International Journal of Scientific Research & Engineering Trends
Volume 8, Issue 4, Jul-Aug-2022, ISSN (Online): 2395-566X

# An Exploration of Methods for Empathetic Cluster Formation Using Mobile Computing Systems

**Naheeda Zaib, Saiba Jan**
NIMS School of Data Science and Engineering,
NIMS University,
Rajasthan Jaipur
Naheedazaib93@gmail.com, Saibjan92@gmail.com

**Abstract- A distributed system is a collection of independent units working to solve a problem that none could solve on their own. Specific tasks in a distributed system known as smartphones are carried out on base stations, whose location within the network changes over time. Distributed mobile systems introduce new issues such as mobility, a lack of a reliable, consistent store on mobile nodes, poor wireless frequency band, interruptions, and limited battery life. This paper discusses the problem of fault-tolerant computing in mobile distributed databases. The given processes are built on the concepts of checkpointing and flip restoration. We have also solved the challenge of recovering from simultaneous failures in a distributed computing framework. We have developed a novel strategy in which we have successfully dealt with lost and orphaned messages.**

**Keywords- mobility, martphones etc.**

## I. INTRODUCTION

Distributed & cluster computers are extensively used because of their accessibility, scalability, and capacity to meet robust computational requirements; nevertheless, as the number of features rises, so does the risk of failure. Fault tolerance provides a deep understanding of the many fault conditions that might occur in these networks. The two primary kinds of flaws are permanent and temporary. Modifications cause transient faults, whereas long-term degradation to one or more elements causes permanent problems. Endless conflicts can be addressed by component repair and replacement. Since they only exist for a short time, transient flaws are brutal to identify and correct. Therefore, fault tolerance becomes crucial, particularly for brief failures in distributed applications.

A computer can complete tasks by utilizing fault-tolerant techniques, such as fault identification, localization, confinement, and restoration. Repairing or replacing a gear can solve persistent issues. Due to their transient nature, transient flaws are problematic to find and rectify. Determining fault tolerance is crucial, particularly for brief failures in distributed systems. Fault-tolerant methods, such as fault localization, containment, and recovery, can be used in a plan to complete tasks. These systems make use of various computing, transmission, and storage devices.

There are many different types of faults that can occur in a system, including hardware failure, environmental interference, software bugs, security flaws, and human error. The two types of problems that may be separated are permanent and temporary flaws. Permanent faults are errors that impair a specific system component in a permanent way. Recovery from persistent issues necessitates the regeneration of the destroyed systems and components reconfiguration. Temporary and not harmful in the long run are transient flaws. Recovery from momentary failures is more straightforward than long-lasting problems since layering and integration are unnecessary. It may be challenging to detect transient flaws even though they may disappear with no discernible impact on the system [8]. Redundancy of any kind may be used to provide fault tolerance. Both geographical and temporal redundancy is possible. Temporal redundancy, sometimes called checkpoint-restart, is the process of restarting an application once a problem arises by using an earlier checkpoint or recovering point.

The processing of specific applications may be lost if they cannot meet strict deadlines. Due to the program's ability to run many copies concurrently on different processors, severe timing constraints may be fulfilled where there is spatial redundancy. Spatial redundancy is a costly method of fault tolerance and may also need additional gear. If a temporary error occurs in research and commercial applications, the project's operation must be stopped and resumed from scratch. Due to this, the system must have a sufficient amount of fault-free time before the enormous applications can be completed.

The average programming time may exponentially grow over time if there are mistakes. Checkpointing is used chiefly to avoid losing any beneficial processing finished before a problem. As a portion of checkpointing, a project's information is periodically stored in a reliable storage medium. Upon discovery of a defect, the previous reliable condition is restored. Checkpointing enables the restart of a program's execution in case of a bug. The

quantity of helpful processing lost due to the issue is significantly decreased. The typical programmable performance with checkpointing only effect increases with the length of the program [8].

The technique of backward error recovery, commonly referred to as checkpoint-restart, is frequently inexpensive and doesn't require any additional hardware. Along with fault tolerance, checkpointing may impact roles, post-mortem analysis, process transfer, distributed applications debugging, identifying stable properties, and resuming [95].

**1. Two techniques exist for fixing mistakes:**
The types of faults and harm breakdowns must be carefully and accurately evaluated with forwarding error-correcting systems to eliminate such faults from the system. The procedure can go forward as it stands right now [70]. In a distributed system, that may not be possible to examine every failure critically.

Three phases make up backward error recovery. These include Regular checkpointing of the error-free state, restart from the recovered state, and recovery in case of failure.Reversible failure recovery is also known as checkpoint-restart and checkpoint-restore-restart. The checkpointing process is routinely used to advance the recovery line.

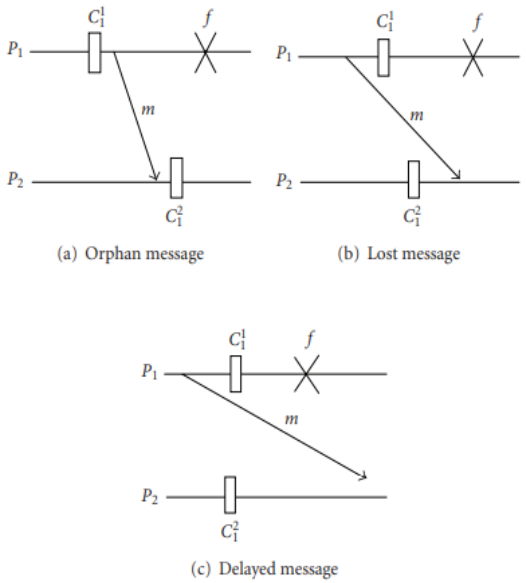


Fig 1. Types of messages in check pointing.

