Journal of Computer Science 3 (8): 617-623, 2007
ISSN 1549-3636
© 2007 Science Publications

Recovery in Distributed Systems from Transient and Permanent Faults

M. Aliouat and Z. Aliouat
Department Informatique, U.F.A.S, Maabouda, Route de Bejaia 19000 Algeria

Abstract: The recovery mechanism from transient fault in distributed systems has been intensively studied in the past, but to our best knowledge, none of these studies has been devoted to cope together with transient and permanent hard faults. Our study devoted to recovery processes in a distributed environment in case of hard faults like transient or permanent. The recovery mechanism we presented can be based on one of the six proposed strategies involving checkpointing and message logging between distributed application processes. This exhaustive number is system-dependant. The strategies have been examined with respect to propagation recovery through processes in order to prevent the fastidious well known domino effect problem. The considered framework was a distributed system composed of a set of autonomous nodes running each one a local system; and some of them were predisposed to replace failing ones in case of permanent fault. Our main contribution was to enable a distributed application to meet its requirements of terminating its mission in spite of node crash.
Preliminary experimental results of a fault tolerant mechanism based upon one of the proposed strategies demonstrated that our proposals seem to be conclusive.

Keywords: recovery, distributed systems, transient permanent fault tolerance

INTRODUCTION

Meeting requirements of dependability in critical applications has led to the development of techniques to improve design of fault tolerant systems which can maintain specified services in spite of fault occurrences. The developed techniques can be dedicated to cope with hardware faults, software faults or both. They are essentially based on two fundamental and complementary approaches: backward error recovery and forward error recovery [1, 4, 6].
Based on the identification and the accurate knowledge of the error, Forward Error Recovery (FER) copes with the failure in correcting erroneous system state by acting on the damaged part. This action needs first an accurate assessment of damages inflicted to the system. Backward Error Recovery (BER) is independent towards assessment and prediction of damages caused by a fault. It is more general since it does not depend on applications. So, recovery from an error is then achieved by restoring an old state (presumed error free) prior to the fault occurrence, from which restarting of the failed system will take place.

Designing a recovery mechanism, particularly in a distributed concurrent processes system, implies to take into account the costly and complex problem of error propagation resulting from message exchanges between processes. So, if interprocess communications are not well coordinated according to the establishment of checkpoints, BER approach can be exposed to an uncontrolled recovery propagation which can degenerate into domino effect [8].
Restarting of distributed applications must take place from a Consistent Global Checkpoint (CGC) [5] or Recovery Line (RL) and the way according to which this RL is determined, constitutes a characteristic of a recovery mechanism. Two policies are commonly used: Static determination: where the recovery line is established during writting of programs [8] and dynamic determination: where the RL is determined automatically during a recovery operation [3, 6]. The latter one may include three common approaches used for creating global checkpoints like:

a) Coordinated checkpointing where local checkpoints of communicating processes are coordinated with the initiator in order to form together a recovery line [6].
b) Uncoordinated or asynchronous checkpointing where a total freedom is given to processes when recording their local checkpoints without regard to obtaining or not a recovery line [3].

Corresponding Author: Makhlouf Aliouat University of Ferhat Abbes, Faculty of Engineer Science, Computer Science Department, Setif 19000 Algeria

c) Communication-induced checkpointing where each process is forced to take checkpoints based on information piggybacked on the application messages received from other processes [9].

Another characteristic is the relationship existing between application programmer and a recovery mechanism. The latter one can be transparent [10, 11] if it is based on BER approach, or fully integrated into application if it is issued from FER approach [6]. In this case, the programmer must take into account the fault tolerance problem when writting his application. A recovery mechanism is also distinguishable with regard of the fault type which can handle: i.e, can it only handle software fault? Hardware fault? Or both? In case of hardware fault, can it handle transient fault, permanent fault or both? Most of protocols referred in the literature are devoted to transient faults, exception handling [4] and more generally FER approach is dedicated to tolerate essentially software faults, while BER fits as well on software faults as on hardware faults. However, if BER approach is fitting on hardware faults, more particularly in permanent ones, the recovery of these faults always require some redundancy in hardware components. Based on their inherent and intrinsic redundancy and on their large availability, distributed systems provide an appropriate environment to improve capabilities of fault tolerance, and thus are suitable framework to enhance their resilience to failures.

