Low-cost Checkpointing-based Rollback Recovery Algorithm Considering Scalability

Jinho Ahn[1]

Dept. of Computer Science, Kyonggi University, Suwon-si, Gyeonggi-do, Korea
jhahn@kgu.ac.kr

Abstract

In this paper, we design a low-cost checkpointing-based rollback recovery algorithm to address the traditional scalability problem of synchronous checkpointing in the completely different point of view compared with existing ones. This algorithm enables a cluster-wide set of processes to take their semi-global checkpointing procedure while a small set of cluster heads monitor local commit of their respective administrative areas and always observe the global consistency condition. It can considerably lower communication overhead that may occur in the previous ones. This feature can enormously decrease the frequency of cluster-to-cluster communications especially in large-scale hierarchical multi-cluster systems.

Keywords: *Multi-cluster systems, Resiliency, Rollback recovery, Synchronous checkpointing*

1. Introduction

Checkpointing-based recovery is a well-known method to allow the current faulty state of a fail-stop processors system [13] to be rolled back to a globally consistent state of the system recorded on stable storage before its failure [9, 10, 12]. For this purpose, every process should periodically take its own checkpoint on stable storage with the information needed for tracking recoverability of the entire system. It classifies two checkpointing-based recovery algorithms, synchronous checkpointing and asynchronous checkpointing, whether checkpointing synchronization actions occur during normal operation or not [4]. As each process takes its local checkpoint periodically without any synchronization with other processes in asynchronous checkpointing, this approach allows the process the maximum autonomy and reduces checkpointing overhead during failure-free execution. However, if processes in the system fail, the approach may suffer from the domino effect, in which processes roll back recursively until the system has a globally consistent state. Therefore, it may complicate recovery and result in high recovery overhead. Additionally, it forces processes to take some useless checkpoints that will never be part of any globally consistent state. Moreover, the approach should maintain multiple checkpoints of each process in the stable storage and remove periodically the useless checkpoints among them using a special garbage collection mechanism. In contrast, synchronous checkpointing eliminating all of the disadvantages of the first approach can be very attractive to be used in inexpensive multi-cluster systems with a lot of commodity computers more vulnerable to process failures for executing numerous long-running applications [1, 5]. However, the only drawback of the synchronous checkpointing is the scalability constraint that may happen when the checkpointing initiator takes a global checkpoint with the others. The previous synchronous checkpointing algorithms have been attempting to solve this problem by selecting one of the

[1] Corresponding author: Tel.: +82 31 249-9674, FAX: +82 31 249-9673

two following methods. The first tries to force only a small set of related processes to take their respective checkpoints [3, 7, 8], reducing the number of checkpointing-induced messages and stable storage recording overhead. In the second method, some processes continue to aggressively progress their normal operations while others perform their checkpointing operations [6, 14]. A checkpointing algorithm [11] has attempted to gain all the advantages of both methods in the extreme, which can never exist [2]. Therefore, if someone intends to get more beneficial features of one between the two methods, he or she can take only fewer advantageous ones of the other. Assuming multi-cluster systems become more complex, hierarchical and large-scale, the two dimensional settlement may not be enough to alleviate the important overhead. In particular, its disappointed performance results may prevent the synchronous checkpointing approach from being used in real-world application fields [5].

This paper presents a low-cost checkpointing-based rollback recovery algorithm to address the traditional scalability problem of synchronous checkpointing in the completely different point of view compared with existing ones. This algorithm enables a cluster-wide set of processes to take their semi-global checkpointing procedure while a small set of cluster heads monitor local commit of their respective administrative areas and always observe the global consistency condition. It can considerably lower communication overhead that may occur in the previous ones. This feature can enormously decrease the frequency of cluster-to-cluster communications especially in large-scale hierarchical multi-cluster systems.

The rest of the paper is organized as follows. Section 2 describes our new synchronous checkpointing-based recovery algorithm. In sections 3 and 4, we compare related works and summary this paper in order.

2. New Synchronous Checkpointing-based Recovery Algorithm

2.1 Limitation of Traditional Approaches

They force checkpointing initiator to directly contact every other participant because they don't consider any physical or logical network topology while performing its checkpointing actions. Let us explain the checkpointing procedure in details as follows; first, the initiator sends every other process a checkpointing request message. Then, receiving the message, the latter takes its local tentative checkpoint on stable storage and sends its checkpointing reply message back to the initiator. If the participating process receives an application message from another process before getting the checkpointing request message from the initiator, the following validation process should be performed not in order to make the application message the orphan one like in figure 1; if the checkpointing sequence number(csn) included in the application message is greater than the process's csn, csn_p, p must take its tentative checkpoint before delivering the application message to its application. Afterwards, if the initiator obtains each a positive checkpointing reply message from all other processes, it sends them checkpointing confirmation messages. In this case, every process makes its tentative checkpoint permanent on stable storage. If any checkpointing participant gives a negative checkpointing reply message to the initiator, the latter forces every other process to remove its tentative checkpoint from the stable storage. Thus, this feature may result in significantly high latency traffic onto inter-cluster networks if the number of processes executing a large-scale multi-cluster based distributed application is greatly growing.

