On Reducing Rollback Propagation Effect of Optimistic Message Logging for Group-Based Distributed Systems

Jinho AHN\(^{(a)}\), Member

SUMMARY This paper presents a new scalable method to considerably reduce the rollback propagation effect of the conventional optimistic message logging by utilizing positive features of reliable FIFO group communication links. To satisfy this goal, the proposed method forces group members to replicate different receive sequence numbers (RSNs), which they assigned for each identical message to their group respectively, into their volatile memories. As the degree of redundancy of RSNs increases, the possibility of local recovery for each crashed process may significantly be higher. Experimental results show that our method can outperform the previous one in terms of the rollback distance of non-faulty processes with a little normal time overhead.

key words: distributed computing, scalability, fault-tolerance, group communication, message logging and recovery

1. Introduction

Optimistic message logging is very attractive for providing fault-tolerance with low failure-free overhead for large-scale distributed systems [3]. However, it may suffer from cascading rollback due to its message log volatility. As the existing protocols [2], [4] assume reliable FIFO point-to-point communication links, it is very difficult to alleviate their inherent drawback while preserving their advantage. But, we have identified that if reliable FIFO group-based communication links, becoming a mandatory building block for this type of computing, are assumed, this limitation may be lifted to a certain extent by using each same message sent to a group of processes. In this paper, we present a new method to considerably reduce its rollback propagation effect by utilizing positive features of reliable FIFO group communication links. To satisfy this goal, the proposed method forces group members to replicate different receive sequence numbers (RSNs), which they assigned for each identical message to their group respectively, into their volatile memories. As the degree of redundancy of RSNs increases, the possibility of local recovery for each crashed process may significantly be higher while increasing the failure-free overhead resulting from log information synchronization. Therefore, the degree can adaptively be configured according to the requirements on availability of applied systems.

2. The Proposed Method

For a better understanding about the common problem of the traditional optimistic message logging protocols occurring when applying them in reliable FIFO group communication based distributed systems, let us see an example of message sending to a process group in Fig. 1. Suppose that there are three processes, \(P_{g_1}^1\), \(P_{g_1}^2\) and \(P_{g_1}^3\), in group \(g_1\). When three different senders, \(S_1\), \(S_2\) and \(S_3\), have sent three messages, \(m_{g_1}^1\), \(m_{g_1}^2\) and \(m_{g_1}^3\), to group \(g_1\) respectively, the messages are not delivered to \(P_{g_1}^1\), \(P_{g_1}^2\) and \(P_{g_1}^3\) in a same delivery order. Thus, RSNs of each message assigned by the processes are not identical to each other. For example, \(P_{g_1}^1\) receives \(m_{g_1}^1 \rightarrow m_{g_1}^2 \rightarrow m_{g_1}^3\) in order in Fig. 1. Similarly, \(P_{g_1}^2\) and \(P_{g_1}^3\) receive \(m_{g_1}^1 \rightarrow m_{g_1}^2 \rightarrow m_{g_1}^3 \) and \(m_{g_1}^2 \rightarrow m_{g_1}^1 \rightarrow m_{g_1}^3\), respectively. Due to this discrepancy between RSNs of each identical message, if \(P_{g_1}^1\) crashes in this example, \(P_{g_1}^2\) and \(P_{g_1}^3\) may not provide \(P_{g_1}^1\) with RSNs of the three messages assigned by \(P_{g_1}^1\) before failure even though they have the same contents of the messages in their volatile memories like in Fig. 1. Therefore, the loss of the state interval starting with each received message may cause state intervals of other non-faulty processes to have to be undone up to their last recoverable state intervals (excluding themselves).