**2. Relative Processing Framework and Process Model**
**2.1 The System Model**: The distributed system provides features [13]: operations do not exchange memory, but interact by messages transmitted through pathways, processes are deterministic, and processes fail to continue running when they encounter an obstacle. A system has to be brought back to a stable

**2.2. Relevant Data Models.** The following information structures are required for each process for the suggested recovery strategy to function. Consider a group of n processes (P1, P2,..., Pn) executing a distributed algorithm. In other words, the kth service message will have k as its sequence number. This is because service messages are assumed to be piggybacking on individual checksum. Relative Processing Framework and Process Model.

The overall sequence of the messaging by each process is maintained using these sequence numbers. Process The interval between the Xth and Xth checkpoints of Pi is designated as (Ci x-Ci x1) and represents the period between these two checkpoints. At its Xth checkpoint Ci x, each process Pi keeps two vectors of size n, one for sending and the other for receiving. As soon as the system starts, these vectors are initialized to zero. As indicated below, these vectors.

- Vi x(sent) = [Si1 x , Si2 x , Si3 x , ... , Sin x ], where S ij x represents the largest sequence number of all messages sent by process Pi to process Pj in the interval (Ci x–Ci x−1). Note that Sii x = 0. (ii) Vi x(recv) = [Ri1 x , Ri2 x , Ri3 x , ... , Rin x ], where Rij x represents the largest sequence number of all messages received by Pi from Pj in the checkpointing interval (Ci x–Ci x−1). Also Rii x = 0.
- Forming a global system state from the collection of recorded processes and channel states. The underlying calculation is to be overlaid by the dynamic memory detection algorithm, which must operate concurrently with it without changing it [22].

**2.3 Disrupted Messages & Checkpointing Interval:**
We now explain why we believe the standard checkpointing interval T should be slightly more significant than the maximum message transmission duration between any two system processes. It is well known that the current solution for lost and delayed communications is message logging. So, it only follows the question of how soon a process can continue to log all messages just sent until a failure (if any) happens.

We've demonstrated below that a process Pi only has to preserve all of the messages. That has already been sent in the most recent checkpointing interval (Ci x-Ci x1) in its current state checkpoint Ci x because of the value of the shared checkpointing interval T given above. To put it another way, we can maintain consistency for a while after the server restarts by using the least amount of data feasible concerning the missed and postponed messages. Take a look at the circumstance in Figure 2. As previously mentioned, we will demonstrate a straightforward system with only two processes, but the fact holds for distributed systems with various techniques. You'll see that every message delivered by process Pi during its checkpointing interval (Ci x1-Ci x2) always reaches. Its recipient before

the most recent checkpoint Cj x of process Pj due to supposition that T, the length of the checkpointing interval, is equal to the value of T. Assumes that there is a loss f because it is depicted in the image. Please consider that the two processes will resume from their most recent Xth checkpoints following recovery.

Keep in mind that such message m doesn't need to be sent again because the received process Pj processes it before its most recent checkpoint Cj x. The inability of such a communication to be lost or delayed is therefore evident.

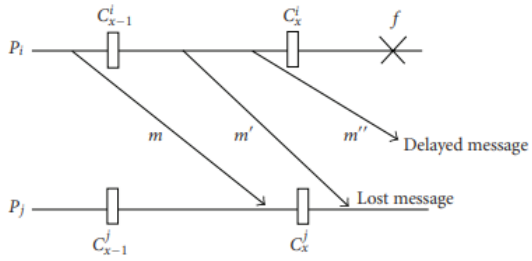


Fig 2. There can be no lost or delayed messages in message m.

As a result, the sender Pi need not register such messages at its most recent checkpoint Ci x. Nevertheless, messages delivered by operation Pi in the range (Ci x-Ci x1), such as m and m, might be missed or lost. Therefore, if there is a failure, f, we must log just these transmitted messages at the sender's most recent checkpoint Ci x. So that they may be sent again when the processes restart to prevent any inconsistencies in calculation after the system resets from the previous checkpoints.

Keep in mind that the message that is delayed, like message m, is effectively a message that is lost in the case of a failure. As a result, in our method, we only consider the most recent checkpoints of the processes, and the messages recorded at these checkpoints were just delivered recently.

### 3. Uncoordinated Checkpointing

In unstructured or autonomous checkpointing, processes are not required to coordinate their checkpointing operations; instead, each function separately maintains its checkpoint [14], [86], [96]. It allows each process to choose how to conduct a checkpoint with maximum autonomy, enabling them to use it and when it is more practicable. It eliminates coordination overhead after a defect and establishes a sustainable world state [14].

After a failure, a trustworthy global checkpoint is produced by keeping track of the dependencies. Because of the domino effect, it can need cascaded rollbacks to restore the system to its initial state [44].Many checkpoints must be kept for each process, and the trash collection technique is frequently used to retrieve no longer needed checkpoints. Under this paradigm, a method may take an unneeded checkpoint that would not have been a component of the constant global state. Unnecessary checkpoints waste time without advancing the recovery line [27].

