ISSN:2231-0711
www.ijcset.net

Specification of Important Features in Fault Tolerance for Distributed Systems

Amir Massoud Bidgoil[#1], Ali Pajoohanfar[*2], Maryam Amanifar[†3]

[#]*Department of computer engineering ,Islamic Azad University, Tehran North Branch, Tehran, Iran*
am_bidgoli@iau-tnb.ac.ir

[*]*SAMA Technical and Vocational Training college, Islamic Azad University, Ahvaz, Iran*
a.pajooh@gmail.com

[†]*Khoramshahr Branch, Islamic Azad University, Khoramshahr, Iran*
m.amanifar@khouzestan.srbiau.ac.ir

***Abstract*— Performance of a checkpointing based multiple fault tolerance is low. The main reason is overheads associate with checkpointing. A checkpointing algorithm can be improved by improved storing strategy and checkpointing scheduling. Improved storage strategy and checkpointing scheduling will reduce the overheads associated with checkpointing. Performance and efficiency is most desirable feature of recovery based on checkpointing. In this paper important critical issues involved in fast and efficient recovery are discussed based on checkpointing. Impact of each issue on performance of checkpointing based recovery is also discussed. Relationships among issues are also explored. Finally comparisons of important issues are done between coordinated checkpointing and uncoordinated checkpointing.**

***Keywords*- Checkpointing, Distributed System, Recovery, Fault Tolerance**

I. INTRODUCTION

Checkpoint with rollback-recovery is a well-known technique to tolerate process crashes and failures in distributed system. In order to tolerate crash of process failure one of approach is create a new process of that process with same state. This can be done if state and complete description of all processes executing in distributed environment must be saved on stable stored time and time and when any crash failure of processes are detected this save checkpoints are used to crate new processes identical to crashed processes and this way multiple process crashed can be tolerated. Checkpoint is an operation which stores the current state of computation in stable storage. Checkpoints are established during the normal execution of a program periodically. This information is saved on a stable storage so that it can be used in case of node failures. The information includes the process state, its environment, and the value of registers. A fundamental goal of any rollback recovery protocol is to bring the system into a consistent state when inconsistencies occur because of a failure [1]

II. TYPES OF CHECKPOINTING AND CORRESPONDING RECOVERY ISSUES

Coordinated checkpoint and uncoordinated checkpoint associated with message logging are the two main techniques used for saving the distributed execution state and recovering from system failures [2].

A. Coordinated checkpoint

In coordinate check point processes coordinate their checkpoints in order to save a system-wide consistent state. Coordinate check points are consistent set of checkpoints. These consistent check points are used to bound rollback propagation. Consistency is more in case of coordinate check points due to consistent set of checkpoints [3].

Coordinated checkpoint involves the rollback check point of all processes from the last snapshot when a faulty situation is detected, even when a single process crashes. For this reason recovery time is very large and it makes unsuitable for real time applications. In case of frequent failures and multiple faults coordinate check point technique can not be used. Performance can be improved by decreasing the recovery time .Main reason for large recovery time is restarting all the initial state. Recovery time can be reduced by enabling the restart from last correct state instead of from very first state. There must be some mechanism to ensure restarting from last correct state will reach a state matching the rest of the system, as before the crash. Checkpointing is only taken when all process agree for a consistent state. There are two main ways to implement coordinate checkpointing; blocking and none blocking. Blocking checkpointing consists of stopping the computation to take the global state. This permits better control on the state of the different processes and their communication channels. The second one, called non-blocking coordinated checkpointing, does not provide this kind of control, but does not require the interruption of the computation [4].

B. Uncoordinated checkpoint with message logging

In Uncoordinated checkpoint protocols, all processes execute a checkpoint independently of the others so that recovery can be done independently with each other. There is a big question if checkpoints are taken independently than how complete and overall description and order of process execution is determined. One way is uncoordinated checkpointing is combined with message logging.

