Hybrid Log-based Fault Tolerant scheme for Mobile Computing System

Yongning Zhai[1, a], Zhenpeng Xu[1, b], Weini Zeng[1, c]

[1]Jiangsu Automation Research Institute, Lianyungang, Jiangsu, 222061, China

[a]email: xingyuant@126.com, [b]email:xuzhenpeng@jari.cn, [c]email:zengweini@jari.cn

Keywords: Mobile System, Fault Tolerant, Checkpoint, Message Log

Abstract. Many new characteristics are introduced in the mobile computing system, such as mobility, disconnections, finite power source, vulnerable to physical damage, lack of stable storage. Since the related log-based rollback recovery fault tolerant schemes may still lead to dramatic performance loss in failure-free or inconsistent recovery caused by the fault, a hybrid log-based fault tolerant scheme was proposed combining the checkpointing mechanism with the message logging mechanism in this paper. The checkpoint, the logs and the happened-before relations were all logged synchronously at local mobile hosts temporarily and asynchronously into the persistent storage at mobile support station. By contrast, the proposal incurred a lower failure-free overhead on the premise of the consistent recoverability, as the consistent recovery was supported.

Introduction

In Checkpointing and rollback-recovery schemes, each of the replicated state of the process is called a checkpoint [1-2]. Upon a fault, there is a recovery mechanism which brings the failure process to the normal execution [3-5]. The log-based rollback recovery schemes based on independent checkpointing, is preferable for fault tolerance of mobile computing [6]. Depending on how the determinants are logged to stable storage, log-based rollback recovery scheme is classified into three types: pessimistic logging, optimistic logging and causal logging [7].

In pessimistic schemes, the process has to block waiting for the determinant of each nondeterministic event to be stored on stable storage before the effects of that event can be seen by other processes or the outside world. Pessimistic logging simplifies recovery and garbage collection but hurts failure-free performance due to synchronous logging [8]. In optimistic schemes, the process does not block, and determinants are transferred to stable storage asynchronously [9]. Thus, optimistic logging does not require the application to block waiting for the determinants to be actually written to stable storage and only record the nondeterministic event, and therefore incurs little overhead during failure-free execution. However, this advantage comes at the expense of more complicated recovery, garbage collection and propagated rollback. In case of some loss of temporary logs, it may lead to unrecoverable rollback without well consideration of the input/output commit problem, as the input/output devices that cannot roll back. Causal schemes satisfy *always-no-orphans property* by ensuring that the determinant of each nondeterministic event that causally precedes the state of a process is either stable or it is available locally to that process. Therefore, it gets a balance between optimistic and pessimistic logging to maintain the dependency relations through each common computing message [10]. However, many records of the dependency relations require to be exchanged among the process. As a result, carrying the dependency relation graph information on each application message may lead to an unacceptable overhead, since each MH corresponds to one antecedence graph.

As shown in Figure 1, new characteristics are introduced in mobile computing, such as mobility, disconnections, finite power source, vulnerable to physical damage, lack of stable storage [6]. Mobile computing system, $MCS=\langle N, C\rangle$ contains a set of nodes N and a set of channels C. The set of nodes $N=M\cup S$ can be divided into two types, $M=\{MH_1, MH_2,..., MH_n\}$ is the set of Mobile Hosts (MH), while $S=\{MSS_1, MSS_2,..., MSS_m\}$ is the set of static nodes acting as the access point with extra processing power and storage capabilities, static backbone node[7]. The set of channels $C=W\cup W'$ can also be divided into two disjoint sets, the set of high-speed wired channels W and the set of low bandwidth wireless channels *W'*. All the channels *W* and *W'* provide reliable, sequenced FIFO delivery of messages[6]. For simplicity, only one process running on a MH is assumed. These distributed processes communicate each other only through exchanging computational messages, and interact with the outside world only through input/output commit [7]. The execution of the process is assumed to follow Piece-wise Deterministic (PWD) model, and the transient fault of the process follows fail-stop model [7].