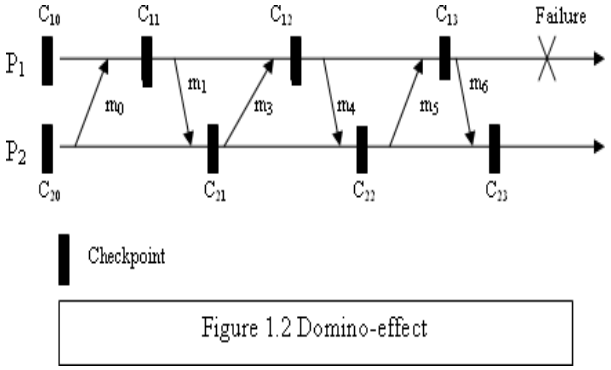


Fig 3. The chain reaction.

The major flaw in this method is the chain reaction. In this example, operations P1 & P2 have taken a set of checkpoints. According to the parallelism of messages and checkpoints, P1 and P2 only have one consistent checkpoint, the very first one at "C10, C20." Thus, if P1 fails, P1 and P2 must start the computation over [44]. It should be noted that the contradictory global state "C11, C21" is caused by orphan message m1. The local state "C12, C22" is also inconsistent due to the orphan message m4.

### 4. Co-ordinated checkpointing

The global state that results from coordination checkpoints or synchronous checkpoints is consistent. Frequently, the submit design is two-phase [22], [28], and [44]. Processes create temporary checkpoints in the first step and permanent ones in the second stage. One tentative and one permanent checkpoint can be stored, which is the main advantage. In case of failure, processes revert to the latest checkpointed state. An inflexible checkpoint cannot be reversed. It guarantees that the computations necessary to reach the checkpoint position won't be finished again. An interim checkpoint can be made permanent or overturned, though.

A straightforward fix obstructs communication when the synchronized checkpointing protocol is active [88]. The coordinator conducts a checkpoint and sends a message to each operation asking them to complete one. When a process receives a message, it immediately stops all computations, wipes all communication channels, performs a temporary checkpoint, and then sends a message of acknowledgment to the coordinator. After receiving revelations from all processes, the supervisor transmits a commitment message to conclude its two-phase checkpoint mechanism. When a process gets a commit, it converts its temporary checkpoint from speculative to permanent and, if there are any, discards its prior permanent checkpoint. The process is then free to carry on operating and engage in interposes communication. There are two types of coordinated

checkpointing techniques: blocking and non-blocking. Checkpointing in blocking algorithms, as was previously shown, results in some process blocking [44], [88].

Using non-blocking algorithms eliminates the necessity for process blocking [22], [28]. Algorithms for minimal and total processes can also be grouped to form coordinated checkpointing algorithms. Utilizing all-process coordinated checkpointing strategies, each process must initiate its checkpoint [22], [28]. In minimum-process algorithms, checkpoints must be taken during the commencement of minimum interacting processes [44].

## II. AIMING FOR CHECKPOINTS

**1. Checkpoint Continuity:**
A checkpointing strategy is used while the basic computation is being performed. To reduce checkpointing overheads, this should be. To prevent losing a substantial number of calculations in the case of an error, checkpointing should allow users to recover quickly. So necessitates frequent checkpointing and hence high overhead.

A sufficient number of checkpoints will be launched to ensure minimal overhead and minimal data loss costs as a result of failure. Both the probability of failure as well as the importance of computation have an impact on these. In a system that processes transactions, a checkpoint might be run after each operation, for example, if each transaction is essential and uncertainty is not permitted [42]. Checkpoint overhead is significantly raised as a result.

**2. Contents of Checkpoint:**
When something goes wrong, a process must have its state saved in a trustworthy place so that it may be restarted. The state/context also includes the ambient and the register contents, as well as the stack, data, and pieces of code. The environment holds all files, references, and information about the many currently used files. In message-passing systems, environment variables also contain delivered but unread messages. The background of that operation [42] is the understanding needed to carry out an action once it has been pre-empted.

**3. Overheads in the Checkpointing Algorithm:**
In a multiprocessor system, coordination costs and context-saving overhead are associated with each global checkpoint during a failure-free run. Process coordination is required in parallel/distributed systems to achieve a cohesive and comprehensive state. Special communications and data that are also piggybacked onto regular communications are used to establish process coordination. Coordination overhead is brought on by piggybacked information and particular control messages. Additionally, its expense is increased by the bookkeeping duties necessary to maintain coordination.

The time required to preserve a computation's whole context is known as the overhead of context saving. If each node does not have dependable storage, the context is transferred over the networks in a virtual machine. The latency in communication links is included in the overhead [42].

**4. Alternative Concepts:**
When processes connect by exchanging messages, it becomes difficult to have a complete ordering of events because dependencies are created between the events of different techniques. Lamport [52] proposed the concept of a relation called "happened before" to acquire the anticipated occurrences in a dispersed network (denoted by). This connection is irreflexive, antisymmetric, and transitive.

If occurrences "a" and "b are related events and a occurs earlier, b, therefore ab. When an event involves sending a message and an event b consists of receiving the same statement, the result is ab. Two instances, "a" and "b, are considered contemporary only if a doesn't occur before b and b doesn't happen before b. Local checkpoints are times when the state of a procedure on a processing unit is noted at a specific time. One from each phase, a few of the fellow checkpoints combine to form a global checkpoint. The overall position is constant if each event belongs to a concurrent set. A cohesive and comprehensive checkpoint is made up of a collection of local control points out of each procedure that is all synchronized with one another. Rollback recovery is continuing or returning to computation from a complete and coherent checkpoint.

