Hybrid Message Logging Protocols for Fast Recovery

Sriram Rao, Lorenzo Alvisi, and Harrick M. Vin
Department of Computer Sciences, UT-Austin

1 Introduction

Fast recovery has received little attention in the context of message logging protocols, which have instead focused on minimizing their overhead during failure-free executions. As distributed computing becomes commonplace, and many more applications are faced with the current costs of high availability, there is a fresh need for recovery-based techniques that combine high performance during failure-free executions with fast recovery.

As an initial step towards the development of these new techniques, we have implemented a sender-based pessimistic protocol [5], a receiver-based pessimistic protocol [2] and a causal protocol [1] and studied their performance during recovery [4] [1].

All of these protocols log (1) the content and (2) the order of receipt of each message delivered by each process during its execution. Processes synchronously log on stable storage [2] the order of receipt, encoded in tuples called *determinants*. Receiver-based pessimistic protocols store on stable storage also the content of each message. However, this is not necessary: it can be shown that if all determinants are available during recovery, then the content of any message can be regenerated. Sender-based pessimistic and causal protocols take advantage of this observation to reduce logging overhead significantly during failure-free executions by keeping the contents of messages in volatile memory at the corresponding senders.

Table 1 shows the results of our experiments. [3] Recovery in sender-based protocols is 2% to 20% slower than in receiver-based protocols when $f = 1$, where f is the maximum number of concurrent failures. As f increases, however, sender-based protocols become as much as 390% slower. The cause for this peformance degradation is the increase in the time necessary to acquire the content of messages that have to be replayed during recovery. In receiver-based protocols these messages can be retrieved from stable storage, independent of the value of f.

In sender-based protocols, messages are stored only in the senders' volatile memory and are lost if the sender fails. Consequently, whenever a recovering process needs to acquire one of these missing messages, it stops rolling forward, waiting for the sender to recover to the point at which the message is regenerated and resent. We call this effect *stop-and-go*. As f increases, more messages are temporarily lost, and *stop-and-go* has an increasingly adverse effect on the overall recovery time.

2 Hybrid Protocols

Our results indicate that existing logging protocols exhibit a tradeoff between fast failure-free execution and fast recovery. To address this tradeoff, we introduce a new class of *hybrid* protocols. The objective of hybrid protocols is to maintain the failure-free performance of sender-based protocols while approaching the performance of receiver-based protocols during recovery. In hybrid protocols, messages are logged both at the sender *and* at the receiver. The sender synchronously logs messages in its volatile memory; the receiver asynchronously logs messages to stable storage. Since logging on stable storage is asynchronous, performance during failure-free executions is virtually identical to that of sender-based protocols. However, recovery is much faster, since *stop-and-go* cannot occur when the recovering process rolls forward using the messages available from stable storage. Note that in hybrid protocols stable storage does not contain all messages needed during recovery (as it did in receiver-based protocols). Missing messages are either available in the volatile memory of other operational processes, or have to be regenerated if the sender has failed. In either case, a process can roll forward using the messages on stable storage while in parallel acquires the missing messages.

We have implemented and evaluated hybrid versions of sender-based pessimistic and causal protocols. We show the results for hybrid causal protocols in Table 2. Our ex-

[1] Our study includes also a receiver-based optimistic protocol [3]. For the sake of space, in this abstract we focus on pessimistic and causal protocols, which try to speed up recovery by preventing the rollback of correct processes. A full discussion of our experiments can be found in [4]

[2] Pessimistic protocols implement stable storage using the file server's disk, while causal protocols use the volatile memory of at least $f + 1$ processes, where f is the maximum number of concurrent failures.

[3] Since the performance during recovery of causal and sender-based pessimistic protocols does not vary significantly, we only show the results for the causal protocol.