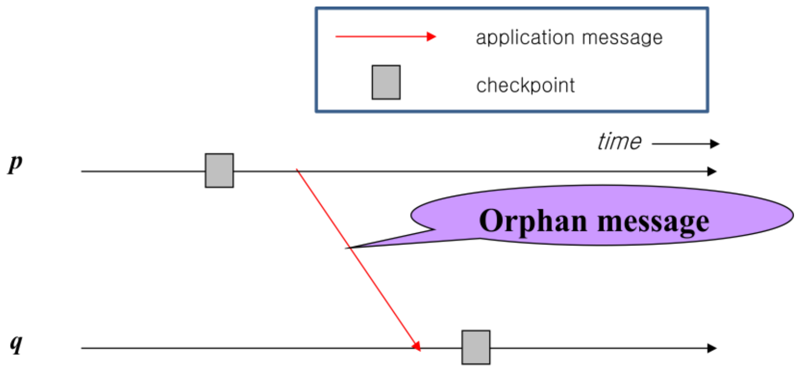

Figure 1. Illustration of Orphan Message Occurrence

2.2 Our Solution

A new topology-aware aggressive synchronous checkpointing algorithm is designed to solve this scalability problem of the traditional synchronous checkpointing algorithm by dividing a whole flat-style cluster network into a set of multiple clusters according to a particular criterion such as geographical topology, data dependency, inter-process communication frequency and so forth like in Figure 2. Like in Figure 3, an area consists of a small set of processes and has a cluster head, which is elected among the processes based on availability of their computing and networking resources or importance of their location. In our algorithm, the cluster head for each area should fill the following two roles, cluster area and virtual head area checkpointing orchestrations to significantly enhance scalability of synchronous checkpointing.

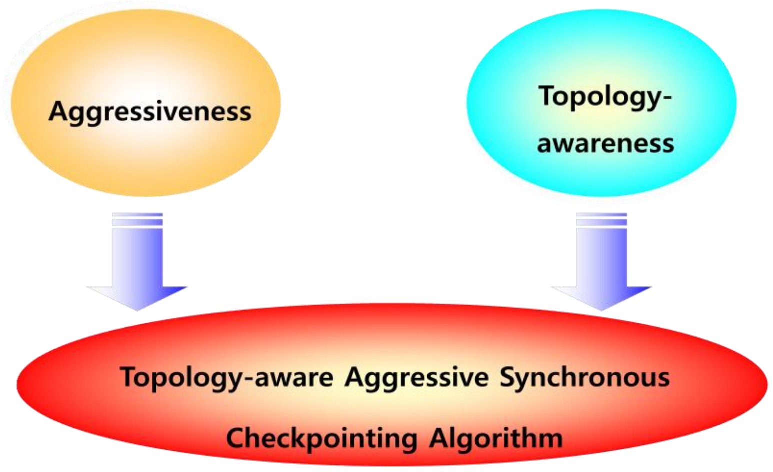
Aggressiveness
Topology-awareness
Topology-aware Aggressive Synchronous Checkpointing Algorithm

Figure 2. Main Design Rationale of our Checkpointing Algorithm

In other words, when the checkpointing initiator in figure 3 attempts to send out its checkpointing control messages, it communicates only with cluster heads, not every other participating process. This behavior may tremendously reduce the inter-network traffic unlike the traditional ones. Moreover, each cluster head pretends to be the checkpointing initiator to its local processes by executing the checkpointing orchestration. For example, the global checkpointing initiator sends inter-cluster checkpointing request messages to each a cluster head for area A. Then, the cluster head forwards the message to its local members inside this area. Similarly, the initiator and cluster head for area B also transmits the message to its local members. Receiving the message, each local member tries to take its local checkpoint. Depending on its checkpointing result, it transmits a checkpointing reply message with the result to its cluster head. In this case, the cluster head informs the initiator of its area level decision on checkpointing. If the initiator gets each a positive checkpointing reply message from all the cluster heads, it gives them back global checkpointing confirmation messages for having each participating process change its tentative checkpoint to the permanent one. When receiving this message, each cluster head forwards it to its local processes.