In this paper, we focus on the concurrent processes recovery in distributed environment from hardware faults (transient or permanent). Many error recovery strategies based on BER approach are presented. The particularity of this work is:

1. Handling together transient or permanent faults without requiring special architecture, so the provided recovery strategies are system level based and may be usable in numerous existing systems with a transparent manner [10].
2. The cost incurred (time overhead) resulting from recovery operation are partially avoided by message logging technique [7], or completely when (in addition) some easily feasible criteria [2], which prevent forward propagation (D7) between processes, hold.
3. No constraint on processes communications or checkpoints creation is imposed. This total freedom can be profitably used by processes to save their local checkpoints at convenient instants. Therefore, this last situation is particularly important when a process decide to record a checkpoint in order to save an important event like receiving messages from a sender which cannot recover, for instance a sensor. Furthermore, when the recovery blocks scheme [8] is integrated in our strategies, the complexity will be certainly increased, but the expected recovery mechanism will be of general purpose (handling of hardware faults as well as software ones). This combination is actually under study. This paper is organized as follows. After a given a background in section 2, we present, in section 3, many basic recovery strategies. Each strategy is examined with regard to system perturbation (overhead) in case of error recovery from transient or permanent fault. In order to get minimal system perturbation values, we combine, in section 4, the positive aspects of basic strategies in order to form two mixed ones. In section 5, we describe a simple implementation to point out one of mixed strategies feasibility. A conclusion is given in section 6.

BACKGROUND

System Model: We consider a distributed system consisting of a set of stations or nodes running each one its own system. A distributed computation is performed by a set of N processes Pi, $i \in [1, N]$, running concurrently on nodes. A piece wise deterministic model of computation is assumed, that is, a process always generates the same sequence of outputs from its execution for the same sequence of inputs. Fail-stop failure mode of processes is assumed except where otherwise stated. Hardware faults, transient or permanent, are considered in this paper. In transient error (an error is an erroneous process state originated from fault), no physical damage is created, and so a process can be re-executed and is likely to fail again after it recovers from a failure.

Definitions: Before detail in the sequel, some preliminary definitions are needed.

D1: A checkpoint (CP) is a process state periodically saved for restoring it later in case of error. The period of activity between two consecutive CPs is called a Recovery Region (RR).

Remark: For the sake of implementation each RR can be associated to reception or emission of message(s). So, we need to characterize it as an RR which receives or sends message(s) with an Identifier Receiving Recovery Region (IRRR) or Identifier Sending Recovery Region (ISRR).

D2: A CP is said to be active if it is the most recently recorded one. We call Restarting Checkpoint (RCP), a selected CP from which an execution of a process involved in a recovery operation is restarted. Let CPi and CPj belong to a process Pi, CPi is said to be dominant of CPj if CPi is created before CPj (noted: CPi < CPj).

D3: For every pair (CPp, CPq) of checkpoints, such that CPp belongs to process P and CPq to Q, CPp is said to be a Direct Propagator (DP) of CPq (noted CPp $\rightarrow$ CPq) if and only if: one message at least flows from the recovery region identified by CPp in P to the recovery region identified by CPq in process Q. Conversely CPq is said to be a Direct Dependent (DD) of CPp. In the same way, CPp is said to be an Indirect Propagator (IP) of CPq (noted CPp CPq) if $\twoheadrightarrow$and only if: - either CPp is a DP of CPq, - or (recursively), there exists CPs belonging to a process S such that: CPp is a DP of CPs and, CPs or any other checkpoint of S, successor to CPs, is an IP of CPq. Conversely, CPq is called an Indirect Dependent (ID) of CPp.

D4: A Dependents List (DL) of a process P is the set of dependents of all CP's belonging to P at time t. A sub_list of DL associated to a given CP of P is called an Immediate Dependents List (IDL) of that CP. In the same way, the set of DP of all CP's belonging to P is called Propagators List (PL) of P. The DP's of a given CP which belongs to P represent an Immediate Propagators List (IPL) of that CP.

D5: A consistent global state of a system (or Recovery Line: RL) is a state defined by a set of CPs, one per communicating process. They form together a "barrier" which stops any recovery propagation.

D6: A Definite Invocation (DI) of a process is an invocation for a recovery where the called process must roll back to RCP supplied in received invocation message. As opposed to DI, a Random Invocation (RI) is one where the RCP of the invocated process P is determined in accordance with the RCP of the calling

Process indicated in received invocation message.

D7: An information message is said to be revoked if it is erroneous, or if it is correct but its use causes problems. A message is said to be indispensable if it is required for a reexecution of the recovered process. A backward propagation is a propagation of recovery generated by a process Pj to reach Pi in order to re-create for Pj at least one indispensable message. A forward propagation issued from Pj to reach Pk is recovery propagation subsequent to reception by Pk of one revoked message at least, sent by Pj.