In order to solve the common drawback of the previous works mentioned earlier, we propose a novel method to significantly reduce the rollback propagation effect of optimistic message logging by allowing redundant message contents to be capable of being used for recovering failed processes to consistent states to a maximum, if possible,
their pre-failure states like in the pessimistic message logging. Specifically, when a process \( P_{gi} \) receives a message \( m^e_{gi} \) to group \( gi \) from a sender \( S_w \) and assigns a RSN to the message, it informs other group members of the RSN after saving the log information of the message in its volatile memory. Here, the number of the group members keeping the RSN in their memories is called degree of redundancy, \( k \). Thus, after having obtained acknowledgments of the notification from \( (k-1) \) other members, the state interval beginning with the message is \( k \)-stable, which means this interval can be reconstructed by the recovering process \( P_{gi} \), even if \( (k-1) \) group members having the RSN (including \( P_{gi} \)) might crash. Then, \( P_{gi} \) should update its transitive dependency tracking information with the dependency vector attached on \( m^e_{gi} \) by taking the component-wise maximum of all elements. If \( P_{gi} \) tries to send another message \( m^e_{gi} \) depending on \( m^e_{gi} \) before obtaining the acknowledgments from other members, it should piggyback its transitive dependency tracking information on the outgoing message. If this optimism is disallowed and it is assumed there might occur at most \( (k-1) \) concurrent process failures, this method enables optimistic message logging to have the same feature of \( k \)-pessimistic message logging. To decrease the number of update messages required for RSN notification to other members, this method forces each group member to piggyback its assigned RSN of every received message on outgoing acknowledgments. This optimization procedure may reduce the failure-free overhead resulting from the log information replication to a certain extent. To satisfy this requirement, each process should have the following information in the proposed method.

- **\( GCMLog^1_{gi} \)**: It is a vector which maintains \( e(\text{sid}, ssn, data, rsn, \text{RSNCount}, listOf\text{RSNs}) \) of each message received by \( P_{gi} \). Here, \( e \) is the log information of the message and the first four fields are the identifier of the sender, the send sequence number assigned by the sender, data, and the receive sequence number assigned by \( P_{gi} \) of the message respectively. The fifth field is the counter for recording the number of process members in group \( gi \) including \( P_{gi} \) that knows \( rsn \) of the message in their volatile memories. Its maximum value is equal to the degree of redundancy of RSNs, \( k \). The sixth element is a set of elements whose form is \( (\text{pid}, drsn) \), where \( \text{pid} \) is another group member that has assigned \( drsn \) as RSN to the message when \( \text{pid} \) received it. This set can help other crashed group members replay the message in their pre-failure orders with its corresponding \( drsn \).

- **\( RSN^1_{gi} \)**: the receive sequence number of the latest message delivered to \( P_{gi} \).

- **\( SSN^1_{gi} \)**: the send sequence number of the latest message sent by \( P_{gi} \).

- **\( SSNVector^1_{gi} \)**: It is a vector where \( SSNVector^1_{gi} [q] \) is the \( ssn \) of the last message that was delivered to \( P_{gi} \) from a process \( P_{gi} \).

- **\( TDTIVector^1_{gi} \)**: It is a vector for tracking dependencies between states of \( P_{gi} \) and another process member in group \( gi \). Here, an entry \( TDTIVector^1_{gi} [q] \) is the form of \( (incar, rsn) \) where \( incar \) is the incarnation number indicating the number of rollbacks \( P_{gi} \) has experienced due to its own failures or others so far, and \( rsn \) is RSN of the latest message \( P_{gi} \) knows \( P_{gi} \) has received during the period of the incarnation number \( incar \). It is very similar to the one in the previous work [2].

For example, Fig. 2 shows how our method can have the beneficial feature mentioned above in the same case of Fig. 1. When \( P_{gi} \) receives \( m^1_{gi} \), it assigns a RSN, \( i \), to the message and records its tag \( (\text{sid}, ssn) \), data and RSN on \( GCMLog^1_{gi} \), and sends a control message \( \text{updateRSN} \), including its tag and RSN, to two group members \( P^2_{gi} \) and \( P^3_{gi} \). if the degree of redundancy is assumed to be 3. With this notification, the two processes can keep the RSN of \( m^1_{gi} \), \( i \), in \( GCMLog^2_{gi} \) and \( GCMLog^3_{gi} \), respectively. In this case, \( P^2_{gi} \)