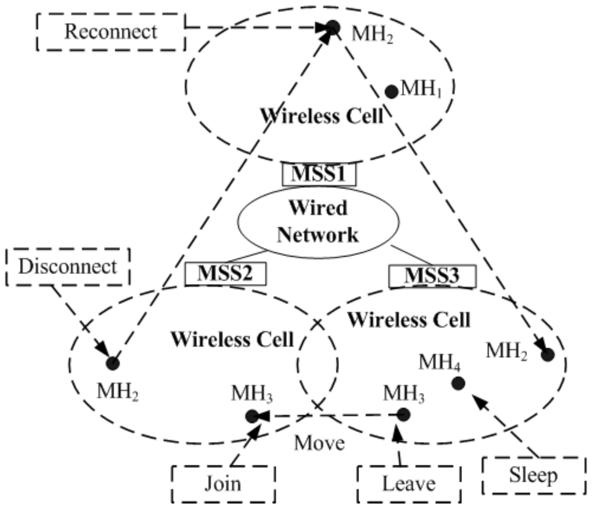

Fig.1. Mobile computing system

The Hybrid Logging Scheme

In our model, 'Deterministic' indicates that event *e* is not a nondeterministic event. The deliver sequence number, *e.dsn* encodes the order in which *e* is delivered by the computing process. Thus, if process P_i has delivered *e* and $e.dsn = \theta$, then *e* is the θth event that P_i has delivered. For simplicity, we refer to *#e* the determinant of *e*. This tuple *#e* determines event *e* and related information, which is useful for the rollback recovery phase.

To record the process state(experienced events) and happened-before relation exactly, the antecedence graph is introduced based on the state transition advanced by the nondeterministic event. Each MSS maintains only one antecedence graph for all the MHs in the local cell to support the possible rollback recovery. The antecedence graph contains a summary of the local cell's execution. The antecedence graph $AG =<E, V>$, contains the node set *V* including the logged determinant of the event experienced by each computing process, and the edge set *E* representing happened-before relations among the nodes in *V*.

Specifically, in the antecedence graph, each node σ corresponds to one logged event determinants *#e* ($\#e=\sigma\in V$). Let σ_θ^i indicates the θth node of P_i in the antecedence graph.

Only one entire antecedence graph of all the local MHs is managed and maintained by MSS. Locally, Each MH_i only maintains a temporary event logging queue EQ_i, to record all the events experienced by the local computing process P_i. Furthermore, the content of the checkpoint and message logs are also recorded in the graph node set *V*, in addition to the happened-before relations recorded in the edge set *E*. That means no dependency relations require to be exchanged among mobile hosts during failure-free execution.

Each MH_i manages and maintains a temporary version event logging queue EQ_i, to record all the events experienced by the local computing process P_i, including message-sending, message-receipt, input, output and the checkpointing event.

For checkpointing, the local checkpointable interface is inhabited at each MH. P_i takes a checkpoint with a fixed interval. Let $C(i,\alpha)$ denotes the αth checkpoint of P_i at MH_i.

The process takes periodic checkpoint to limit the amount of work that has to be repeated in execution replay upon recovery. The checkpoint period between two consecutive checkpointing of P_i is determined by itself. That means the process takes local checkpoints asynchronously.

Time to take checkpoint, the replicated process state of P_i is created by MH_i using the checkpointing operation of the local checkpointable interface. The new checkpoint is encapsulated into a determinant tuple #*e* with ‘checkpoint’ flag, and #*e* is appended into the event logging queue EQ_i. After that, procedure Update_AG (EQ_i) is invoked for updating the antecedence graph. After the updating, the logged determinant in the local EQ_i is cleared, to free the storage space in MH_i.

During the normal failure-free execution, each event *e*, delivered by the local P_i, is passed to the local logging mechanism, packed into the determinant #*e*, through Message_logging (*e*).

Upon a user input from outside world, a copy of the input is firstly passed to the local logging mechanism, packed into the determinant #*e*, through Message_logging(*e*). After the event logging, MH_i starts to process the input. Similarly, before interacting with the outside world to show the outcome of, a copy of the outcome is forwarded to local logging mechanism for logging firstly. During procedure Message_logging(*e*), if the experienced event is an input or output, EQ_i is sent to the local MSS_p for updating the antecedence graph.

Upon the specific event(Periodically, the event logging queue of MH), EQ_i is transferred to the local MSS_p,

Considering the reliability of the fragile MH, each MSS_p manages and maintains a persistent antecedence graph AG_p, to record the logs and happened-before relations in the local cell on behalf of the local mobile hosts. The message determinants and checkpoints of each local MH are positioned logically in the antecedence graph AG_p, according to the happened-before relations defined in definition 1.