D8: A recovery propagation is said to be of level n ($n > 0$) if it reaches n different processes from the initiator one. A recovery propagation is said to be of order p ($p > 0$) relatively to process Pi, if the length of roll back executed by Pi is p recovery regions.

D9: A recovery graph (RG) of process Pi ($1 \leq i \leq N$) is a graph which describes exchanges between Pi and its partners Pj ($i \neq j$, $1 \leq j \leq N$). An RG contains information needed by a recovery operation like DL and PL lists.

BASIC STRATEGIES

Several strategies for recovery from transient or permanent faults have been studied; but due to the restricted paper space, only few of them are examined below and the others are merely introduced (For more details refer to [2]).

Before developing some strategies, we first look into the principle of a recovery operation, the needed hypotheses, and how to avoid rollback propagation.

Recovery Principles

A recovery operation is the action taken by an appropriate mechanism to enable a failing system to recover a correct state from which reexecution is resumed. The faults we consider are hardware (transient or permanent), and the distinction between them is complex (indeed impossible). So, when detected, an error is firstly considered as transient (except the case of processor crash where this one is detected by its peers via an appropriate diagnostic) and handled as such. It is considered as permanent one after several vain trials. Usually, processing of transient faults is accomplished in two stages: detection, and recovery; but permanent faults may need four: detection, diagnosis and hard reconfiguration (switching to spare component), soft reconfiguration (selecting an efficient process context), and restarting. To avoid any ambiguity in recovery operation, the following hypotheses are applied.

H1: the error latency is scarcely presumed nil except where otherwise stated. In other words, processors follow fail-silent failure mode.

H2: no error happens during a recovery operation.

H3: Communication system is reliable.

With H1 and H2, all messages sent and received before errors are considered to be correct, then no propagation of roll-back to partners of a failing process is needed. H1, particularly, depends on the error detection method, like concurrent detection by means of self-checking logic. The criteria defined below allow forward propagation to be avoided.

Criterion 1: When received, a message is recognized to be useful for the execution of the receiving process or harmful because it results from a roll-back of the issuing process and must be ignored.

Criterion 2: The recovery processor is able to recognize the message already sent during a normal execution of a process P and avoids the reemission of the same message when P is rolling back after error.

Criterion 3: After a process roll back, all messages are resent but without causing any problem in receiving processes, i.e. the actions generated by those messages are idempotent. It is intended that the non respect of criteria 1, 2 is merely due to the unused of the associated algorithms. In the sequel, we consider criteria 1, 2 to be held. Then the consistent system state (i.e. RL) may be reached in only one roll back per process. A distributed algorithm [2], initiated by a recovery processor, enables to know all CP's of processes represented in the RL; after this, a recovery message is sent to every concerned process.

Strategy A: In this strategy, *every node saves messages that it receives.* In order to see what happens and how each type of hard fault is handled, strategy A is successively examined according to transient fault, permanent fault and finally both integrated in the same handling algorithm.

Transient faults:Their occurrences may lead to a state alteration of the affected process Pi and possibly, by "contamination", to its partners Pjs. This contamination takes place when Pj's partners have consumed at least a revoked message (D7) produced by Pi. To correct this erroneous system state, Pi and Pj's must suspend their current execution and resume from a recovery line. Since the new execution is submitted to the same events as during the first one, particularly, the need to re-use the same information, this is accomplished by means of locally saved messages in the failing node.
With hypothesis H1 and criteria 1 or 2, recovery operation is achieved without any propagation to Pj's partners. Then, the recovery of any failing process takes place in the original failing node, since the latter is not affected by any irreversible physical damage.

Permanent fault: Contrary to what happens in case of transient fault, the damaged node is now unable to pursue the execution of its process Pi, and then recovery is achieved in another operational node called Recovering Node (RN) associated to the first one. The RN may be a spare node in case of non graceful system degradation, in which original architecture topology is preserved, or in one of the active nodes with degradable performance (may be in modified topology). All information saved in the failing node is presumed to be lost and the only recovery information available on the RN is: the code of failing process Pi and one CP at least. To resume execution of Pi, the recovery processor must impose the backward propagation to all Pj partners to recreate indispensable messages for Pi. Forward propagation inherent to messages sent previously by Pi, is avoided by criteria 1 and 2. We now examine what information is needed to the mechanism which may adopts this recovery strategy.

