A New Recovery Scheme for the Long-Term Mobile Application

Sungchae Lim

Dongduk Women's University
sclim@dongduk.ac.kr

Abstract

Recently, the widespread use of mobile devices, such as smartphones and tablet PCs, leads to a new demand for distributed mobile applications. When the mobile application has a long lifetime and it is comprised of many parallel tasks, it is required to safely checkpoint its processing states against abrupt failure. To this end, lots of works have been done to reduce the wireless traffic overhead caused by distributed checkpointing protocols. In this paper, we also propose a new recovery scheme with less networking overhead and high flexibility in its protocol. For this, we deploy logging agents across mobile support stations so that they can gather the causality dependency vectors of the involved application processes without the use of wireless data transmission. Because of these features, our scheme provides the benefits of high flexibility during the checkpointing time and low traffic overhead, while preventing severe cascaded rollbacks efficiently.

Keywords: *recovery, domino-effect, causality dependency, cascaded rollback*

1. Introduction

Recently, we have seen much technological progress in the both fields of wireless networks and mobile devices. Such progress results in a new demand for the distributed mobile application that can execute on multiple mobile devices and performs its tasks by communicating through wireless data links [1, 2, 3]. Originally, the distributed application was assumed to execute on a number of fixed-networked computing servers, and its recovery schemes have been also based on the physical features of the fixed-network environment [4, 5, 6, 7]. That is, the schemes suppose that participant application processes can use sufficient computing powers and pay rather cheap costs for data transit between different processes [7, 8]. Although those schemes are eligible for the traditional fixed-network environment, they cannot be used without significant modifications, if applied to the mobile computing environment [8, 9]. Since the mobile device has a battery limitation and data transit within wireless networks is much costly, compared with that in fixed networks, it is required to redesign the checkpointing scheme suitably for the mobile application. For this reason, a group of new recovery algorithms for the mobile distributed application are proposed to overcome difference between those two network environments [2, 10, 11, 12, 13].

In the paper, we also propose a checkpointing scheme used to save log records during the execution of a distributed mobile application. In particular, our scheme focuses on how to efficiently checkpoint a distributed mobile application with a long lifetime. Here, we suppose that the long-term mobile application usually has intermittent critical points during its long execution time and a particular group of participant mobile processes attempt to save their checkpoints more frequently than others. In this situation, if every checkpointing request from a mobile process freezes the whole processing of the distributed application, then use of checkpointing could not be acceptable for long-term mobile applications. If we remove such freezing periods through loose synchronization between mobile processes, by contrast, the

cost for the recovery phase in the future time is likely to grow due to undesirable cascaded rollbacks and prolonged time to scrutinize distributed checkpoint records [8, 9, 14].

To solve the technical dilemma above, we employ the special-purpose software agent that is aimed at reducing the amount of wireless data transit as well as the occurrences of execution halting during the synchronization phase of checkpointing time. The logging agent is a process running on the mobile support station, and it is responsible for actively storing checkpoint records and accepting checkpointing request from its associated mobile process [11]. To reduce wireless data transit, the logging agent exchanges causality dependency vectors with others by using fixed-network data links alone.

Through those checkpointing-related data, the logging agent always monitors which collections of checkpoints can make the global consistent state of the monitored mobile application. If a high possibility of severe rollbacks is estimated from the monitoring, the logging agent can initiate the actions of enforced checkpointing to create a global consistent state on behalf of its mobile processes. Correspondingly, when any mobile process requests checkpointing, its logging agent may do nothing for that request, if a global consistent state can be formed from only the recent checkpoints. This means that the undesirable execution freezing of the mobile application can be avoided in many cases. Sine a rollback limit is also guaranteed and checkpointing-related data are shared among logging agents, our scheme can have high flexibility in the checkpointing actions. Due to such features, our scheme can significantly reduce network and CPU overhead for checkpointing, while preventing the cascaded rollback from nullifying the recent checkpoints beyond a given limit.

