to calculate the snapshot is considerably less. When the, locally stored snapshot is invalid, MHP_p requests a new snapshot from MH_p. In that case the time taken to take a checkpoint is equal to the system without proxies as seen in the case 6, 11 and 18. We define communication cost as the total number of bytes transferred per checkpoint. In our scheme, due to the existence of MHP there is less communication cost in comparison with system without proxies (Fig. 5).

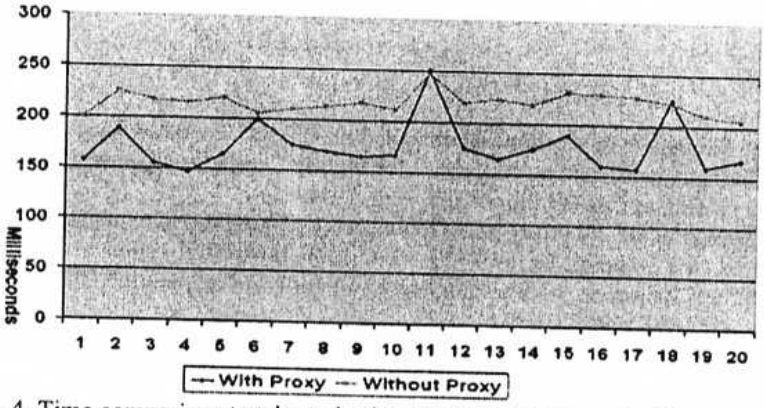

Fig. 4. Time comparison to take a checkpoint using proxies and without proxies

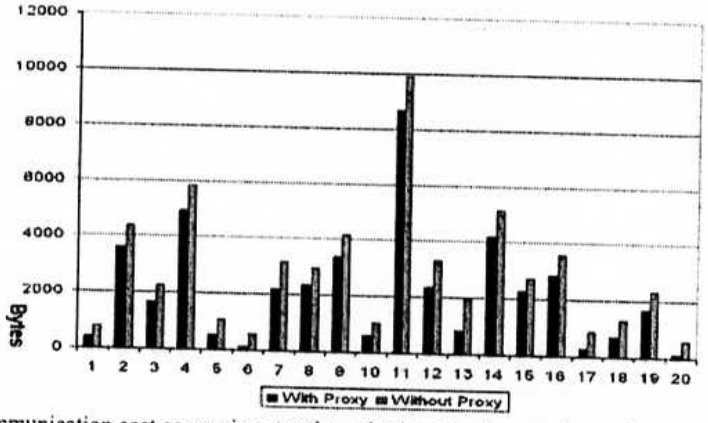

Fig. 5. Communication cost comparison to take a checkpoint using proxies and without proxies

5 Conclusion and Future Work

In this paper, we presented a proxy based checkpointing scheme for wireless mobile systems. We extend the work done by [2] and propose a proxy based extension of this checkpointing approach. Our proxy-based coordinated checkpointing scheme takes storage and processing overhead from low-power mobile devices and delegates it to their respective proxies running on mobile service station (MSS).

In future we plan to implement our proposed scheme and accumulate the experiment result to find correctness of our scheme. We also intend to investigate the performance and storage overhead of our scheme.

Acknowledgements. This research was supported by the MIC(Ministry of Information and Communication), Korea, under the ITRC(Information Technology Research Center) support program supervised by the IITA(Institute of Information Technology Advancement). (IITA-2006-(C1090-0602-0002))

References

1. G. Cao, M. Singhal. "On the Impossibility of Min-Process Non-Blocking Checkpointing and An Efficient Checkpointing Algorithm for Mobile Computing Systems", pp. 37-44, In Proc. of Int'l Conf. on Parallel Processing, Aug. 1998
2. I. Rao, N. Imran, et. al, "A Proxy based Efficient Checkpointing Scheme for Fault Recovery in Mobile Grid System", pp. 448-459, HiPC 2006, Bangalore, India
3. Forman, G. and Zahorjan, J. "The Challenges of Mobile Computing", IEEE Computer, vol. 27, no. 4, (April 1994)
4. R. Prakash and M. Singhal, "Low-Cost Checkpointing and Failure Recovery in Mobile Computing Systems", pp. 1035-1048, IEEE Trans. on Parallel and Distributed System, Oct. 1996
5. G. Cao, M. Singhal, "Checkpointing with mutable checkpoints", Theoretical Computer Science, Volume 290, Issue 2, Jan 2003
6. A. Sajjad, et. al., "MAGI - Mobile Access to Grid Infrastructure: Bringing the gifts of Grid to Mobile Computing", pp 311-322, NODe/GSEM 2005
7. Y. Tamir, C.H.Sequin, "Error Recovery in Multicomputers using global checkpoints", in Proc. 13th Intl. conf. Parallel Processing, Aug, 1984.
8. R. Koo , S. Toueg, "Checkpointing and rollback-recovery for Distributed Systems", IEEE Transactions on Software Engineering, v.13 n.1, p.23-31, Jan. 1987
9. G. Cao, M. Singhal, "Mutable Checkpoints: A New Checkpointing Approach for Mobile Computing Systems," IEEE Transactions on Parallel and Distributed Systems, vol. 12, no. 2, pp. 157-172, Feb., 2001.
10. G. Barigazzi and L. Strigini. "Application-transparent setting of recovery points.", pp. 48 - 55, In Proc. of the 13th Intl. Symposium on Fault-Tolerant Computing Systems, 1983
11. K. M. Chandy and L. Lamport, "Distributed Snapshots: Determining Global States of Distributed Systems", pp 63-75, ACM Transactions on Computer Systems,1985
12. E.N. Elnozahy, et. al, "The Performance of Consistent Checkpointing", pp 39-47, In Proc of the 11th Symposium on Reliable Distributed Systems, October 1992
13. L.M. Silva and J.G. Silva, "Global Checkpointing for Distributed Programs", pp. 155-162, In Proc. of IEEE Symposium on Reliable Distributed Systems, Oct. 1992
14. E.W. Dijkstra, " Self-stabilizing Systems in Spite of Distributed Control", pp 643-644, Communications of the ACM vol.17, 1974
15. Acharya, B. R. Barinath, "Checkpointing Distributed Applications on Mobile Computing", pp. 73-80, In Proc. of the 3rd International Conference on Parallel and Distributed Information Systems, Sep. 1994
16. N. Vaidya, "Staggered Consistent Checkpointing", pp. 694-702, IEEE Trans. Parallel Distributed Systems vol. 10, no. 7, 1999