Message Logging Protocol for Hierarchical Server Architectures

J. Ahn

1School of Computer Science and Eng., Kyonggi University, Suwon, Gyeonggi, Republic of Korea

Abstract—Two-level hierarchical architecture consisting of a set of node groups is one of the most well-known organizations for managing a large scale distributed system in an efficient way. This network organization generally has the topological feature that inter-group communication overhead is considerably higher than intra-group one. This paper presents a new message logging protocol as lightweight fault-tolerance technique to reduce the number of in-group control messages and the number of inter-group control messages while distributing the load of logging of the message as evenly as possible compared to the existing sender-based message logging ones with high failure-free overhead and recovery inefficiency.

Keywords: Hierarchical and Distributed Architecture, Fault-tolerance, Message Logging and Checkpointing, Recovery

1. Introduction

Distributed computing architectures require efficient fault-tolerance techniques for improving service availability in case of sequential node failures [3], [4], [5]. For this purpose, checkpointing and message logging may satisfy the requirement with much less computing resources during failure-free operation than process replication [2]. Among the message logging algorithms, Sender-Based Message Logging(SBML) can greatly reduce the cost of synchronous logging to the stable storage compared with receiver-based pessimistic message logging by maintaining the log information of each message onto the volatile memory buffer of its sender [3], [4], [5]. Also, if nodes or processes sequentially fail, they can obtain the contents and the receive sequence number on starting recovery. This behavioral feature may cause the following performance problems in the two-level hierarchical and distributed architectures.

- A message transmission in a group makes two additional in-group control messages between its sender and receiver.
- A message transmission between two groups makes two additional inter-group control messages between its sender and receiver.
- If a recovering process received at least one inter-group message before failure, it should broadcast a request for its log information to every other group on starting recovery.

In this paper, we present a novel message logging protocol consisting of three modules for two-level hierarchical and distributed architectures to address the drawbacks of sender-based message logging. The first module reduces the number of in-group control messages and, the rest, the number of inter-group control messages while localizing recovery. Also, it can distribute the load of logging and keeping inter-group messages to group members as evenly as possible.

2. New Message Logging Protocol

The proposed protocol consists of three modules to address the three drawbacks of the original sender-based message logging. The first two modules are performed for intra group messages and, the last, for inter group messages. The first is called causal message logging; when the first member of group 1, \(P_{1,1}\), sends message \(m_1\) to the second member of the same group in figure 1(a), \(P_{1,2}\), the log information of \(m_1\) except its receive sequence number, \(smi(m_1)\), is kept on its sender’s volatile buffer, \(s-log(1,1)\). Here, \(smi(m_1)\) is composed of the identifier of the receiver(\(rid\)), the send sequence number(\(ssn\)), and the data of the message(\(data\)). Receiving \(m_1\), \(P_{1,2}\) assigns a receive sequence number(\(rsn\)) to it, and then, saves the determinant of the message, \(det(m_1)\), in its buffer, \(d-set_{1,2}\). Here, \(det(m_1)\) consists of the identifier of the receiver(\(sid\)), \(ssn\), \(rid\), the receive sequence number of the message(\(rsn\)), and the identifier of the immediate dependent(\(did\)) which is the member having a copy of the determinant. When the receiver of \(m_1\), \(P_{1,2}\), sends message \(m_2\) to the third group member, \(P_{1,3}\), \(det(m_1)\) is piggy-backed on \(m_2\) to maintain in the buffer of the immediate dependent of \(m_1\), \(d-set_{1,3}\). Receiving \(m_3\), the immediate dependent of \(m_2\), \(P_{1,4}\), also keeps \(det(m_2)\) in its buffer \(d-set_{1,4}\) in the same way. However, when the fourth member, \(P_{1,4}\), attempts to send an inter group message \(m_4\) to a member of group 2, \(P_{2,1}\), the second module, named sender-based message logging, is executed to do group-wide output commit meaning, in case of \(P_{1,4}\)’s failure, it can obtain \(det(m_3)\) from one of its own group members, in particular, the sender of \(m_3\), \(P_{1,3}\), not \(P_{2,1}\), during recovery. This behavioral feature enables the in-group recovery for reducing the number of recovery control messages from other groups. Afterwards, if
$P_{1,4}$ fails like in figure 1(b), sender-based logging allows $P_{1,4}$ to get $m_3$ and its rsn in $det(m_3)$ from $P_{1,3}$ to restore to its pre-failure state. In case of $P_{1,3}$’s failure, causal logging enables $P_{1,3}$ to attain $m_2$ from $P_{1,2}$ and its rsn in $det(m_2)$ from $P_{1,4}$. Therefore, our protocol can considerably alleviate the first drawback of the previous ones stated earlier.

When any message from a group is transmitted to another group, its receiver performs the third module, called immediate dependent message logging (IDML). Figure 2(a) shows an example of this case that $P_{1,4}$ sends message $m_4$ to $P_{2,1}$ and keeps $smi(m_4)$ in its buffer $s-log(1,4)$. In this example, $P_{2,1}$ first saves $smi(m_4)$ in another send log buffer for inter-group messages directly received by itself, $d-log(2,1)$, and then, assigns a rsn to $m_4$ and records $det(m_4)$ in its buffer $d-set(2,1)$. As it sends $m_4$ to another member of the same group, $P_{2,2}$, $smi(m_4)$ and $det(m_4)$ are transmitted with $m_4$. In this case, $P_{2,2}$ saves them in its buffers, $s-log(2,2)$ and $d-set(2,2)$, respectively. Afterwards, intra-group messages like $m_5$, $m_6$ and $m_7$ are handled by causal

message logging module as mentioned above. Thus, when $P_{2,1}$ crashes like in figure 2(b), $P_{2,2}$ can provide $m_4$ and its rsn in $det(m_4)$ to $P_{2,1}$ to recover consistently without any help of the sender of $m_4$, $P_{1,4}$. From this example, we can see our protocol allows the logging procedure of each inter-group message to be fulfilled at its own receiver, not the group leader, improving scalability in terms of message logging load and log information maintenance. In addition, the protocol can require no extra control messages during failure-free execution and localize recovery in this message exchanging pattern.

References