The rest of this paper is organized as follows. In Section 2, we present some technical backgrounds regarding the necessity of distributed checkpointing and related earlier schemes. Then, we propose a new efficient checkpointing scheme in Section 3, and conclude this paper in Section 4.

2. Preliminaries

2.1. Global Consistent State

Since there is no global clock able to give a total order to events arising on more than one application process (AP), Lamport [15] partially ordered distributed application's events based on the relation of "happen-before". Here, we say that any event *e1* is a causal event of *e2*, if *e1* "happen-before" *e2*. Since the "happen-before" relation has a transitive property, we can decide on the existence of a causality relation between two different events within a distributed application.

From this, Lamport declares that an execution state *G* is globally consistent, if there is no event $e1 \in G$ such that *e2* is a causal event of *e1* and $e2 \notin G$. In this way, the GCS (Global Consistent State) of a distributed application was formally defined [15].

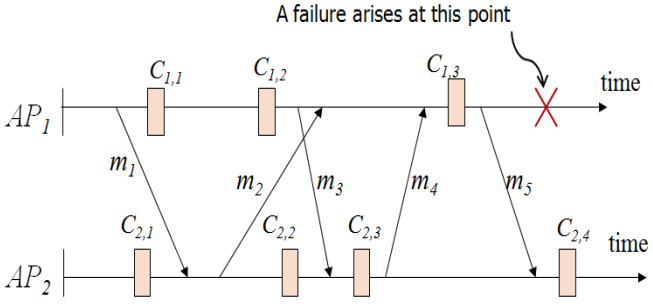

Figure 1. An Example of Failure on a Distribute Application

Figure 1 shows a distributed application scenario where two different APs have joined in the application and an application failure occurs in the AP *AP1*. Ahead of the failure time, the two APs have stored their local checkpoint records and their records are represented by rectangles in the figure. In this scenario, message *m5* has a causality to the checkpoint record $C_{2,4}$, and *m5* becomes to be lost since there is no record saving *m5* in *AP1*. As a result, record $C_{2,4}$ has to be eliminated from a GCS since its causality event (i.e., the send event of *m5*) was not checkpointed. For the same reason, $C_{1,3}$ is rolled back because of cancelation of *m4*, and the record $C_{2,3}$ should be deleted since it has causality from *m3*. Consequently, we obtain the latest GCS that consists of $C_{1,2}$ and $C_{2,2}$. In the recovery phase, this disrupted application will be resumed by restoring the processing state saved in $C_{1,2}$ and $C_{2,2}$. Here, we say that there exist cascaded rollbacks of checkpoints $C_{2,4}$, $C_{1,3}$, *and* $C_{2,3}$ because of **domino-effect**. query.

2.2 Previous Schemes

In mobile computing environment, each wireless cell is managed by a single mobile support station (MSS) and the mobile host sends its data thorough any wireless link made in the wireless cell [8, 11]. The MSS is also responsible for supporting seamless hand-offs and message routing between mobile hosts. For this, MSSs are interconnected with high-speed fixed networks. In this environment, the cost for wireless data transmits is much higher than that in fixed networks. For this reason, the previous checkpointing schemes place their performance focus on reducing data traffic over wireless links [6, 7, 8, 9].

Those previous schemes can be categorized into the synchronized schemes and the asynchronized ones. In the synchronized scheme, whenever a mobile AP asks for saving its current execution state, this triggers a new creation of a global checkpoint. For the global checkpointing, subsequently, other co-worker APs have to be frozen for synchronization. Besides such an undesirable freezing period, global checkpointing tends to incur a large number of message transmits over wireless data links. Consequently, the synchronized scheme is apt to suffer from a very expensive communication overhead as well as long freezing times for global checkpointing [7, 9, 10, 13].