Message logging is a common technique used to build systems that can tolerate process crash failures. These protocols require that each process periodically records its local state and log the messages received since recording that state. Message logging stores all interprocess messages in order to bring checkpoint up to date. When a process crashes, a new process is created in its place: the new process is given the appropriate recorded local state, and then it replays the logged messages in the order they were originally received [5].Message logging is combined with uncoordinated checkpoint to restart the system from last correct state. It is combined with message logging to ensure the complete description of a process execution state in case of its failure. Besides logging of all received messages, re-sending the same relevant messages in the same order to the crashed processes during their re-execution is also main function of message logging. There are three kinds of message logging protocols: optimistic, pessimistic and causal. Pessimistic protocols ensure that all messages received by a process are logged on reliable media before it sends information in the system. Logged information on reliable media can be re-sent later and only if necessary during rollback. Message logging optimistic protocols just ensure that all messages will eventually be logged. So, one usual way to implement optimistic logging is to log the messages on non-reliable media. Causal protocols log message information of a process in all causally dependent processes [6].

Uncoordinated checkpointing when used with message logging having fast recovery since restart is from last consistent state not from initial sate as in case of coordinate checkpointing. Since checkpointing is done independently hence multiple faults can be handled by this approach which can not be handled by coordinated checkpointing.

III. ISSUES WITH CHECKPOINTING BASED RECOVERY: IN THIS SECTION WE DISCUSSED CRITICAL AND IMPORTANT ISSUES RELATED TO CHECKPOINTING BASED FAULT TOLERANCE

A. Recovery cost

Conventional rollback-recovery protocols redo the computation of the crashed process since the last checkpoint on a single processor. As a result, the recovery time of all protocols is no less than the time between the last checkpoint and the crash. Researcher proposed a new application-level fault-tolerant approach for parallel applications called the Fault-Tolerant Parallel Algorithm (FTPA), which provides fast self-recovery. When fail-stop failures occur and are detected, all surviving processes recomputed the workload of failed processes in parallel. FTPA, however, requires the user to be involved in fault tolerance. In order to ease the FTPA implementation, Researcher developed Get it Fault-Tolerant (GiFT), a source-to-source precompiled tool to automate the FTPA implementation. Researcher evaluates the performance of FTPA with parallel matrix multiplication and five kernels of NAS Parallel Benchmarks on a cluster system with 1,024 CPUs. The experimental results show that the performance of FTPA is better than the performance of the traditional check pointing approach due to fast recovery **[7]**.However this is only suitable for large problem. If the problem size is not large enough, not all processes will contribute to parallel recomputing.

B. Adaptiveness

A fault tolerance technique is expected to have the capability of dynamically adapting to distinct runtime conditions. The capability of dynamically adapting to distinct runtime conditions is an important issue. One of way by which a fault tolerant technique can be made dynamic is by an adaptive programming model **[8]**. This programming model is hybrid, composed by a synchronous part (where there are time bounds on processing speed and message delay) and an asynchronous part (where there is no time bound). There is further research scope to develop more adaptive programming model to make fault tolerance technique more adaptive to dynamic situation. In case of fault, the most important issue is efficient recovery in dynamic heterogeneous systems. Recovery under different numbers of processors is highly desirable. The fault tolerant and recover approaches must be suitable for applications with a need for adaptive or reactionary configuration control. Researcher proposed flexible rollback recovery in dynamic heterogeneous computing for such crucial requirements **[9]**. Still overhead of this technique is significant and need to be address further. Performance of any fault tolerant technique depends upon recovery time. Adaptive checkpointing is required to cope with volatile dynamic environment.