The output of the fundamental computation is known as calculating messaging or just notifications, and they are denoted either by symbols mi or m. Pi shows the processes. The ith CI of a function is defined as the computation between its ith through (i+1)th checkpoints, including the ith stop but excluding the (i+1)th checkpoint. A plan of action Pi is only directly reliant on Pj if m exists such that I Pi got m that Pj sent out, II Pi hasn't achieved a lasting checkpoint after getting m, III Pi hasn't achieved a continuing. Checkpoint after receiving m, and IV Pj has not yet reached a continuing checkpoint before sending m. Direct dependencies can be stored in a bit array with a set duration for n operations at Pi. Declare ddvi[j. The assumption ddvi[j]=1 link between processes and minimum set computation [48], [64] implies that Pi is directly dependent on Pj.

## III. INTERESTED WORK

A examination of the literature reveals that fault-tolerant checkpointing has been the subject of several investigations. The majority of them were created by relaxing a number on Chandy & Lamport's (1985) presumptions. The main goal of improving the earlier

adaptations to Chandy and or Lamport's (1985) methodology was to lower the operational expenses of coordinating among tasks in a multicore processor. A few ways for checking pointing shared-memory multi processors have been developed in order to ensure consistent memory.

In essence, these methods broaden cache coherence protocols. These techniques make the assumption that perhaps the ram is safe while not storing context on a disc. Recently, shared memory distributed systems have been proposed as a technique. The virtual global memory's cache coherence must be preserved enabling checkpoints in such systems. Due to the dispersed nature of physical memory, it is crucial to store primary memory data on a disc. Sensory saving latency is thus higher than in shared memory systems. We should also point out that the majority of solutions do not assume that users have previous knowledge of the programme structure designed for multiprocessor execution.

Approaches for distributed systems and their communications costs have been developed with the assumption that hosts' locations within the network don't change and its connections is stable in the lack of faults. The advent of cellphones has rendered these assumptions obsolete. Additionally, mobile hosts must adhere to rigorous regulations regarding their power usage, and the wireless links which join M.H.s to nearby M.S.Ss have a specific bandwidth.

The Chandy-Lamport [22] method is one of the early non-blocking, only those coordinated checkpointing techniques for static nodes. Since every network route is used in this method, the message complexity is O(N2), making FIFO channel ordering necessary. Lai and Yang [50] proposed an approach to relax the FIFO assumption. In this approach, a signal is piggybacked onto the signal that is sent across each channel when a process reaches a checkpoint.

The receiver scans just the piggybacked flag while executing the signal to determine if a checkpoint is required. If that's so, another checkpoint is run to avoid inconsistency before the signal is handled. As part of state checkpoints, each process must preserve all previous messages sent along each path in order to collect channel information. There must be checkpoints in every procedure. Elnozahy et al. suggested an all-process synchronous checkpointing method that is non-blocking and has message complexity of O. (N). By employing checkpoint sequence numbers to identify orphan messages, they lessen the need to interrupt processes during checkpointing. However, this technique requires interaction in between initiator and each processing step. According to the Silva & Silva [85] technique, the operations that could not interact during the previous checkpointing period do not need to take fresh checkpoints. These two methods [28], [85] both assume that a prominent initiator choose the ideal moment to begin the checkpointing activity. They therefore encounter centralised algorithms' limitations, like one failures, traffic bottlenecks, etc.

To accomplish this in the minimum level method, Cao and Singhal [20] proposed the concept of changing checkpoints. According to their technique, an initiator, like Pi, will send the checkpoint request toward any process, like Pj, if Pi has previously won m from Pj during the current CI. If Pj has sent m to Pi in the present CI, Pj acknowledges its preliminary checkpoint; otherwise, Pj decides the demand for just a checkpoint is useless. Similar to something like this, when Pj creates its temporary checkpoint, it advertises the request to other processes.

This approach is repeated until all actions upon which initiate transitively relies are reached, at which time a checkpointing tree is created. Suppose Pj transmits m and Pj already has taken numerous checkpoints within the current beginning before sending m. In that case, Pi could be instructed to take another checkpoint termed as just a mutable checkpoint during checkpointing. If it does not fall under the minimum threshold and is removed at commit, Pi's changeable checkpoint is useless. The massive data architecture M.R. [] is also associated with both the checkpoint requests to reduce the number of unnecessary checkpoint requests. Each operation communicates its response to the initiator right away.

That technique [20] was built to accommodate concurrent executions using the approach outlined in [73]. According to Ni et al. [61], the Cao-Singhal process [20] may produce inconsistencies when executed concurrently. To accommodate concurrent executions, the authors [61] modified the technique proposed in [20]. The number of needless checkpoints in [20] might be rather significant in some cases [48].

L. Kumar et al. [48] and P. Kumar et al. [64] reduced the depth of the synchronization tree and the number of unnecessary checkpoints. By maintaining non-intrusiveness, but at the additional cost of keeping and trying to gather physiological reliance matrices, computing the required minimum, and conveying it on the knowable system along with the checkpoint proposal. In step [48, before sending the dependent vector and obtaining the most miniature set, Pi examines the message given by Pj to see which of the following conditions hold:

- Pj did not execute any checks for such a current beginning before sending m since Pj is indeed a direct dependent of Pi.
- Pj has established several long-term checkpoints since broadcasting m.
- For the continuing start, Pi has already finished its produced checkpoint.

- Pi has already achieved its induced checkpoint for this commencement.
- Since the last committed checkpoint, Pi has not sent any messages.
- If not, Pi executes its inspired checkpoint, which is similar to a mutable checkpoint, before processing m.