Meanwhile, the asynchronized scheme does not advance application's GSS point according to every request of checkpointing from APs. Instead, during the recovery phase, the latest GCS is calculated by scrutinizing local checkpoint records scattered across MSSs. Since there is no frequent checkpointing enforced by other APs, this scheme can evade the drawbacks found in the synchronized scheme. Moreover, when it comes to the wireless network environment, the asynchronized schemes are much more feasible because of less use of wireless traffic [12, 13]. However, this scheme has a problem of high chances of the cascaded checkpoint rollback in the recovery phase. In a worst case, the whole intermediate processing states, saved in local checkpoint records, can be cancelled. To remove such domino effect in asynchronized scheme, the **message-induced** checkpointing scheme [9, 10, 12] was proposed.

In this message-induced scheme, the mobile AP toggles its state between SEND and RECEIVE states according to the events of its message send and receive, respectively. That is, if a mobile AP sends a message after an event of message receiving, then it changes its state to SEND from RECEIVE. Then, it remains in the SEND state unless it receives any message

from other APs later. Correspondingly, when the AP with a SEND state receives a message from other AP, then it makes a local checkpoint for saving its own processing state before it switches to the RECIVE state. As a result, since there is always a local checkpoint between two consecutive events of message receive and message send, the message-induced scheme can easily remove the chance of cascade rollbacks of local checkpoints [9, 10, 12]. Of cause, after changing its state to RECEIVE, the AP remains in the same state unless it sends a new message outwards.

3. Semi-synchronized Checkpointing

3.1 Idea Sketch

Although the message-induced scheme can considerably reduce the occurrence of cascaded rollbacks during the recovery phase, it has a problem in that checkpointing is totally dependent on the events of message arrivals. Therefore, the logging policy of the message-induced scheme is apt to generate obsolete local checkpoints because of lack of context awareness. For example, if a message saying “hello” induces a local checkpoint, that checkpoint is unlikely to save any meaningful processing state. Moreover, lack of considerations on application’s context can be more problematic in the long-term application. While a long-living distributed application is being in process, some periods in its lifetime have greater importance than other periods. Suppose that an AP has accepted valuable input data from its mobile host and advanced its application state just now. If the input data is one that comes from a laborious user interaction, the AP may want to include this period of time into a GCS. However, the message-induced scheme cannot reflect such application’s context.

Against those problems, we make efforts to devise a semi-synchronized scheme such that mobile APs can determine their own checkpointing times on the consideration of application’s context. Additionally, global checkpoints can be made with more flexibility rather than in previous schemes. For this, our scheme employs the logging agent running on the MSS.

Figure 2 shows the assumed system architecture where the loggings agent exists between two different APs. The figure depicts a situation where a mobile AP (say P_a) in cell A sends a message to the counterpart mobile AP (say P_b) in cell B for exchanging application’s context. The message from P_a is first put into the outbound queue of MSS A, and then the logging agent in MSS A dequeues the message for parsing and modification. To decrease the amount of wireless traffic, checkpoint-related data, which include the causality vector and serial numbers of local checkpoints made in all APs, are delivered only in fixed networks. For this, the checkpoint-related data are inserted into the application message by the logging agent in MSS A. Correspondingly, the counterpart logging agent in MSS B detaches the additional data before it puts the original message into its inbound queue. While modifying the message, the logging agents in MSS A and B update their checkpoint records within main memory.