![Fig. 2](image-url)  
Example showing how the proposed method can address the problem.
and $P_i^1$ have also received message $m_i^1$, and saved its log information into each of their own GCMLog$_g^1$. If the two processes have not transmitted their updateRSN messages for $m_i^1$ to $P_i^1$, they can put their assigned RSNs of $m_i^1$, $(j+2)$ and $(k+1)$, on an acknowledgment destined to $P_i^1$ in Fig. 2 (a). After having updated the element for $m_i^1$ on $P_i^1$ with the RSN information piggybacked on the acknowledgment message, $P_i^1$ should also confirm its RSN update completion to them respectively. At this point, if there are other messages that $P_i^1$ has received and whose RSNs have been assigned by $P_i^1$, but not notified $P_i^2$ and $P_i^3$ of, and vice versa, their log information may be included in the ongoing acknowledgments for decreasing communication overheads of the method. In the same way, our method makes $P_i^2$ and $P_i^3$ maintain each other’s assigned RSN in its own GCMLog$_g^2$ like in Fig. 2 (b). Afterwards, even if, among three processes, two concurrent crashes might occur, one surviving process can provide all log information of the three messages for the two failed processes to be capable of replaying them in the same order like before their failures. Therefore, this advantageous feature can allow optimistic message logging to significantly reduce the possibility that non-faulty processes should be rolled back due to others’ failures. The detailed formal description of our algorithm is shown in Fig. 3.

**Theorem 1.** Optimistic message logging with the proposed method enables any failed process $p$ to recover in order to be consistent with every other process.

*Proof:* Suppose the set of all the messages $p$ has received before failure is denoted by SET-OF-MGS$_p$. The proof proceeds by induction on the number of all the messages in SET-OF-MGS$_p$, denoted by NUMOF(SET-OF-MGS$_p$).

**[Base case]**

In this case, there is one message $m$ to group $i$ that $p$ has received and there are two cases we should consider.

**Case 1:** there is at least one non-faulty process $q$ in group $i$ which is maintaining $m$’s log information $(m.sid, m.ssn, m.data, p, m.rsn)$ for $p$’s recovery in GCMLog$_g^i$.

In this case, $p$ can trivially obtain $m$’s log information from $q$ and replay $m$ in the same order like in $p$’s pre-failure state. Thus, $m$ will not make any process orphan regardless of whether there is any message $m’$ has sent to another process after $m$’s receipt of $m$ or not.

**Case 2:** there is no non-faulty process which is keeping $m$’s log information for $p$’s recovery in its volatile memory.

In this case, there are two subcases we should consider.

**Case 2.1:** there is no message sent to another process after $m$’s receipt of $m$.

In this case, even though $p$ regains $m$ from $m$’s sender and replays it in any order, $m$ will not make any process orphan.

**Case 2.2:** there is at least one message sent to another process in group $j$ after $m$’s receipt of $m$.

In this case, when $p$ is about to send any message $m’$ depending on $m$ to another non-faulty process $r$, it should have piggybacked its transitive dependency tracking information TDTIVector$_g^1$, on $m’$. After having received $m’$, $r$ has updated its transitive dependency tracking information TDTIVector$_g^1$ with TDTIVector$_g^2$, attached on $m’$ by taking the component-wise maximum of all elements excluding its own. Thus, using TDTIVector$_g^2$, $r$ can be rolled back to its consistent state regarding $p$’s recovered state in the same way as in the traditional optimistic message logging.

**[Induction hypothesis]**

We assume that the theorem is true for $p$ in case that NUMOF(SET-OF-MGS$_p$) = $k$.

**[Induction step]**

By induction hypothesis, $p$ can obtain the log information
of all the other k messages. Therefore, if p can obtain the log information of \((k + 1)\)-th message during p’s recovery, the theorem is true for p in case that \(\text{NUMOF}(\text{SET-OF-MSG}1) = k + 1\). We assume that the \((k + 1)\)-th message is \(m\). The following case is similar to the base case stated above.

By the induction, after having executed optimistic message logging with the proposed method, any failed process can recover to be consistent with every other process.