Suppose Pi learns after acquiring the minimal set that it was not a part of the minimal set. In that case, it eliminates its initial checkpoint or converts whatever inspired checkpoints it does have into a tentative one. A checkpointing tree is not produced by this method. A process is instructed to take its preliminary checkpoint by the algorithm [64] if this is in the minimum set when it gets the minimal set; otherwise, it is told to refuse the requests. If a process Pi depends on a procedure Pj directly but Pj is not in the estimated minimum set; When Pi executes its tentative checkpoint, Pi submits the checkpoint request to Pj.

When Pi receives m from Pj, Pi only runs m after performing its triggered checkpoints if the following conditions are met: Before transmitting, I Pj double-checked a few items at the beginning. (ii) During this initiation, Pi hasn't stopped to take any checkpoints. (iii) Since the previous permanent checkpoint, Pi has only possibly delivered one message. Upon commit, if Pi learns it isn't a part of the group, it discards any inspired checkpoints it may have created if it wasn't a part of the minimum set. The main goal of the suggestions made in [64] & [48] is to minimize the length of time that a procedure may be forced to wait at a checkpoint that is induced or changeable before taking. By reducing this time, the number of unnecessary checkpoints is automatically reduced.

## REFERENCES

[1] Acharya A. and Badrinath B. R., "Checkpointing Distributed Applications on Mobile Computers," Proceedings of the 3rd International Conference on Parallel and Distributed Information Systems, pp. 73-80, September 1994.
[2] Acharya A., "Structuring Distributed Algorithms and Services for networks with Mobile Hosts", Ph.D. Thesis, Rutgers University, 1995.
[3] Alvisi, Lorenzo and Marzullo, Keith, "Message Logging: Pessimistic, Optimistic, Causal, and Optimal", IEEE Transactions on Software Engineering, Vol. 24, No. 2, February 1998, pp. 149-159.
[4] L. Alvisi, Hoppe, B., Marzullo, K., "Nonblocking and Orphan-Free message Logging Protocol," Proc. of 23rd Fault-Tolerant Computing Symp., pp. 145-154, June 1993.
[5] L. Alvisi, "Understanding the Message Logging Paradigm for Masking Process Crashes, "Ph.D. Thesis, Cornell Univ., Dept. of Computer Science, Jan. 1996. Available as Technical Report TR-96-1577.
[6] L. Alvisi and K. Marzullo, "Tradeoffs in implementing Optimal Message Logging Protocol", Proc. 15th Symp. Principles of Distributed Computing, pp. 58-67, A.C.M., June, 1996.
[7] Adnan Agbaria, William H Sanders, "Distributed Snapshots for Mobile Computing Systems", IEEE Intl. Conf. PERCOM "04, pp. 1-10, 2004.
[8] Avi Ziv and Jehoshua Bruck, "Checkpointing in Parallel and Distributed Systems", Book Chapter from Parallel and Distributed Computing Handbook edited by Albert Z. H. Zomaya, pp. 274-302, Mc Graw Hill, 1996.
[9] A. Borg, J. Baumbach, and S. Glazer, "A Message System Supporting Fault Tolerance", Proc. Symp. Operating System Principles, pp. 90-99, ACM SIG OPS, Oct. 1983.
[10] Adnan Agbaria, William H. Sanders, "Distributed Snapshots for Mobile Computing Systems", Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications (Percom" 04), pp. 1-10, 2004.
[11] Baldoni R., Hélary J-M., Mostefaoui A. and Raynal M., "Rollback Dependency Trackability: A Minimal Characterization and its Protocol", Information and Computation, 165, pp. 144-173, 2003.
[12] Baldoni R., Hélary J-M., Mostefaoui A. and Raynal M., "A Communication- Induced Checkpointing Protocol that Ensures Rollback-Dependency Trackability," Proceedings of the International Symposium on Fault-Tolerant-Computing Systems, pp. 68-77, June 1997.
[13] Bhagwat P., and Perkins, C.E., "A mobile Networking System based on Internet Protocol (I.P.)",USENIX Symposium on Mobile and Location-Independent Computing, August 1993.
[14] Bhargava B. and Lian S. R., "Independent Check pointing and Concurrent Rollback for Recovery in Distributed Systems-An Optimistic Approach," Proceedings of 17th IEEE Symposium on Reliable Distributed Systems, pp. 3- 12, 1988.
[15] G. Barigazzi and L. Strigni, "Application-Transparent Setting of Recovery Points", Digest of Papers Fault-Tolerant Computing Systems-13, pp. 48-55, 1983.
[16] Badrinath B. R, Acharya A., T. Imielinski "Structuring Distributed Algorithms for Mobile Hosts", Proc. 14th Int. Conf. Distributed Computing Systems, June 1994.
[17] Badrinath B. R, Acharya A., T. Imielinski "Designing Distributed Algorithms for Mobile Computing Networks", Computer Communications, Vol. 19, No. 4, 1996.
[18] Cao G. and Singhal M., "On coordinated checkpointing in Distributed Systems", IEEE Transactions on Parallel and Distributed Systems, vol. 9, no.12, pp. 1213-1225, Dec 1998.