To update the persistent antecedence graph, MH_i sends the event logging queue EQ_i to the local connected MSS_p upon the specific event. Upon the receipt of the event logging queue EQ_i, the local MSS_p add the content of the event logging queue into the antecedence graph AG_p, through procedure Update_AG (AG_p, EQ_i).

For each logged determinant #*e* in EQ_i, firstly a corresponding graph node σ_θ^i of AG_p is created by create_node_AG (AG_p, #*e*) according to #*e*. Then, the previous graph node $\sigma_{\theta-1}^i$ precedes to #*e* is retrieved in AG_p. The corresponding edge between $\sigma_{\theta-1}^i$ and σ_θ^i of AG_p is created by Add_edge_AG (AG_p, $\sigma_{\theta-1}^i$, σ_θ^i), according to happened-before relation defined in definition 1.

If the new created graph node σ_θ^i corresponds to a normal message-receipt event, The related graph node σ_β^j corresponding to the message-sending event is retrieved from AG_p firstly, and if the σ_β^j exists in AG_p, then the corresponding edge between σ_β^j and σ_θ^i of AG_p is created by Add_edge_AG (AG_p, σ_β^j, σ_θ^i), according to happened- before relation defined in definition 1.

If the new created graph node σ_θ^i corresponds to a normal message-sending event, The related graph node σ_β^j corresponding to the message-receipt event is retrieved from AG_p firstly, and if the σ_β^j exists in AG_p, then the corresponding edge between σ_θ^i and σ_β^j of AG_p is created by Add_edge_AG (AG_p, σ_θ^i, σ_β^j), according to happened- before relation defined in definition 1.

If the new created graph node σ_θ^i corresponds to a checkpointing event, the procedure Garbage_Collection (AG_T_p) is invoked to delete the nodes in AG_p, happened-before the new checkpoint to free the storage space.

Figure 2(a) presents a simple scene of mobile computing in which all the messages to and from the local MHs are traversed through its connected MSS locally. The simple system contains three mobile hosts, MH_1, MH_2 and MH_3. All the MHs reside in the geographical cell of MSS_p, CL_p. In practice, a typical system may contain many such components of MH and MSS.

According to the proposed log-based hybrid scheme, the copy of P_k’s input I_1 in m_3, the checkpoint of P_j in m_4 and the copy of output O_1 in m_5, are received by the local MSS_p for logging at time t_1, t_2 and t_3, respectively, as shown in Figure 2(a).

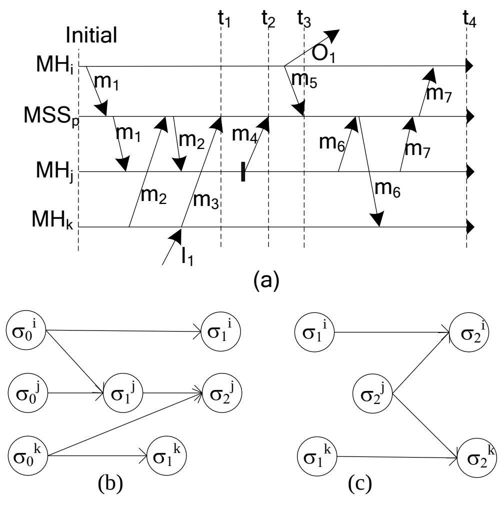

Fig. 2. An example of the antecedence graph

According to the proposal, Figure 2(b) and Figure 2(c) show the recorded temporary and persistent antecedence graph of a cell at time t_4, at MSS_p. Specifically, in persistent AG_P_p, node σ_2^i records $\#m_7$, while node σ_2^j records $\#m_2$. In temporary AG_T_p, node σ_2^i records $\#m_7$, while node σ_2^j records $\#m_2$, $\#m_4$, $\#m_6$ and $\#m_7$.

The Corresponding Recovery

For process P_i, a complete log consists of the latest checkpoint and logged determinants of all the experienced nondeterministic events after the checkpoint. The proposed log-based hybrid scheme supports independent rollback-recovery with the complete log. Considering the reliability of the fragile MH, the rollback recovery can be classified into three cases as follows.