Required information: To respect criterion 1, it is necessary for every process to keep information relative to received messages, (message serial number, and sender identifier). For the criterion 2, saving of serial number of each sent message and identifier of its receiver process is needed.
In case of unsatisfied criteria 1 or 2, it is important to propagate the recovery from the failing process Pi to every partner Pj which has received from Pi one revoked message at least. This action is achieved by means of information which identifies message exchanges established between Pi and Pj. That information is found in the DL list (D4) of the recovery graph (D9). Thereby, a protocol between processes is needed, so when a process receives a message, it sends to its sender the Identifier of Receiving Recovery Region (IRRR). Thus, with DL list, a recovery processor is able to designate to each partner Pj of the failing process Pi, an RCP (D2) from which restart can be done (definite invocation, D6). The information required handling permanent faults, and available on each Recovering Node, is: the process code and one or several contexts associated to CP of that process. Application messages may piggyback some additional information to be used by the recovery mechanism; for instance: message number, the Identifier of the Emission Recovery Region (ISRR), required constructing the PL list (D4) of recovery graph.

Recovery operation

Case of Transient Fault: After error detection, the Recovering Processor RPri restarting the failing process Pi, performs the following recovery algorithm.

1. RPri uses as Restart Checkpoint (RCP) the active CP of Pi. Since no forward propagation is possible, there is no return of propagation to Pi reaching a dominant CP (D2) of RCP;

2. RPri restores the state of Pi corresponding to RCP;
3. RPri resumes execution of Pi.

In this case, only a single CP is needed to save for each process of the system, and moreover, criterion 2 may be preferably satisfied.

Case of Permanent Fault: The failing node is recognized as such, diagnosis and hard reconfiguration are supposed realized. Because lack of Pi Recovery Graph, the Recovering Processor RPri initiates a random invocation (D6) to cause backward propagation of every Process Pj, partner of Pi, such that : $\exists$ CPj$\in$Pj and CPj $\rightarrow$ RCP, i.e. CPj is a direct propagator (D3) of RCP of the failing process Pi. The following actions are executed.

1. Sending by RPri, to every process/node, of a random invocation message which contains the RCP of the failing process Pi. This RCP is the active CP of Pi.
2. After the invocation message is sent, RPri restores the state of Pi associated to RCP, and restarts Pi execution.
3. As soon as an invocation message is received, the receiver processor RPrj determines if there is one of its processes Pj such that : CPj $\in$ Pj and CPj $\rightarrow$ RCP, if several CPj may exist, then only the dominant one is selected.
4. Deletion of all possible successors of CPj.
5. Restoration of Pj state corresponding to CPj.
6. Restarting of Pi (Because criterion 2 is set, only application messages needed by Pi are re-sent).

Case of transient or permanent Faults: The major aim of this study is to recover from solid faults, whatever their type. Then when detected, a fault is firstly handled as a transient one; it is considered as a permanent after many vain attempts to recover from it. The precedent actions in a) and b) are then applied.

Overhead and roll-back length: The concern of a recovery mechanism is tightly related to its cost (overhead incurred), then the difficulties resulting from its design is more relevant to efficiency than to the error recovery aspect. Therefore, the strategy is considered according to a determining factor inherent to recovery operation cost like domino effect.
Avoidance of any propagation in transient fault, gives the strategy optimal perturbation values, thus the factors of propagation (D8) are: level n=0 and order p= 1.
Generally, with any type of solid fault, strategy A is exempt from domino effect. Indeed, the latter one can only be caused by backward propagation, but while only a single permanent fault is considered at time, any process Pj reached by a backward propagation cannot propagate the latter (since indispensable messages are locally available for failing process Pi). Then there is no possibility of creating a cycle of backward propagation, and the absence of a cycle implies absence of a domino effect. Therefore the level of propagation is: $n \leq m$ where m is the number of direct propagators of Pi.
The strategy A is attractive for recovery of transient faults. However permanent faults may create situation of unnegligible system overhead during recovery operation. Since the occurrence frequency of these permanent faults is lower than the one of transient fault, this does not represent a handicap. Nevertheless, an improvement to handle this class of faults is devoted to the strategy B.

Strategy B: Every message received by a node is saved in its producer one. Due to restricted space, we only give the obtained results (Cf. [2]).