[19] Cao G. and Singhal M., "On the Impossibility of Min-process Non-blocking Checkpointing and an Efficient Checkpointing Algorithm for Mobile Computing Systems," Proceedings of International Conference on Parallel Processing, pp. 37-44, August 1998.
[20] Cao G. and Singhal M., "Mutable Checkpoints: A New Checkpointing Approach for Mobile Computing systems," IEEE Transaction On Parallel and Distributed Systems, vol. 12, no. 2, pp. 157-172, February 2001.
[21] Cao G. and Singhal M., "Checkpointing with Mutable Checkpoints", Theoretical Computer Science, 290(2003), pp. 1127-1148.
[22] Chandy K. M. and Lamport L., "Distributed Snapshots: Determining Global State of Distributed Systems," A.C.M. Transaction on Computing Systems, vol. 3, No. 1, pp. 63- 75, February 1985.
[23] F. Cristian and F. Jahanian, "A timestamp-based Checkpointing Protocol for Long-Lived Distributed Computations", Proc IEEE Symp. Reliable Distributed Systems, pp. 12-20, 1991.
[24] David R. Jefferson, "Virtual Time", A.C.M. Transactions on Programming Languages and Systems, Vol. 7, NO.3, pp 404-425, July 1985.
[25] Dang Y., Park, E.K, "Checkpointing and Rollback-Recovery Algorithms in Distributed Systems", Journal of Systems and Software, pp. 59-71, April 1994.
[26] Dieter Kranzlmuller, Nam Thoai, Jens Volkert, "Error Detection in Large Scale Parallel Programs with Long runtimes, Future Generation Computer Systems 19, pp. 689- 700, 2003.
[27] Elnozahy E.N., Alvisi L., Wang Y.M. and Johnson D.B., "A Survey of Rollback-Recovery Protocols in Message-Passing Systems," A.C.M. Computing Surveys, vol. 34, no. 3, pp. 375- 408, 2002.
[28] Elnozahy E.N., Johnson D.B. and Zwaenepoel W., "The Performance of Consistent Checkpointing," Proceedings of the 11th Symposium on Reliable Distributed Systems, pp. 39-47, October 1992.
[29] Elnozahy and Zwaenepoel W, "Manetho: Transparent Roll-back Recovery with Low-overhead, Limited Rollback and Fast Output Commit," IEEE Trans. Computers, vol. 41, no. 5, pp. 526-531, May 1992.
[30] Elnozahy and Zwaenepoel W, " On the Use and Implementation of Message Logging," 24th int"l Symp. Fault-Tolerant Computing, pp. 298-307, IEEE Computer Society, June 1994.
[31] George H. Forman and John Zahorjan, "The Challenges of Mobile Computing", IEEE Computers vol. 27, no. 4, April 1994, pp. 38-47.
[32] Richard C. Gass and Bidyut Gupta, "An Efficient Checkpointing Scheme for Mobile Computing Systems", European Simulation Symposium, Oct 18-20, 2001, pp. 1-6.
[33] Hélary J. M., Mostefaoui A. and Raynal M., "Communication-Induced Determination of Consistent Snapshots," Proceedings of the 28th International Symposium on Fault-Tolerant Computing, pp. 208-217, June 1998.
[34] Higaki H. and Takizawa M., "Checkpoint-recovery Protocol for Reliable Mobile Systems," Trans. of Information processing Japan, vol. 40, no.1, pp. 236-244, Jan. 1999.
[35] Higaki H. and Takizawa M., "Recovery Protocol for Mobile Checkpointing", IEEE 9th International Conference on Database Expert Systems Applications, Viena, pp. 520-525, 1998
[36] Higaki H. and Takizawa M., "Checkpoint Recovery Protocol for Reliable Mobile Systems", 17th Symposium on Reliable Distributed Systems, pp. 93-99, Oct. 1998.
[37] Ioannidis, J., Duchamp, D., and Maguire, G.Q., "IP-based protocols for Mobile Internetworking", In Proc. of ACM SIGCOMM Symposium on Communications, Architectures, and Protocols, pp. 235-245, September 1991.
[38] Johnson, D.B., Zwaenepoel, W., "Sender-based message logging", In Proceedings of 17th international Symposium on Fault-Tolerant Computing, pp 14-19, 1987.
[39] Johnson, D.B., Zwaenepoel, W., "Recovery in Distributed Systems using optimistic message logging and checkpointing. In 7th A.C.M. Symposium on Principles of Distributed Computing, pp 171-181, 1988.
[40] D. Johnson, "Distributed System Fault Tolerance Using Message Logging and Checkpointing," Ph.D. Thesis, Rice Univ., Dec. 1989.
[41] JinHo Ahn, Sung-Gi Min, Chong-Sun Hwang, "A Causal Message Logging Protocol for Mobile Nodes in Mobile Computing Environments", Future Generation Computer Systems 20, pp 663-686, 2004.
[42] Kalaiselvi, S., Rajaraman, V., "A Survey of Check pointing Algorithms for Parallel and Distributed Systems", Sadhna, Vol. 25, Part 5, October 2000, pp. 489-510.
[43] Kistler, J., and Satyanarayana, M., "Disconnected Operation in the Coda file system", A.C.M. Trans. on Computer Systems 10, 1 (Feb. 1992).
[44] [44] Koo R. and Toueg S., "Checkpointing and Roll-Back Recovery for Distributed Systems," IEEE Trans. on Software Engineering, vol. 13, no. 1, pp. 23-31, January 1987.
[45] J.L. Kim, T. Park, "An efficient Protocol for checkpointing Recovery in Distributed Systems," IEEE Trans. Parallel and Distributed Systems, pp. 955-960, Aug. 1993.
[46] Kyne-Sup BYUN, Sung_Hwa L.I.M., Jai-Hoon K.I.M., "Two- Tier Checkpointing Algorithm Using M.S.S. in Wireless Networks", IEICE Trans. Communications, Vol E86-B, No. 7, pp. 2136-2142, July 2003.
[47] L. Kumar, M. Misra, R.C. Joshi, "Checkpointing in Distributed Computing Systems" Book Chapter