C. Multiple fault capability:

Multiple fault handling in distributed system is very crucial. In case of multiple faults, most of existing techniques have not enough provision or capability to handle multiple faults. Even, if enough capable to handle multiple faults than with low and unacceptable performance. Single point failure must be taken seriously while designing the multiple fault tolerance technique. Some multiple fault tolerant algorithms are based on optimistic or causal message logging approach. One approach to handle this critical issue suggested by researcher is use of uncoordinated checkpoint, distributed message logging and uses a reliable coordinator and checkpoint servers. In real situation, none of their existing implementation tolerates more than one fault. There is strong need for proper augmentation by appropriate mechanisms. Restart of full system in case of multiple faults is also a major disadvantage in terms of performance due to high recovery time. Thus optimistic or causal message logging required some mechanism to start from last checkpoint instead of restart. This problem can be overcome by using pessimistic message logging principle. In pessimistic message logging principle based algorithms, all in transit messages are stored on reliable media. In such a case recovery does not required restart Thus it reduces the recovery time on cost of large number of non computational reliable resources. Costs of such resources are very high

[10].Coordinate checkpointing can not handle multiple faults but uncoordinated with message logging can handle the multiple faults. In order to tolerate multiple faults using checkpointing and recovery, three critical functionalities that is necessary for fault tolerance: a light-weight failure detection mechanism, dynamic process management that includes process migration and a consistent checkpoint and recovery mechanism. Hugo Jung et al. proposed a technique to address this critical functionality [1].

D. Performance

But in case of uncoordinated checkpointing with message logging is based on piecewise information that need be making integrated information to recover the system. Most recent consistence state is constructed with this piecewise information. Storing of these piece wise information adversely affect the failure free performance, bandwidth and latency. The root cause of these is overheads for collecting this piece wise information in non failure environment and construction of most consistence state in multiple faults case. Coordinate checkpointing although incapable of handling multiple faults but overhead are low. Coordinate checkpointing are simple than uncoordinated checkpointing [9]. Run time must be low for any checkpointing based fault tolerance in both fault free and faulty case. Another overhead associate with sender based pessimistic logging is due to huge amount of messages. These huge amounts of messages must be kept in memory. These not only lower the performance but require larger amount of stable storage in general performance of checkpointing technique is very low. Researcher proposed replication based check-pointing to improve the performance [11].There are many issues related to replication based check pointing fault-tolerance technique. These issues are mainly degree of replication, check pointing storage type and location, check pointing frequency, check point size and check point run time. At the same time researcher suggested a adaptive check pointing and replication to dynamically adapt the check pointing frequency and the number of replicas as a reaction on changing system properties (number of active resources, resource failure frequency and system load)[12].

E. Automatic multiple fault detection and recovery

One of issue with multiple fault tolerance is automatic detection and recovery. Uncoordinated checkpointing with optimistic or causal message logging is combined to achieve automatic multiple fault tolerance in distributed system. A distributed system must tolerate n number of multiple faults but in reality none of system is available who can tolerate multiple tolerate automatically **[2]** A critical aspect for an automatic recovery is the availability of checkpoint files: If a resource becomes unavailable, it is very likely that the associated storage is also unreachable due to a network partition. A strategy to increase the availability of checkpoints is replication . Replication is number of copies of checkpointing. Migol is a framework for automatic recovery in grid application based on replication **[3].**

F. Memory requirement

In order to tolerate multiple faults, memory requirement should be low or medium but not large. Message logging stores all transit messages on reliable media. In that case require a large number of non computational reliable resources **[10]**.In order provide multiple fault capability this memory requirement because very huge. If the MPI implementation is to tolerate n concurrent faults (n being the number of MPI processes), then a reliable coordinator and a set of reliable remote checkpoint servers should be used.

Multiple fault capability can be increased using replication based checkpointing. N number of replicas can handle atleast N number of faults. But areplication protocol must be practical and simple. The protocol must provide rigorously-proven yet simply-stated consistency guarantee with a reasonable performance. Niobe is such protocol purposed by researcher **[13]**.Number of replicas must be sufficient. Large numbers of replicas will increase the cost of maintaining the consistency. Less number of replicas will affect the performance, scalability and multiple fault tolerance capability. Therefore, reasonable number replicas must be estimate as per system configuration and load. . Researcher proposed adaptive replicas creation algorithm **[14]**.There is further research scope to develop improved algorithm to maintain a rational replica number. Replica on demand is a feature that can be implemented to make more adaptive, flexible and dynamic. There is research scope to further improve protocols to achieve replication efficiently. There are some crucial requirements with replication protocol. These crucial requirements are support for a flexible number of replicas, strict consistency in the presence of network, disk, and machine failures and efficient common case read and write operations without requiring potentially expensive two or three-phase commit protocols.