Roll-backs length and overhead: The forward propagation is avoided, but backward propagation may be imposed to get indispensable messages for the failing process P. Therefore two cases may occur: - Before a permanent fault occurrence, no propagation is possible then the overhead factors are optimal: level n = 0, order p = 1 because only the failing process has to roll back at its active CP. - After permanent fault, it may be possible to impose the backward propagation to message producers when the former ones are located in a recovery node. Then, factors are: level: $n \leq m$ (m is the number of these producers), order P = 1 only for P.

Strategy C: No saving messages are required
This strategy is only based on checkpointing but no messages logging is used. It can be studied when all strategies concern only checkpointing or can be used as a reference position. Knowing that strategy C is optimal in terms of incurred overhead during normal execution (no messages logging), we can then use it when comparing among others strategies.

Strategy D: This strategy is characterized as follows:
Every message received by a node is picked up and saved by the corresponding recovering node.
Because process code, checkpoints and needed messages are available in the recovering node, this strategy is particularly attractive in case of permanent errors. So, in long living application when hard components redundancy is very extensive, like in Blue Gene/L supercomputer with 16384 processors [12], we can make no difference between transient and permanent error. For instance, every error (even

transient) affecting a node may involve to immediately and systematically switching to associated recovering node. This approach is very realistic because the Mean Time between Failure factor is very small like about few hours. Therefore, to save time from repetitive tries to recover from a recurrent transient error, it is convenient to make no distinction and consider all errors as permanent.

Description: As we have seen before, saving messages plays a privileged part in the limitation of perturbations, and permanent faults may involve many processes, causally related to a failing one, in backward propagation actions. The aim of this strategy is to create an adequate environment to handle permanent errors. Therefore, every received message in a node is captured and saved in the corresponding recovery node. We, thereby, consider the recovery operation according to the type of occurred error:

- When the detected error is considered as a transient one, the recovery processor requests the recovery node to get saved messages which are indispensable to resume the execution of the failing process.
- When the error is handled as a permanent one, the failing process P will be resumed in recovery node, where essential P context is provided. No backward propagation is needed unless case where a recovery node is crashed before P (case of multiple permanent faults). Due to the strategy specification, it convenient to have a communication medium allowing a capture of messages easily (like broadcast medium). So, the strategy best fits an Ethernet or Wireless area network. The information needed for a recovery mechanism is of two sorts:
- One relative to the recovery node with recovery graph adapted to this case, particularly, each element of the propagator list must indicate the number of the first message received in the associated Receiving Recovery Region. This is needed for requesting indispensable saved messages in case of roll-back.
- Another one, related to the recovery node, concerning the saved messages and associated RP.

Recovery in Transient or permanent fault

1. Transient fault:

a Sending, by a Recovery Processor RPri, a request to the recovery node to supply the indispensable messages. The request must piggyback the RCP of the failing process Pi, or the first message number previously received in the RCP recovery region.

b Restoration of Pi state corresponding to RCP and resumption of Pi execution.

2. Permanent fault:

a Reconfiguration: i.e. disconnection of failing node and commutation to corresponding RN.

b Restoration of state associated to RCP (most recently saved in RN) and resumption of Pi.

Perturbations and Roll-back length: The perturbations are eliminated; so the propagation factors are: level $n = 0$ and order $p = 1$. When a request for indispensable messages cannot be satisfied because of absence of the concerned Recovery Node, the level n is such that $n \leq m$ where m is the number of direct propagator processes of the failing one.

All previous strategies are of concern to find a tradeoff between several factors like: lowering or preventing domino effect, minimizing overhead during normal execution of the system ... The next mixed strategies particularly advocate creating an appropriate environment to recover from permanent errors.

MIXED STRATEGIES

Since these strategies are combined from basic ones, a summary description of each one is given below.

Strategy AB : Sender-Receiver based strategy.

Every message received and saved in a receiving node is also saved in sending node.

This strategy is a combination of strategy A and strategy B. Two important results are reached namely: 1) The perturbations are optimum so, level $n = 0$ and order $p = 1$. 2) A single context corresponding to the active CP is saved for any process.

Strategy AD: Sender-Recovering node based strategy.

Every message received and saved in a Node is captured and saved by the corresponding Recovering Node.

With duplication of message saving, in both potential failing node and Recovering Node, we gather advantages of basic strategies A and D in order to easily handle transient faults and permanent ones. So, it results from this association two fundamental characteristics, namely: - No transfer of indispensable messages takes place during a recovery operation, then neither penalization of processes other the failing one, and no tendency to increase the activity of transmission system is incurred. - Complete isolation of safe processes opposite to the failing one. Consequently a single context is saved for each process.