Case 1. If the content of the local event logging queue EQ_i is still available upon a process fault of MH_i, then MH_i directly reloads the latest checkpoint from EQ_i, and replays the following logged determinants after the checkpoint. According to the content of C, the failure process is able to recover the lost computation independently, through replaying nondeterministic event logs in the original irreflexive partial order.

Case 2. Only one failure process P_i loses the corresponding content of the local event logging queue EQ_i, and no other failure process loses the local during the following recovery phase. In that case, the checkpoint and all the nondeterministic event logs in lost EQ_i, can be extracted from the latest local antecedence graph AG_p. According to the checkpoint and nondeterministic event logs extracted from AG_p, the failure process is able to recover the lost computation independently.

Case 3. More than one failure processes lose the corresponding local event logging queues, or another failure processes lose the corresponding content of the local event logging queue EQ_i during a recovery of Case 2.

The non-failure process may also require to rollback for a consistent recovery without the complete log. During the recovery, the logged events in the antecedence graph require to be replayed in the original irreflexive partial order.

If the content of the local event logging queue EQ_i does not available through checking, then MH_i sends a recovery request, to its local MSS_p. Upon the receipt of the recovery request, the updating AG request is broadcasted to MHs by MSS_p. Upon the receipt of the updating request, each MH sends its EQ to the local MSS for updating the antecedence graph. After each MSS updating the antecedence graph, the required recovery information EQ_i is extracted from the antecedence graph AG, and finally the extracted EQ_i is sent to the failure process for rollback recovery. Upon the receipt of the extracted EQ_i, MH_i reloads the latest checkpoint in EQ_i, and replays the following logged determinants after the checkpoint. According to the content of EQ_i, finally the failure process is able to recover the lost computation, through replaying nondeterministic event logs in the original irreflexive partial order.

During the recovery, deterministic event logs are useful to eliminate the repeated message to the non-rollback process. Upon the receipt of the recovery request, the sub-antecedence graph AG_F_r is extracted from temporary AG_T_p, and persistent AG_P_p. if AG_T_p is unavailable, AG_F_r is required retrieving from the antecedence graphs at other MSSs combining with AG_T_p, and the rollback has to be propagated to all the MHs in the local cell for consistent recoverability. At last, AG_F_r is sent or broadcasted to the rollback MH. AG_F_r contains the latest checkpoint, nondeterministic message logs and related happened-before relations, which are useful to recover the failure process.

The rollback recovery will be propagated to all the MHs in the local cell for consistent recoverability without the complete log. the lost nondeterministic messages in AG_T_p, the across-cell messages, can be retrieved from the antecedence graphs at other MSSs, since the proposal the antecedence graphs also records the local deterministic sending events, which is the nondeterministic events for the failure destination MH in other cell. Under PWD assumption, the lost computation within the cell can be deterministically recreated during the recovery.

Performances

For performance analysis, the fixed checkpoint interval of MH_i is T_i. Log arrival rate of MH_i follows a Poisson process with rate λ_i per time unit. Let α denote the ratio of the number of input/output to the total experienced messages. Let β denote the ratio of the number of connecting message to the total messages. Let θ denote the ratio of the size of a checkpoint to a normal messages.