G. Synchronization

Checkpointing must no rely on global synchronization because some nodes may leave or join the distributed system in dynamically manner.

H. Domio Effect and roll back propagation

This is an undesirable feature generally with uncoordinated checkpointing with message logging. Uncoordinated checkpointing is based on inter process communication, Message are logged independently and in order to get complete description of failed process these inter process dependencies may force some healthy process to roll back. This roll back of healthy process is called rollback propagation. This rollback may bring system at initial stage with loss of all computation. This situation is called domio effect **[9]**

I. Storage strategy

Performance of checkpointing technique depends upon storage strategy. Central dedicated servers and network storage are generally used a storage for checkpointing **[15] [16] [17]**. Although it is simple strategy to store the checkpoints but performance is low because all load of checkpointing is on central storage.

So overheads are very high. Further these central or network storage are single point failure. Scalability of central storage is also low. Performance can be improved if loads of checkpointing can be distributed evenly over all node involved in computation. John Paul Walter suggested Replication based checkpointing that distributed checkpointing over all computational loads **[11]**.Performance and scalability is improved on cost of consistency. Replication based checkpointing methods need more care attention related to consistency like degree of replica, consistency among replica, replica on demand etc. Consistency among replicas is a major issue. Multiple copies of same entity causes problem of consistency due to update of any copy by one of the user. A replication protocol must ensure the consistency among all replicas. **[18]**. The major issues with checkpointing storage are capacity, scalability, performance and overheads. Different storages which is used for checkpointing are parallel file system, centre storage, distributed storage area, network storage, disk etc. replica consistency usually requires deterministic replica behavior **[14]**.Researcher proposed an algorithm uses both active and passive strategies to implement optimistic replication protocol **[18]**. Researcher also proposed a simple protocol by combining the token with cache. This gives benefits of token as well as cache **[3].**There is still need of more simple, adaptive and practical replication protocol with adequate and sufficient ensured consistency [19].

J. Levels of checkpointing

There are mainly three level of checkpointing. These levels are kernel level, user level and application level. Kernel level works as kernel. It is easy but depends upon operating system. User level checkpointing acts as a library. Portability is high but on the cost of limited access to kernel specific attributed. In user level programmer put the location of checkpointing in programming. Researcher propose operational level checkpointing to lower the cost of recovery.

K. Scheduling of checkpointing:

Scheduling of checkpointing decide the overall performance of checkpointing. Improved scheduling approaches are suggested to reduce the various overheads like computational overhead, storage overheads and transfer overheads of checkpoints. Wang et al. moves checkpoint data to centralized storage in smaller groups. However, their results are limited to 16 nodes, making the scalability of their solution unknown[20]. Jung et al. show that the overhead of SAN-based checkpoint storage may be partially mitigated by first storing checkpoints locally before serializing their transfer to a SAN. To help improve the performance of checkpointing, particularly the checkpointing delay due to shared storage, they propose a group-based checkpointing solution that divides processes into multiple (smaller) checkpointing groups. Each group checkpoints individually in an effort to reduce the checkpointing overhead. In order to achieve low overhead, however, their solution requires that non checkpointing groups make computational progress during other groups' checkpoints. This requires that the processes be divided into groups according to their communication patterns, which in turn requires information from the user **[1]**.