While overhead incurred from message management is proportional to processes interaction, the former may be unnegligible. However, a separate processor may be dedicated to perform this management.

Recovery Operation: While strategy AD is combined from strategies A and D, the recovery information required are: a recovery graph as described in strategy A, and a data structure inherent to a Recovering Node like one defined in strategy D. The recovery algorithm is formed by the one defined in strategy A for transient fault, combined to which given in strategy D for permanent fault.

Perturbation and Roll-backs Length: Strategy AD is the most optimal one so, we state: level n = 0, order p = 1.

IMPLEMENTATION

Our first simple application was experimented in three workstations on each one is running a process. One of the three processes say P has to determine prime numbers in [1, N] interval and sends then to the others (Q, R) for usage (the prime numbers are also displaying in P screen). Q and R are doing other things where the generated numbers are consumed. Each workstation has an associated recovery one. A transient fault was simulated in executing an interruption handler, while a permanent one is simulated in switching off the P/workstation. In the first case, the process P redisplays the prime numbers previously determined in the active recovery region showing then the AD strategy principles. In the second case, the failed process P is recovered on its recovery workstation (Q); the displaying of prime numbers is resumed at point it was previously interrupted. The recovery workstation screen is divided in two windows where one part shows scrolling of resumed prime numbers and the second part shows the process Q result. The progression of Q and R are not affected. The second switching off of Q/node had led to R/node to support P, Q, R continuation but the given response time was slightly heavy. The obtained results are then to confirm our ideas.

CONCLUSION AND FUTUR WORK

Several recovery strategies in distributed system have been presented. They handle both transient and permanent faults. This can allow the crucial application programs to have a non-stop execution despite solid failures. Generally, the avoidance of situation of important system penalization (undoing a great deal computation in case of error) is obtained from propagation control constraints. These constraints are relative either to the communications or checkpoints setting. This control is achieved in our strategies, partially by saving messages, or completely by additional criteria. Then the processes liberty is preserved. The strategies (notably AD, AB) are very attractive because they are almost domino effect free, and also, only a single checkpoint is saved for each process. The aim of the strategies exhaustivity is to cope with a great class of fault tolerant systems.

It is noteworthy that to improve more and more reliability, it is necessary to take into account both hardware and software faults. So, we are actually incorporating the recovery blocks technique in the proposed strategies in order to reach a more general recovery mechanism.

REFERENCES

1. Aliouat, M. 1986. Reprise de processus en environnement distribué après pannes matérielles. *PHD Thesis*, INPG Grenoble France.
2. Aliouat, M. 2006. Recovery from Hard Faults in Distributed Environment. *RR. N°19B06,* Computer Science dept. U.F.A.S University, Algeria.
3. Storm R.E. and S. Yemini, 1985. Optimistic Recovery in Distributed Systems, ACM Trans. Computer Systems, vol. 3, pp 204-226, Aug.
4. Cristian F, 1982. Exception Handling and Software Fault Tolerance. IEEE Trans. On computers, C.31 pp. 531-539.
5. Netzer R. and J. Xu, 1995. Necessary and Sufficient Conditions for Consistent Global Snapshots, IEEE Trans on Parallel and Dist. System, Feb.
6. Elnozahy E. N., L. Alvisi, Y. M. Wang and D. B. Johnson, 2002. A Survey of Rollback-Recovery Protocols in Message-Passing Systems, ACM Comp. Surveys, Sep.
7. Alvisi L. and K. Marzullo, 1998. .Message Logging: Pessimistic, Optimistic, Causal and Optimal, IEEE Transactions on Software Engineering, Feb.
8. Randell B, 1975. System Structure for Software Fault Tolerance. IEEE Trans. Soft. Eng.SE 1, 2 pp. 220-232,
9. Russell D.L, 1980. State Restoration in Systems of Communicating Processes. IEEE Trans. soft. Eng., vol. 6 pages 183-194, Mar.
10. Janakiraman G.F., J.R. Santos, D. Subharaveti, and Y. Turner, 2005. Cruz: Application-Transparent Distributed Checkpoint-Restart on Standard Operating Systems. DSN Yokohama Japan
11. Gao Q., W. Yu, W. Huang, and D.K. Panda., 2006. Application-Transparent Checkpoint/Restart for MPI programs over infiniBand. Proceedings of intl. conf. on parallel Processing, pp. 471-478.
12. Amati G. and others, 2005. Early Experience with Scientific Application on the Bleue Gene/L supercomputer, Springer, vol. 36 (48).