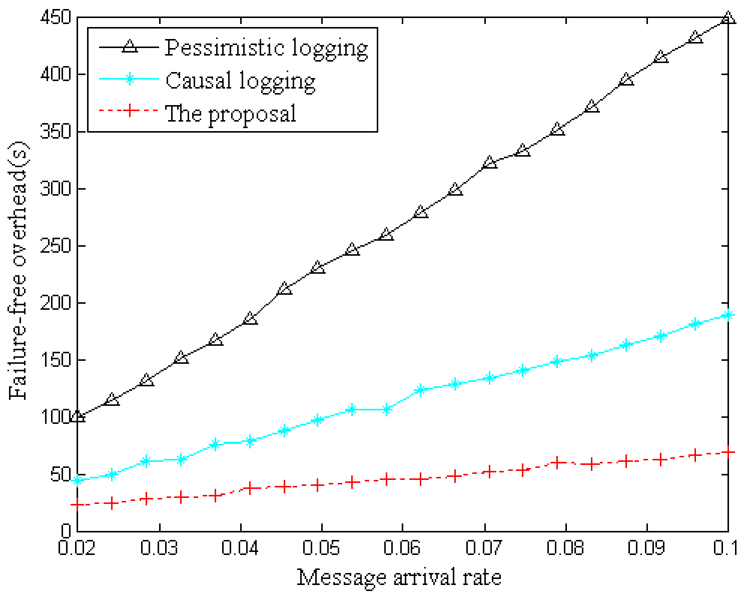

Fig. 3. Average logging overhead incurred

The average logging overhead during the failure-free incurred by various logging schemes with the varying message rate, is shown in Figure 3, where α=3%, β=2.5% and θ=30. The y-axis indicates the average logging overhead incurred by access to the temporary and stable storage for logging and corresponding operations, while the x-axis denotes the message arrival rate of a process.

As shown in Figure 3, the proposed log-based hybrid scheme incurs a lower average logging overhead than pessimistic and causal logging schemes. The reason is that proposed log-based hybrid does not require the application to block but in case of the specific event, which reduced the number of stable storage accessing. However, pessimistic logging requires the determinant of each message to be logged synchronously into the stable storage, and leads to a high overhead of frequent connection to the stable storage and relay blocking. In the causal logging, much overhead is incurred by updating dependency relations among the antecedence graphs of mobile hosts, since each mobile host holds one antecedence graph.

Conclusion

In this paper, we specifically address a hybrid log-based rollback recovery scheme for mobile computing, in which all the messages experienced by the process and their happen-before relations are encoded and recorded in the antecedence graph by the local mobile support station. For efficiency, the antecedence graph is classified into temporary and persistent versions. The temporary version is asynchronously transferred to the stable storage upon the specific event. The proposed logging scheme enables the recoverability of the failure process by the independent or propagated rollback styles. The performance of the proposal is evaluated by the extensive simulation. By contrast, the results show that the proposal incurs a lower failure-free overhead on the premise of the consistent recoverability.

Acknowledgement

In this paper, the work was sponsored by the Natural Science Foundation (No. 61303045).

References

[1] Kuang P., Field J., Varela C. A.. Fault tolerant distributed computing using asynchronous local checkpointing[C]. Proceedings of the 2014 ACM SIGPLAN Workshop on Programming Based on Actors, Agents, and Decentralized Control, 2014:81-93

[2] Meroufel B., Belalem G.. Lightweight coordinated checkpointing in cloud computing[J]. Journal of High Speed Networks, 2014, 20(3):131-143

[3] Islam T.Z., Bagchi S., Eigenmann R.. Reliable and Efficient Distributed Checkpointing System for Grid Environments[J]. Journal of Grid Computing, 2014, 12(4):593-613

[4] Mendizabal O.M., Marandi P.J., Dotti F.L.. Checkpointing in parallel state-machine replication[C]. 18th International Conference on Principles of Distributed Systems, 2014, 8878:123-138

[5] Awasthi L. K., Misra M., Joshi R. C. et al.. Minimum mutable checkpoint-based coordinated checkpointing protocol for mobile distributed systems[J]. International Journal of Communication Networks and Distributed Systems, 2014, 12(4):356-380

[6] Sunil Kumar Gupta, R. K Chauhan, Parveen Kumar. Backward error recovery protocols in distributed mobile systems: a survey[J]. Journal of Theoretical and Applied Information Technology, 2008, 4 (4): 337-347

[7] E.N.Elnozahy, L.Alvisi, Y.M.Wang and D.B.Johnson. A Survey of Rollback-Recovery Protocols in Message-Passing Systems[J]. ACM Computing Surveys, 2002, 34(3):375-408

[8] Taesoon Park, Namyoon Woo, Heon Y.Yeom. An Efficient Recovery Scheme for Mobile Computing Environments[J]. Future Generation Computer Systems, 2003,19: 37-53

[9] Taesoon Park, Namyoon Woo, Heon Y. Yeom. An Efficient Optimistic Message Logging Scheme for Recoverable Mobile Computing Systems[J]. IEEE Transactions on Mobile Computing, 2002, 1(4):265-277

[10] Zhang Zhan, Zuo Decheng, Ci Yiwei and Yang Xiaozong. A Rollback Recovery Algorithm Based on Causal Message Logging in Mobile Environment[J]. Journal of Computer Research and Development, 2008, 45(2): 348-357