L. Checkpointing overheads

Checkpointing overhead consist of coordination time, memory write time, and continue time. The coordination phase includes the time needed to quiesce the network channels/exchange bookmarks. The memory write time consists of the time needed to checkpoint the entire memory footprint of a single process and writes it to a local disk. Finally, the continue phase includes the time needed to synchronize and resume the computation. On occasion, particularly with large memory footprints, the continue phase can seem disproportionately long. This is due to some nodes' slower checkpoint/file writing performance, forcing the faster nodes to wait **[11]**. By appropriate scheduling in checkpointing in such a way based on its computational power these continue phase can be minimized. Thus, the time required to checkpoint the entire system is largely dependent on the time needed to write the memory footprint of the individual nodes.

M. Checkpointing Interval & frequency

Checkpointing time interval is the time elapsed between two successive checkpoints. Checkpointing frequency is number of checkpoint taken for a particular node for a given amount of time. Checkpointing frequency is reciprocal of checkpointing time interval, various overheads component of checkpointing also depends upon checkpointing frequency. As frequency is reduced overheads also reduces because it's take less time to write footprints to memory. Checkpointing size and frequency must be varying as per trend and potency of dynamism of distributed system.

IV. COMPARISON

In this section we compared coordinated and uncoordinated checkpointing on the basis of some important critical factors discussed in section V

TABLE1

COMPARISON TABLE

Issue	Coordinate checkpointing	Uncoordinated checkpointing
Consistency	More	less
Recovery tine	More	less
Performance	Low	High
Single Faults	Yes	Yes
Multiple Faults	No	Yes
Frequent Failures	No	Yes
Automatic	No	Yes
Domio Effect a Roll back propagation	No	Yes
Overhead	Less	More
Protocol	Simple	Complex
Scheduling	Less Complex	Complex

V. CONCLUSION

We have discussed many issues of checkpointing based recovery for multiple faults. These issues are discussed as an impact of overall performance of distributed system. For distributed system performance and efficiency are important. Some issues are not relevant with coordinate checkpointing but relevant with uncoordinated checkpointing. Better performance and efficiency can be achieved by improved storage strategy and lowering the overheads associate with checkpointing. Write time can be reduced by addressing the issue related the storage strategy. Likewise by scheduling the load of checkpointing equally and evenly over all computing nodes can improve the performance, reduces overheads and cost of checkpointing multiple fault tolerance capability with performance can be achieved by optimizing the replicas and producing the replica on demand. An adaptive replica may further reduce the consistency cost.

Various overhead like memory write time, coordination time, continue time are different with different checkpoint size and checkpointing frequency. These overheads need to optimize as a function of size of frequency. Scalability of checkpointing can be further improved by improved replicated checkpointing.

REFERENCES

[1] H. Jung, D. Shin, H. Kim, and Heon Y. Lee, "*Design and Implementation of Multiple FaultTolerant MPI over Myrinet (M3)* ," SC|05 Nov 1218,2005, Seattle, Washington, USA Copyright 2005 ACM

[2] M. Elnozahy, L. Alvisi, Y. M. Wang, and D. B. Johnson. *A survey of rollback-recovery protocols in message passing systems*. Technical Report CMU-CS-96-81, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA, October 1996

[3] X. China, "*Token-Based Sequential Consistency in Asynchronous Distributed System* ," 17 th Internaional Conference on Advanced Information Networking and Applications (AINA'03),March 27-29, ISBN: 0-7695-1906-7

[4] C. Coti, T. Herault, P. Lemarinier, L. Pilard, A. Rezmerita,E. Rodriguez, and F. Cappello, "*MPI Tools and Performance Studies—Blocking versus Non-Blocking Coordinated Checkpointing for Large-Scale Fault Tolerant MPI*," Proc. 18th Ann. Supercomputing Conf. (SC '06), pp. 127-140, 2006

[5] Emil Vassev, Que Thu Dung Nguyen and Heng Kuang, "*Fault-Tolerance through Message-logging and Check-pointing*," Technical Report Corcodia University.