"Concurrency in Dependable Computing", pp. 273-92, 2002.
[48] L. Kumar, M. Misra, R.C. Joshi, "Low overhead optimal checkpointing for mobile distributed systems" Proceedings. 19th IEEE International Conference on Data Engineering, pp 686 – 88, 2003.
[49] Lalit Kumar, Parveen Kumar, R K Chauhan, "Logging based Coordinated Checkpointing in Mobile Distributed Computing Systems", IETE Journal of Research, vol. 51, no. 6, pp. 485-490, 2005.
[50] T.H. Lai and T.H. Yang, "On Distributed Snapshots", Information Processing Letters, vol. 25, pp. 153-158, 1987.
[51] P.J. Leu and B.Bhargawa, "Concurrent Robust Checkpointing and Recovery in Distributed Systems", Proceeding Fourth Intl Conf. Data Engg. Pp. 154-163, Feb. 1988.
[52] L. Lamport, "Time, clocks and ordering of events in a distributed system" Comm. A.C.M., vol.21, no.7, pp. 558- 565, July 1978.
[53] Lalit Kumar, Parveen Kumar, R K Chauhan, "Pitfalls in Minimum-process Coordinated Checkpointing protocols for Mobile Distributed", ACCST Journal of Research, Volume III, No. 1, 2005 pp. 51-56.
[54] Lalit Kumar, Parveen Kumar, R K Chauhan, "Message Logging and Checkpointing in Mobile Computing", Journal of Multi-disciplinary Engg. Technologies, Vol.1, No.1, 2005, pp. 61-66.
[55] Manivannan D. and Singhal M., "Quasi-Synchronous Checkpointing: Models, Characterization, and Classification," IEEE Trans. Parallel and Distributed Systems, vol. 10, no. 7, pp. 703-713, July 1999.
[56] Manivannan D., Netzer R. H. and Singhal M., "Finding Consistent Global Checkpoints in a Distributed Computation," IEEE Transactions on Parallel & Distributed Systems, vol. 8, no. 6, pp. 623-627, June 1997.
[57] Yoshifumi Manabe, "A Distributed Consistent Global Checkpoint Algorithm for Distributed Mobile Systems", 8th Int" l Conference on Parallel and Distributed Systems", pp. 125-132, 2001.
[58] Mannivannam, D., Singhal, M., "Failure Recovery based on Quasi-Synchronous Checkpointing in Mobile Computing Systems", In T.R. No. OSU-CISRC-7/96-TR-36, Dept of Computer and Information Science, The Ohio State University, 1996.
[59] Mannivannam, D., Singhal, M., "A Low overhead Recovery Techniques using Quasi Synchronous Checkpointing", Proc. 16th int "l conf. Distributed Computing Systems, pp 100-107, May 1996.
[60] Yoshinori Morita, Kengo Hiraga and Hiroaki Higaki, "Hybrid Checkpoint Protocol for Supporting Mobile-to-Mobile Communication", Proc. Of the International Conference on Information Networking, 2001.
[61] Ni, W., S. Vrbsky and S. Ray, "Pitfalls in Distributed Nonblocking Checkpointing", Journal of Interconnection Networks, Vol. 1 No. 5, pp. 47-78, March 2004.
[62] Netzer, R.H. and Xu,J , "Necessary and Sufficient Conditions for Consistent Global Snapshots", IEEE Trans. Parallel and Distributed Systems 6,2, pp 165-169, 1995.
[63] Neves N. and Fuchs W. K., "Adaptive Recovery for Mobile Environments," Communications of the A.C.M., vol. 40, no. 1, pp. 68-74, January 1997.
[64] Parveen Kumar, Lalit Kumar, R K Chauhan, V K Gupta "A Non-Intrusive Minimum Process Synchronous Checkpointing Protocol for Mobile Distributed Systems" Proceedings of IEEE ICPWC-2005, January 2005.
[65] Parveen Kumar, Lalit Kumar, R K Chauhan, "A low overhead Non-intrusive Hybrid Synchronous checkpointing protocol for mobile systems", Journal of Multidisciplinary Engineering Technologies, Vol.1, No. 1, pp 40-50, 2005.
[66] Parveen Kumar, Lalit Kumar, R K Chauhan, "Synchronous Checkpointing Protocols for Mobile Distributed Systems: A Comparative Study", International Journal of information and computing science, Volume 8, No.2, 2005, pp 14-21.
[67] Parveen Kumar, Lalit Kumar, R K Chauhan, "A Hybrid Coordinated Checkpointing Protocol for Mobile Computing Systems", and IETE Journal of research, Vol 52, No. 2&3, pp 247-254, 2006.
[68] Parveen Kumar, Lalit Kumar, R K Chauhan, "A Synchronous Checkpointing Protocol for Mobile Distributed Systems: A Probabilistic Approach, Accepted for Publication in International Journal of Information and Computer Security.
[69] Pradhan D.K., Krishana P.P. and Vaidya N.H., "Recoverable Mobile Environment: Design and Trade-off Analysis," Proceedings 26th International Symposium on Fault-Tolerant Computing, pp. 16-25, 1996.
[70] Pradhan D.K. and Vaidya N., "Roll-forward Checkpointing Scheme: Concurrent Retry with Non-dedicated Spares," Proceedings of the IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems, pp. 166-174, July 1992.
[71] Pushpendra Singh, Gilbert Cabillic, "A Checkpointing Algorithm for Mobile Computing Environment", LNCS, No. 2775, pp 65-74, 2003.
[72] Prakash R. and Singhal M., "Low-Cost Checkpointing and Failure Recovery in Mobile Computing Systems," IEEE Transaction On Parallel and Distributed Systems, vol. 7, no. 10, pp. 1035-1048, October1996.
[73] Prakash R. and Singhal M., "Maximum Global Snapshot with Concurrent Initiations", Proc. Sixth IEEE Symp. Parallel and Distributed Processing, pp. 344-51, Oct. 1994.
[74] M.L. Powell and D.L. Presotto, "Publishing: A Reliable Broadcast Communication Mechanism", Proc. ninth Symp. Operating System Principles, pp. 100-109, ACM SIGOPS, Oct. 1983.