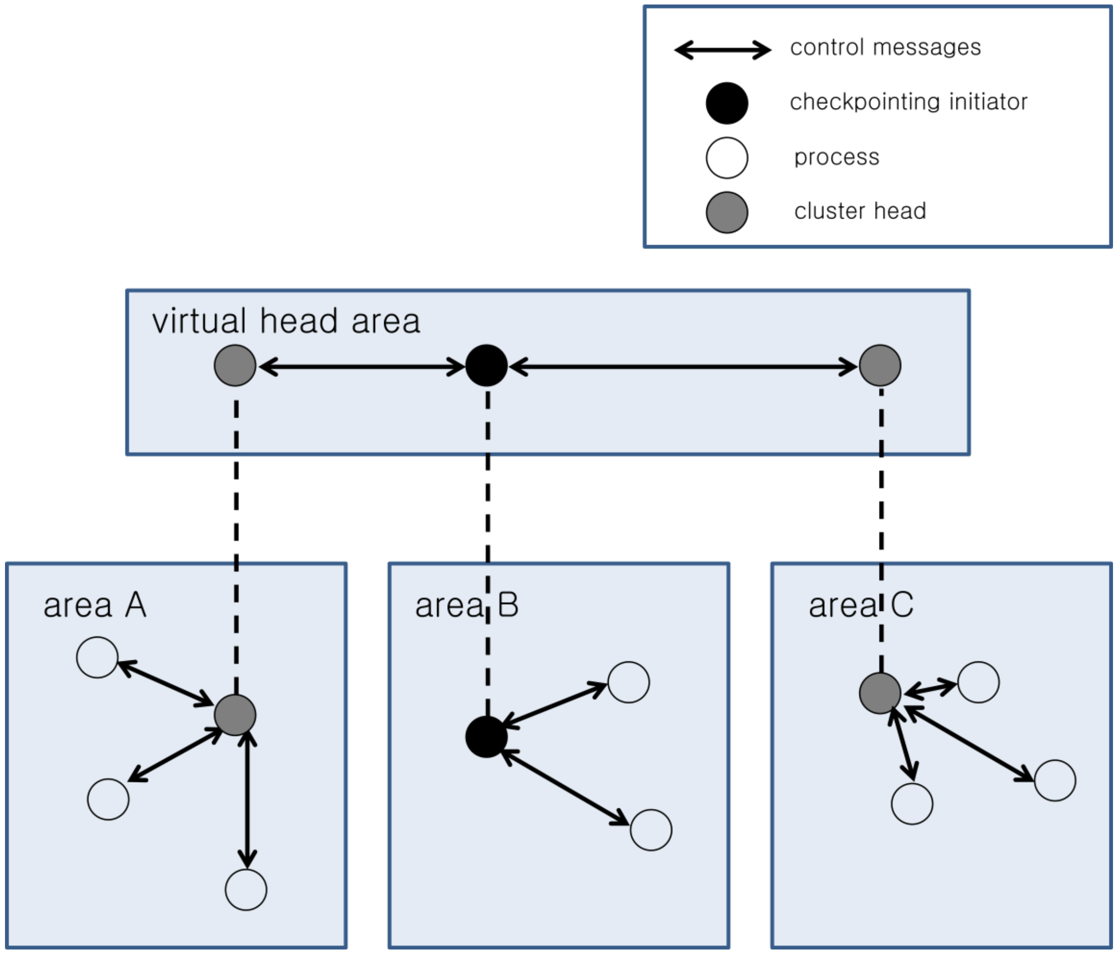

Figure 3. An Execution of our Algorithm in a Multi-cluster System

3. Discussion

Most of traditional synchronous checkpointing algorithms attempt to reduce their coordination overhead by using both two optimization methods or one of both; minimizing the number of participating processes and having the checkpointing process and the normal computation execute in parallel. Koo and Toueg in [8] presented an efficient checkpointing algorithm which forces only the minimum number of processes to take their local checkpoints. This behavior may reduce the number of checkpoints and checkpointing control messages. But, this algorithm should give up the entire checkpointing procedure when any participant fails to take a local checkpoint. In order to address this drawback, Kim and Park's min-process algorithm [7] was proposed to enable a part of participating processes to complete their checkpointing actions even if the others fail. However, all the algorithms mentioned earlier make every participating process's normal computation blocked for not a short time, which may considerably decrease performance of the entire system, in particular, a large sensor network-based system. Moreover, if exchanging messages are cascaded linked with each other or follow data parallel pattern, they may not all actually reduce the number of checkpointing processes and even generate much more synchronization messages [5].

To avoid the disadvantage of the min-process approach, a non-block checkpointing algorithm [6] was designed to be able to progress each process's normal computation while executing its synchronous checkpointing procedure. But, the algorithm requires all processes in the system to participate in its checkpointing. Silva and Silva in [14] introduced a similar algorithm with a difference that some processes not exchanging messages with the others after taking their latest checkpoints can be exempted from their new checkpointing requirement.