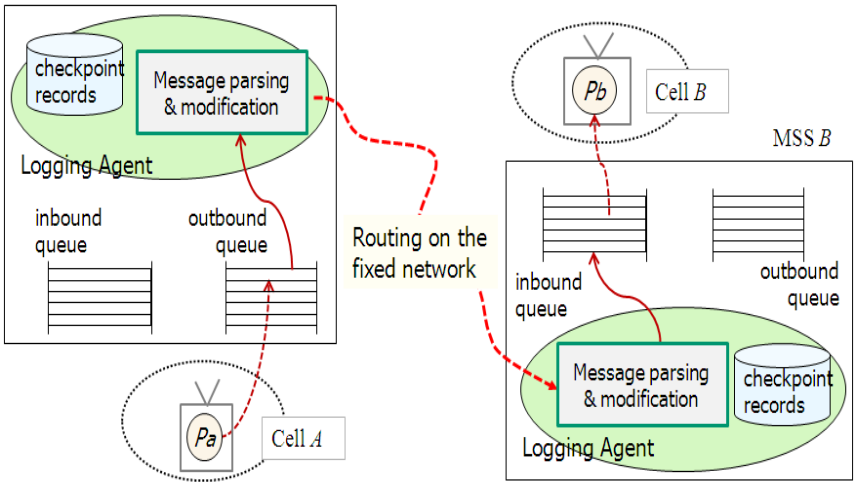

Figure 2. Architecture of the Assumed Wireless Network Environment

To see our idea, let us assume that a mobile AP attempts to make a local checkpoint *L* for saving its current processing state on its own decision. In this situation, there exist two different approaches possible. As one approach, we can create a global checkpoint including *L*. This is just the mechanism of the synchronized scheme, which results in heavy overhead for global synchronization. The other remaining approach is to create just a local checkpoint without synchronization with other APs outsides. Although synchronization overhead decreases in this case, the persistence of checkpoint *L* is not guaranteed in the presence of failure on the mobile application.

To lie in between these two extremes, our scheme employs the notion of the cascaded rollback limit (CRL) and sets an adequate level of CRL to each mobile application. That is, if it is set to a value of *d*, then our scheme makes sure that none of mobile APs loses *d* number of recent local checkpoints in face of worst-case domino-effect. With a CRL level of one, therefore, our scheme will work identically with the previous synchronized scheme; if the value goes big, otherwise, our scheme runs similarly as the asynchronized scheme. In other words, our scheme can adjust the synchronization strictness of checkpointing by picking a proper CRL level. The level of CRL has to be determined by considering application's tolerance to feature and it is set to a value at application's startup time.

For this adjustable synchronization of global checkpoints, the logging agent continuously gathers the checkpoint-related data and calculates possibility of GCS constructions based on the current local checkpoints. Since the agent always transits such data only in fixed networks, our scheme provides low overhead for data communions. Detail about that is addressed in the next section.

3.2 Proposed Algorithm

We first describe the format of the application message to be transferred on the wireless data links. The followings are the fields of the message. At the startup time of a distributed application, a unique *Id* is issued and saved in the field ID_app.

- ID_app: *Id* of the distributed mobile application.
- ID_sender : *Id* of the sender AP of this message.
- ID_receiver: *Id* of the counterpart AP to receive this message.

- Type_m: Flag expressing the type of this message (application data/request for a checkpoint).
- Data: Application data or data for checkpoint creation.

Each logging agent manipulates its own checkpoint record in the main memory temporarily. This memory-resident checkpoint record is continuously updated whenever a new message arrives as in Fig. 3. The memory-resident record is written to the disk at the time of checkpointing later. The fields of the record are as follows.

- ID_app : *Id* of the distributed mobile application.
- Val_crl : CRL level of application ID_app.
- ID_ap[1,..,N]: *Id*s of participant APs in application ID_app.
- Idx_ap : Index of the owner AP.of this record
- Ptr_chpt: Disk address to the previous checkpoint record.
- Data: Area for saving APs processing state and the messages sent to counterpart APs.
- Serial[1,..,N]: Serials of the latest local checkpoints made by all the participant APs.
- Vector[1,..,N] : Causality dependency vector.

The field Vector[] is for saving the causality dependency vector and Serial[*j*] saves the serial number of the latest local checkpoint made by the AP ID_ap[*j*]. The usage of those fields for calculating a GCS is referred to other literature [4, 5, 7].