[75] Purnendu Sinha, Da Qi Ren, "Formal Verification of Dependable Distributed Protocols", Information and Software Technology, 45, pp. 873-888, 2003.
[76] Quaglia, F., Cipriani, R., Baldoni, R., "Checkpointing Protocols in Distributed Systems with Mobile Hosts: A Performance Analysis", IPPS/SPDP Workshop, pp. 742-755, 1998.
[77] Randall, B, "System Structure for Software Fault Tolerance", IEEE Trans. on Software Engineering, 1, 2, 220- 232, 1975.
[78] Russell, D.L., "State Restoration in Systems of Communicating Processes", IEEE Trans. Software Engineering, 6,2. 183-194, 1980.
[79] Ramanathan, P. and K.G. Shin, "Use of Common Time Base for Checkpointing and Rollback Recovery in a Distributed System", IEEE Trans. Software Engg., pp. 571- 583, June 1993.
[80] R K Chauhan, Parveen Kumar, Lalit Kumar, "A coordinated checkpointing protocol for mobile computing systems", International Journal of information and computing science, Accepted for Publication, Vol 9, No. 1, 2006.
[81] R K Chauhan, Parveen Kumar, Lalit Kumar, "Hybrid and intrusive synchronous checkpointing protocols for mobile distributed systems", Accepted for publication in ACCST Journal of Research, Volume IV, No. 4, 2006
[82] R K Chauhan, Parveen Kumar, Lalit Kumar, "Non-intrusive Coordinated Checkpointing Protocols for Mobile Computing Systems : A Critical Survey, ACCST Journal of Research, to be published in Volume IV, No. 3, 2006.
[83] R K Chauhan, Parveen Kumar, Lalit Kumar, "Checkpointing Distributed Applications on Mobile Computers", Journal of Multidisciplinary Engineering and Technologies, Vol. 2 No.1, Jan. 2006.
[84] Ssu K.F., Yao B., Fuchs W.K. and Neves N. F., "Adaptive Checkpointing with Storage Management for Mobile Environments," IEEE Transactions on Reliability, vol. 48, no. 4, pp. 315-324, December 1999.
[85] Silva, L.M. and J.G. Silva, "Global checkpointing for distributed programs", Proc. 11th symp. Reliable Distributed Systems, pp. 155-62, Oct. 1992.
[86] Storm R., and Termini, S., "Optimistic Recovery in Distributed Systems", A.C.M. Trans. Computer Systems, Aug, 1985, pp. 204-226.
[87] A.P. Sistla and J.L. Welch, "Efficient Distributed Recovery Using Message Logging", Proc. 18th Symp. Principles of Distributed Computing", pp 223-238, Aug. 1989.
[88] Tamir, Y., Sequin, C.H., "Error Recovery in multi-computers using global checkpoints", In Proceedings of the International Conference on Parallel Processing, pp. 32-41, 1984.
[89] Terakota, F., Yokote, Y., and Tokoro, M., "A Network Architecture providing host migration transparency", Proc, of ACM SIGCOMM 91, September 1991.
[90] S. Venkatesan and T.Y. Juang, "Efficient Algorithms for Optimistic Crash recovery", Distributed Computing, vol. 8, no. 2, pp. 105-114, June 1994.
[91] S. Venkatesan, "Message-Optimal Incremental Snapshots", Computer and Software Engineering, vol.1, no.3, pp. 211- 231, 1993.
[92] S. Venkatesan, "Optimistic Crash recovery Without Rolling back Non-Faulty Processors", Information Sciences, 1993.
[93] S. Venkatesan and T.T.Y. Juang, "Low Overhead optimistic crash Recovery", Proc. 11th Int. Conf. Distributed Computing Systems, pp. 454-461, 1991.
[94] Wada H., Yazawa, T., Ohnishi, T. and Tanaka, Y., "Mobile Computing Environment based on internet packet forwarding", Winter Usenix, Jan. 1993.
[95] Wang Y. M., Huang Y., Vo K.P., Chung P.Y. and Kintala C., "Checkpointing and its Applications," Proceedings of the 25th International Symposium on Fault-Tolerant Computing (FTCS-25),pp. 22-31, June 1995.
[96] Wood, W.G., "A Decentralized Recovery Control Protocol", 1981 IEEE Symposium on Fault-Tolerant Computing, 1981.
[97] Wang Y. and Fuchs, W.K., "Lazy Checkpoint Coordination for Bounding Rollback Propagation," Proc. 12th Symp. Reliable Distributed Systems, pp. 78-85, Oct. 1993.
[98] Bin Yao, Kuo-Feng Ssu & W. Kent Fuchs, "Message Logging in Mobile Computing", Proceedings of international conference on FTCS, pp 294-301, 1999.
[99] Yasuo Sato, Michiko Inoue, Toshimitsu Masuzawa, Hideo Fujiwara, "A Snapshot Algorithm for Distributed Mobile Systems" Proceedings of the 16th ICDCS, pp734-743,1996.