Application	f	Receiver-based				Sender-based (Causal)				Percentage Increase
		Restore State (sec.)	Acquire Logs (sec.)	Roll Forward (sec.)	Total Recovery Time (sec.)	Restore State (sec.)	Acquire Logs (sec.)	Roll Forward (sec.)	Total Recovery Time (sec.)	
grid	1	0.31	1.4	30.28	31.99	0.36	3.67	30.74	34.77	9%
	2	0.34	1.43	32.1	33.87	0.41	7.52	80.69	88.62	162%
	3	0.39	1.5	32.29	34.18	0.43	6.1	127.63	134.16	293%
nbody	1	0.29	3.5	78.01	81.8	0.35	4.42	78.1	82.87	2%
	2	0.41	3.7	78.19	82.3	0.47	10.64	112.89	124.0	50%
	3	0.42	4.1	78.3	82.82	0.49	7.25	129.71	137.45	66%
gauss	1	2.61	11.15	207.13	220.89	2.56	15.7	210.7	228.96	4%
	2	3.1	13.15	208.7	224.95	3.22	25.3	229.88	258.4	15%
	3	3.7	14.7	209.4	227.8	3.78	21.7	235.3	260.78	16%
life	1	0.31	0.95	41.1	42.36	0.36	2.55	40.97	43.88	4%
	2	0.41	1.3	43.36	45.07	0.45	6.1	108.84	115.39	156%
	3	0.46	1.5	43.8	45.76	0.5	4.25	151.09	155.84	240%
p2fox	1	1.85	0.8	22.29	24.94	1.87	5.91	21.39	29.17	17%
	2	2.1	1.23	22.7	26.03	2.2	10.39	67.92	80.51	209%
	3	2.7	1.64	22.78	27.12	2.65	8.43	122.24	133.32	390%

Table 1: The cost of recovery in receiver-based pessimistic and sender-based (causal) protocols. These experiments are with 4 processes, and failures are induced approximately 3 minutes after a checkpoint. The last column shows the percentage increase in recovery cost of sender-based protocols when compared to receiver-based protocols.

periments show that hybrid logging dramatically reduces the recovery cost of sender-based protocols, which are now within 30% of receiver-based protocols, even when $f > 1$. At the same time, Table 3 shows that when hybrid logging is used in place of sender-based logging, application execution time increases by at most 1%.

In conclusion, our experiments show that to avoid the *stop and-go* effect, logging protocls should not rely on other processes to providethe messages that have to be replayed during recovery. Hybrid protocols approach this goal, without imposing a high overhead during failure-free executions.

References

[1] L. Alvisi and K. Marzullo. Message logging: Pessimistic, optimistic, causal, and optimal. *IEEE Transactions on Software Engineering*, 24(2):149—159, February 1998.

[2] A. Borg, J. Baumbach, and S. Glazer. A message system supporting fault tolerance. In *Proceedings of the Symposium on Operating Systems Principles*, pages 90–99. ACM SIGOPS, October 1983.

[3] O. P. Damani and V. K. Garg. How to recover efficiently and asynchronously when optimism fails. In *Proceedings of the 16th International Conference on Distributed Computing Systems*, pages 108–115, 1996.

[4] S. Rao, L. Alvisi, and H. Vin. The cost of recovery in message logging protocols. Technical Report 98-02, University of Texas at Austin, March 1998.

[5] R. E. Strom, D. F. Bacon, and S. A. Yemini. Volatile logging in n-fault-tolerant distributed systems. In *Proceedings of the Eighteenth Annual International Symposium on Fault-Tolerant Computing*, pages 44–49, 1988.

Application	f	Restore State (sec.)	Acquire Logs (sec.)	Roll Forward (sec.)	Total Recovery Time (sec.)	Percentage Increase
grid	1	0.3	1.2	30.58	32.08	0%
	2	0.32	1.23	37.22	38.77	14%
	3	0.42	1.25	42.5	44.17	29%
nbody	1	0.32	2.9	77.75	80.97	0%
	2	0.35	2.86	89.8	93.01	13%
	3	0.38	2.92	95.3	98.6	19%
gauss	1	2.75	9.8	211.01	223.56	1%
	2	3.2	9.83	216.58	229.61	2%
	3	3.68	9.91	223.5	237.09	4%
life	1	0.33	0.78	41.5	42.61	0%
	2	0.42	0.81	47.3	48.53	7%
	3	0.45	0.83	53.2	54.48	19%
p2fox	1	1.9	0.6	22.6	25.1	0%
	2	2.25	0.7	33.5	36.45	11%
	3	2.73	0.75	42.7	46.18	30%

Table 2: Recovery cost of hybrid protocols for the same experimental environment of Table 1. The last column shows the percentage increase in recovery cost of hybrid protocols when compared to receiver-based protocols.

Application	Additional Overhead
grid	0.03%
nbody	0.057%
gauss	0.76%
life	0.041%
p2fox	0.2%

Table 3: Overhead of hybrid logging on failure-free execution when compared to sender-based logging