Figure 3 depicts how the logging agent works when it receives message *M* from its local AP with *Id* of ID_ap[Idx_ap]. If message *M* is a message to request checkpointing, which is notified by the value in Type_m, then the lines 4-11 will be executed; otherwise, if *M* is for sending application message to other AP, line 13 is performed. In Fig. 3, the notation *R* represents a memory-resident checkpoint record manipulated by the logging agent. In line 7 of Fig. 4, the logging agent checks if there is a possibility that domino-effect may nullify local checkpoints beyond the level of CRL. If it is possible from the worst-case domino-effect, a new GCS is made in line 8. Otherwise, it is routed to the counterpart logging agent after the modification of *M*.

```
EVENT: a new message M enters the outbound queue in a MSS.
1.  begin
2.  Dequeue message M and check the pair of (M.ID_app, M.ID_sender) to indentify the
    checkpoint record R involved with M.
3.  if ( M.Type_m is for the request of checkpointing ) then
4.      Copy M to R.Data and write R into the log disk.
5.      Initiate R for the next local checkpointing.
6.      Calculate the latest CGS; let K be the largest serial number of M.ID_sender's local
    checkpoint included in that CGS.
7.      if (K < R.Serial[R.Idx_ap] - R.Val_mdr) /* possibility of long cascaded rollbacks */
8.          Call the routine AdvanceGCS(R ). /* GCS advancing */
9.      endif
10.     To broadcast the creation of a new local checkpoint, send messages to all the logging
    agents involved with the application of R.ID_app[].
11.     Send an acknowledgment message to the AP of M.ID_sender.
12. else  /* M is for sending application data */
13.     Append the checkpoint-related fields of (R.Vector[], R.Serila[]) to M ; then send the
    modified M toward the logging agent of AP M.ID_receiver.
14. endif
15. end.
```

Figure 3. Algorithm for Processing a Message Arriving at the Outbound Queue

Meanwhile, Figure 4 shows the way the logging agent works at the arrival event of a message coming from the counterpart logging agent. If the message *M* is for the delivery of application data, it is forwarded to AP *M*.ID_receiver after checkpoint-related data in *M*. If *M* is a broadcasting message saying a new local checkpointing in AP *M*.ID_sender, the logging agent just updates its checkpoint record in line 7. Otherwise, if enforced checkpointing is requested in message *M*, interactions with its local AP are performed as in line 9. While the logging agent is executing the algorithms in Figs. 3 and 4, the routine *AdvanceGCS()* plays a critical role in global synchronization. Its algorithm can be referred to [2].

```
EVENT: a new message M is routed to the logging agent LA in an MSS.
1.  begin
2.  Check M.ID_app and M.ID_receiver both and locate the memory-resident checkpoint record
    R involved with M.
3.  if (M.Type_m is for sending application data)
4.      Remove the checkpoint-related fields from the message; then put it into the inbound
    queue.
5.      Update the checkpoint record R by using checkpoint-related data in M.
6.  else if (M.Type_m is for broadcasting a new local checkpointing )
7.      Update the checkpoint record R by using data in M.
8.  else  /*  request for an enforced local checkpoint */
9.      Perform the steps for the enforced checkpointing of the AP of M.ID_receiver.
10. endif
11. end.
```

Figure 4. Algorithm for Processing a Message from the Counterpart Logging Agent

4. Conclusion

As two key performance metrics of a checkpointing scheme, we consider less overhead paid for making checkpoint records during the normal execution time and a low possibility of domino-effect in the recovery phase. Because there is a trade-off between these metrics, it is hard to develop any checkpointing scheme that has a good performance in both aspects. Therefore, it is likely that the earlier message-induced scheme provides a rather good balance on them. However, when it comes to application context awareness in the checkpointing time, the message-induced scheme has a very poor property because its mobile application process cannot create checkpoint records on its own decision.

To solve those difficulties in earlier schemes, we have proposed a semi-synchronized checkpointing protocol that ensures the durability of local checkpoints within a specific rollback limit. The proposed protocol makes it possible for the mobile AP to create its local checkpoints in accordance with its application context. When the rollback limit is set to *K*, then the worst-case number of checkpoint rollbacks is bounded by *N*(K-1)*, for every distributed application composed of *N* APs. Since the actual number of rollbacks is much less than the worst-case number in reality, our scheme can prevent the undesirable domino-effect. In addition, since the proposed protocol is executed by logging agents, networking overhead is very low.