The first algorithm [11] was proposed to attempt to require only a minimum number of processes to take checkpoints with the same feature of the non-blocking ones. But, the algorithm may incur inconsistency problems in some checkpointing and communication patterns [2]. Cao and Singhal in [1] presented some hybrid algorithms to keep one of both conditions to a maximum while losing the other at minimum. However, the algorithms are very complex and may have some drawbacks of both min-process and non-blocking checkpointing approaches.

All the above synchronous checkpointing algorithms haven't even consider any physical or logical network topology, which may raise serious core network traffic problems. Furthermore, as the current sensor network-based distributed systems based on many connected networks scale up tremendously, their entire performance may be getting increasingly worse due to the inefficiency of the algorithms.

4. Conclusion

In this paper, we proposed a topology-aware synchronous checkpointing-based recovery algorithm to address the traditional scalability problem of synchronous checkpointing in the completely different point of view compared with existing ones. Each cluster head monitors the checkpointing process executing inside its administrative area mostly on an edge network, reducing checkpointing initiator’s load of the global synchronization activities greatly. Therefore, the initiator has only to control global checkpointing actions with cluster heads, significantly decreasing high latency traffic generated across clusters. We believe that this algorithm can be used as an efficient reliability enhancing method for large-scale hierarchical multi-cluster systems.

References

[1] G. Cao and M. Singhal, “Checkpointing with Mutable Checkpoints”, Theoretical Computer Science, 290 **(2003)**.

[2] G. Cao and M. Singhal, “On coordinated checkpointing in distributed systems”, IEEE Transactions on Parallel Distributed System, 9, 12 **(1998)**.

[3] Y. Deng and E. K. Park, "Checkpointing and rollback-recovery algorithms in distributed systems", Journal of Systems and Software, 4 **(1994)**.

[4] E. N. Elnozahy, L. Alvisi, Y. M. Wang and D. B. Johnson, "A Survey of Rollback-Recovery Protocols in Message-Passing Systems", ACM Computing Surveys, 34, 3 **(2002)**.

[5] E. N. Elnozahy and J. Plank, "Checkpointing for Peta-Scale Systems: A Look into the Future of Practical Rollback-Recovery", IEEE Transactions on Dependable and Secure Computing, 1, 2 **(2004)**.

[6] E. N. Elnozahy, D. B. Johnson and W. Zwaenepoel, "The performance of consistent checkpointing", Proc. 11th Symp. on Reliable Distributed Systems, **(1992)**.

[7] J. L. Kim and T. Park, "An Efficient protocol for checkpointing recovery in distributed systems", IEEE Trans. Parallel Distributed Systems, 5, 8 **(1993)**.

[8] R. Koo and S. Toueg, "Checkpointing and rollback-recovery for distributed systems", IEEE Transactions on Software Engineering, 13, 1 **(1987)**.

[9] H. F. Li, Z. Wei and D. Goswami, "Quasi-atomic recovery for distributed agents", Parallel Computing, 32 **(2006)**.

[10] Y. Luo and D. Manivannan, "FINE: A Fully Informed aNd Efficient communication-induced checkpointing protocol for distributed systems", J. Parallel Distrib. Comput., 69 **(2009)**.

[11] R. Prakash and M. Singhal, "Low-cost checkpointing and failure recovery in mobile computing systems", IEEE Trans. Parallel Distributed System, 7, 10 **(1996)**.

[12] J. T. Rough and A. M. Goscinski, "The development of an efficient checkpointing facility exploiting operating systems services of the GENESIS cluster operating system", Future Generation Computer Systems, 20, 4 **(2004)**.

[13] R. D. Schlichting and F. B. Schneider, "Fail-stop processors: an approach to designing fault-tolerant distributed computing systems", ACM Transactions on Computer Systems, 1 **(1985)**.

[14] L. M. Silva and J.G. Silva, "Global checkpointing for distributed programs", Proc. 11th Symp. on Reliable Distributed Systems, **(1992)**.

Authors

Jinho Ahn

He received his B.S., M.S. and Ph.D. degrees in Computer Science and Engineering from Korea University, Korea, in 1997, 1999 and 2003, respectively. He has been an associate professor in Department of Computer Science, Kyonggi University since 2003. He has published more than 70 papers in refereed journals and conference proceedings and served as program or organizing committee member or session chair in several domestic/international conferences and editor-in-chief of journal of Korean Institute of Information Technology and editorial board member of journal of Korean Society for Internet Information. His research interests include distributed computing, fault-tolerance, sensor networks and mobile agent systems.