[6] L. Alvisi and K. Marzullo. *Message logging : Pessimistic, optimistic, and causal. In Proceedings of the 15th International Conference on Distributed Computing,Systems* (ICDCS 1995), pages ,229–236. IEEE CS Press, May-June 1995.

[7] X. Yang, Y. Du, Panfeng W. Fu, and Jia "*FTPA: Supporting Fault-Tolerant Parallel Computing through Parallel Recomputing*," Ieee Transactions On Parallel And Distributed Systems, Vol. 20, No. 10, October 2009

[8] S. Gorender, and M Raynal, "*An Adaptive Programming Model for Fault-Tolerant Distributed Computing*" Ieee Transactions On Dependable And Secure Computing, Vol. 4, No. 1, January-March 2007.

[9] S. Jafar, A. Krings, and T. Gautier," *Flexible Rollback Recovery in Dynamic Heterogeneous Grid Computing*", IEEE Transactions On Dependable and Secure Computing, Vol. 6, No. 1, Jan-Mar 2009

[10] A. Bouteiller, F. Cappello, T. H Krawezik, Pi Lemarinier, F Magniette, "MPICH-V2: *a Fault Tolerant MPI for Volatile Nodes based on Pessimistic Sender Based Message Logging,* " SC'03, NoV 15-21, 2003, Phoenix, Arizona, USA Copyright 2003 ACM 1-58113-695-1/03/001...

[11] J. Walters and V. Chaudhary," *Replication-Based Fault Tolerance for MPI Applications*," Ieee Transactions On Parallel And Distributed Systems, Vol. 20, No. 7, July 2009

[12] M Chtepen, F.. Claeys, B. Dhoedt, P. Demeester, , and P. Vanrolleghem," *Adaptive Task Checkpointing and Replication:Toward Efficient Fault-Tolerant Grids*", IEE ransactions on Parallel and Distributed Systems, Vol. 20, No. 2, Feb 2009

[13] J Maccormick1, C Thekkath, M.Jager,K. Roomp, and L. Peterson , "*Niobe: A Practical Replication Protocol*." ACM Journal Name, Vol. V, No. N, Month 20YY.

[14] Cao Huaihu, Zhu Jianming, "*An Adaptive Replicas Creation Algorithm with Fault Tolerance in the Distributed Storage Network*" 2008 IEEE

[15] H.P. Reiser, M.J. Danel, and F.J. Hauck., " *A flexible replication framework for scalable andreliable .net services.*," In Proc. of the IADIS Int. Conf. on Applied Computing, volume1, pages 161–169, 2005

[16] Q. Gao, W. Yu, W. Huang, and D.K. Panda, "*Application-Transparent Checkpoint/Restart for MPI Programs over Infini-Band*," Proc. 35th Ann. Int'l Conf. Parallel Processing (ICPP '06), pp. 471-478, 2006.

[17] S. Sankaran, J.M. Squyres, B. Barrett, A. Lumsdaine, J. Duell, P. Hargrove, and E. Roman, "*The LAM/MPI Checkpoint/Restart Framework: System-Initiated Checkpointing*," Int'l J. High Performance Computing Applications, vol. 19, no. 4, pp. 479-493, 2005.

[18] A. Kale, U. Bharambe, "*Highly available fault tolerant distributed computing using reflection and replication*," Proceedings of the International Conference on Advances in Computing, Communication and Control ,Mumbai, India Pages: 251-256 ,: 2009 .

[19] Andr´e Luckow Bettina Schnor,"*Adaptive Checkpoint Replication for Supporting the Fault Tolerance of Applications in the Grid*," Seventh IEEE International Symposium on Network Computing and Applications, 978-0-7695-3192-2/08 $25.00 © 2008 IEEE.

[20] C. Wang, F. Mueller, C. Engelmann, and S.L. Scott, "*A Job Pause Service under LAM/MPIþBLCR for Transparent Fault Tolerance*,"Proc. 21st Int'l Parallel and Distributed Processing Symp.(IPDPS '07), pp. 116-125, 2007.