References

[1] S. Gadiraju and V. Kumar, "Recovery in the Mobile Wireless Environment Using Mobile Agents", IEEE Trans. on Mobile Computing, vol. 3, no. 2 **(2004)** April, pp. 180-191.
[2] S. Lim, "A Tunable Checkpointing Algorithm for Distributed Mobile Applications", International Journal of Computer Science Issues, vol. 8, no. 6, **(2011)**.
[3] P. Singh and G. Cabillic, "A Checkpointing Algorithm for Mobile Computing Environment", Personal Wireless Communications (LNCS), vol. 2775, **(2003)**, pp. 65-74.
[4] D. K. Pradhan and N. H. Vaidya, "Roll-Forward and Rollback Recovery: Performance-Reliability Trade-Off", IEEE Trans. Computers, vol. 46, no. 3, **(1997)**.
[5] D. Manivannan and M. Singhal, "Quasi-Synchronous Checkpointing: Models, Characterization, and Classification", IEEE Trans. on Parallel and Distributed Systems, vol. 10, no. 7, **(1999)**.
[6] L. K. Awasthi and P. Kumar, "A Synchronous Checkpointing Protocol for Mobile Distributed Systems: Probabilistic Approach", International Journal of Information and Computer Security, vol. 1, no. 3, **(2007)**.
[7] Y. M. Wang, "Consistent Global Checkpoints That Contain a Given Set of Local Checkpoints", IEEE Trans. on Computers, vol. 46, no. 4, **(1997)**.
[8] S. Lim, "A New Distributed Checkpointing Scheme for the Mobile Computing Environment", Proceedings of the IEEE-RIVF, **(2009)** July 13-17, Danang, Vietnam.
[9] S. Gadiraju and V. Kumar, "Recovery in the Ricardo Baratto, Shaya Potter, Gong Su, and Jason Nieh, MobiDesk: Mobile Virtual Desktop Computing", Proceedings of the 10th International Conference on Mobile Computing and Networking, **(2004)** September 26-30, Philadelphia, USA.
[10] C.-M. Lin and C.-R. Dow, "Efficient Checkpoint-based Failure Recovery Techniques in Mobile Computing Systems", Journal of Information Science and Engineering, vol. 17, no. 4, **(2001)**.
[11] T. Imielinski and B. R. Badrinath, "Mobile Wireless Computing: Challenges in Data Management", Communications of the ACM, vol. 37, no. 10, **(1994)**.
[12] T. Tantikul and D. Manivannan, "Communication-Induced Checkpointing and Asynchronous Recovery Protocol for Mobile Computing Systems", Proceedings of the 6th International Conference on PDCAT, **(2005)** December 5-8, Dalian, China.
[13] L. Alvisi, E. N. Elnozahy, S. Rao, S. A. Husain and A. De Mel, "An Analysis of Communication Induced Checkpointing", Proceedings of the 29th Symposium on Fault-Tolerant Computing, **(1999)** June 15-18, Wisconsin, USA.
[14] F. Zambonelli, "On the Effectiveness of Distributed Checkpoint Algorithms for Domino-Free Recovery", Proceedings of the 7th Symposium on High Performance Distributed Computing, **(1998)** July 31-31, Chicago, USA.
[15] Lapmport, "Time, Clocks, and the Ordering of Events in a Distributed System", Communications of ACM, vol. 21, no. 7, **(1978)**.

Author

Sungchae Lim

He received the B.S. degree in Computer Engineering from Seoul National University at 1992, and achieved the M.S. and Ph.D. degrees in Computer Science from KAIST, at 1994 and 2003, respectively. He is currently an Associate Professor in the Department of Computer Science at Dongduk Women's University.