3. Experiments

In this section, extensive simulations are performed to compare the optimistic message logging with our proposed method (POML) with the traditional one [2] (TOML) using a discrete-event simulation language [1]. Two performance indices are used for comparison of failure-free overhead and recovery overhead respectively; the elapsed time until the same distributed execution has been completed \(T_{\text{complete}}\) and the total number of state intervals rolled back in the entire system due to process failures (NOSI). A system with \(N\) nodes connected through a general network is simulated. Each node has one process executing on it and, for simplicity, the processes are assumed to be initiated and completed together. Each process group consists of five processes and the target of each message sent from a process is always a process group. Thus, IP multicast is used for multicasting a message to a group of processes. The message transmission capacity of a link in the network is 100 Mbps and its propagation delay is 1 ms. Every process has a 128 MB buffer space for storing its GCMLog. The size of application messages ranges from 1 KB to 1 MB. Normal checkpointing is initiated at each process with an interval following an exponential distribution with a mean \(T_{\text{ms}} = 300\) seconds. In addition, a message to a process group is sent from a randomly chosen process with an interval following an exponential distribution with a mean of \(T_{\text{ms}} = 3\) seconds. All experimental results shown in this simulation are all averages over a number of trials.

Figure 4 shows \(T_{\text{complete}}\) for the two protocols, TOML and POML, with varying the degree of redundancy \(k\) in case the number of processes \((\text{NOP})\) is 100 and \(T_{\text{complete}}\) of TOML (POML with \(k = 1\) as shown in this figure) is 300 minutes. As \(k\) becomes bigger, \(T_{\text{complete}}\) of POML also increases because its log replication cost is higher. However, this simulation results indicate the differences between \(T_{\text{complete}}\) of POML and \(T_{\text{complete}}\) of TOML range only from 2% through 9.6% of the latter. As \(k\) grows larger than 4, the increasing rate of \(T_{\text{complete}}\) of POML is stepping up at a much higher speed.

Figure 5 presents NOSI for the two protocols for the specified range regarding the number of processes \((\text{NOP})\) in case two randomly chosen processes crash simultaneously. Here, the degree of redundancy of POML, \(k\), is configured to 2 and 3 respectively. In this figure, as NOP increases, their NOSIs are also becoming larger. This phenomenon results from the reason that complex message exchanging among more processes or groups may greatly increase the number of state intervals nullified by rolling back more non-faulty processes due to two concurrent failures. However, this figure indicates POML \((k = 2)\) and POML \((k = 3)\) reduce approximately 21-54% and 42-72% of NOSI compared with TOML respectively. The reason is that POML allows the other live group members of each failed process to help the latter replay its received messages before failure to a maximum using RSNs of the messages in their volatile memories unlike TOML. Also, we can see that POML \((k = 3)\) decreases about 27-40% of NOSI compared with POML \((k = 2)\) in this figure. It means that increasing \(k\)’s value may make the reduction rate of NOSI much higher.

4. Conclusion

This paper proposed a new method to significantly reduce the rollback propagation effect of the previous optimistic message logging on reliable FIFO group communication infrastructures. This beneficial feature results from its making
group members replicate different receive sequence numbers, which they assigned for each identical message to their group respectively, into their volatile memories. Simulation results show that our proposed method can perform better than the previous one in terms of the rollback distance of non-faulty processes with a little normal time overhead.

Acknowledgments

This work was supported by Kyonggi University Research Grant 2011 (Project Number: 2011-048).

References

[1] R. Bagrodia, R. Meyer, M. Takai, Y. Chen, X. Zeng, J. Martin, and H.Y. Song, “Parsec: A parallel simulation environments for complex systems,” Computer, vol.31, no.10, pp.77–85, 1998.
[2] O.P. Damani, Y.-M. Wang, and V.K. Garg, “Distributed recovery with K-optimistic logging,” Journal of Parallel and Distributed Computing, vol.73, no.6, pp.1193–1218, 2003.
[3] E. Elnozahy, L. Alvisi, Y. Wang, and D. Johnson, “A survey of rollback-recovery protocols in message-passing systems,” ACM Computing Surveys, vol.34, no.3, pp 375–408, 2002.
[4] T. Ropars and C. Morin, “Active optimistic and distributed message logging for message-passing applications,” Concurrency and Computation: Practice and Experience, vol.23, no.17, pp.2167–2178, 2011.