Managing Receiver-Based Message Logging Overheads in Parallel Applications

Hugo Meyer*, Dolores Rexachs, and Emilio Luque

Computer Architecture and Operating Systems Department,
University Autonoma of Barcelona, Barcelona, Spain
{hugo.meyer}@caos.uab.es
{dolores.rexachs,emilio.luque}@uab.es
http://www.uab.es

Abstract. Using rollback-recovery based fault tolerance (FT) techniques in applications executed on Multicore Clusters is still a challenge, because the overheads added depend on the applications' behavior and resource utilization. Many FT mechanisms have been developed in recent years, but analysis is lacking concerning how parallel applications are affected when applying such mechanisms. In this work we address the combination of process mapping and FT tasks mapping on multicore environments. Our main goal is to determine the configuration of a pessimistic receiver-based message logging approach which generates the least disturbance to the parallel application. We propose to characterize the parallel application in combination with the message logging approach in order to determine the most significant aspects of the application such as computation-communication ratio and then, according to the values obtained, we suggest a configuration that can minimize the added overhead for each specific scenario. In this work we show that in some situations is better to save some resources for the FT tasks in order to lower the disturbance in parallel executions and also to save memory for these FT tasks. Initial results have demonstrated that when saving resources for the FT tasks we can achieve 25% overhead reduction when using a pessimistic message logging aproach as FT support.

Keywords: Fault Tolerance, Mapping, Message Logging, Multicore, Overheads.

1 Introduction

Current High Performance Computing (HPC) systems are composed of nodes containing many processing units in order to execute more work in a short amount of time [1]. In order to take full advantage of the parallel environment,

* This research has been supported by the MICINN Spain under contract TIN2007-64974, the MINECO (MICINN) Spain under contract TIN2011-24384, the European ITEA2 project H4H, No 09011 and the Avanza Competitividad I+D+I program under contract TSI-020400-2010-120.

a good process mapping is essential. It is also important to consider that when executing parallel applications the fundamental objectives are: speedup as close as possible to the ideal (scalability) and efficient resource utilization.

Considering that applications are mapped into parallel environments in order to fulfill the above mentioned objectives, any disturbance may render all the mapping work useless. Currently, it is increasingly relevant to consider node failure probability since the mean time between failure in computer clusters has become lower [2] and this may cause loss of significant computation time in long-running applications. Indeed, successful completion of executions should be added to the list of fundamental objectives. In this vein FT techniques are gaining importance when running parallel applications. Nevertheless, FT mechanisms introduce disturbance to parallel applications in the form of overheads, which if not managed can result in large performance degradations, thus FT mechanisms that do no endanger scalability (uncoordinated approaches) are preferred.

Many recent works focus on finding the best checkpoint interval, or determining the best checkpoint or message logging approach for parallel applications [3][4] but few works address assigning resources for fault tolerance tasks considering applications' behavior [5].

When single-core clusters were the only option to execute parallel applications, there was not too many choices when talking about sharing resources. As there was only one computational core available, parallel applications share this resource (as well as the memory and cache levels) with the FT tasks if there was not dedicated resources. Considering that current clusters have more cores and memory levels, there is a need to develop mapping policies that allow parallel applications to coexist with the FT tasks in order to reduce the disturbance caused by these tasks. There is also important to consider that the number of cores has been multiplied by 8, 16, 32, 64 and usually the networks used in these clusters have not increase their speed to the same extent.

The main objective of this work is to determine the configuration of parallel applications in combination with a pessimistic receiver-based logging approach that minimizes the added overhead. We analyze parallel applications and obtain information that allows us to configure properly the FT tasks, specifically we determine if the best option is to share (compete for) resources with application processes or save resources for the FT tasks in order to reduce the introduced disturbance. In order to provide the configurations we consider the balance between computation and communication, message sizes and per-process memory consumption among other values.

The rest of the paper is organized as follows: Section 2 describes related work. Section 3 presents an analysis of the possible scenarios when executing a parallel application. Section 4 describes how to analyze a parallel application in order to find the most suitable message logging configuration. Section 5 shows the experimental validation and finally section 6 draws the main conclusions and mentions future works.

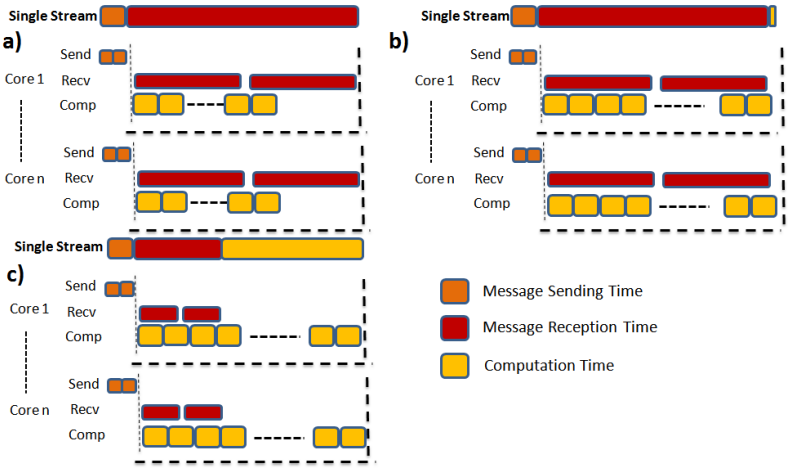


Fig. 1. Parallel Executions Scenarios in a SPMD App. a)Communication Bound. b) Computation and Communication overlapped. c) Computation Bound.

2 Related Work

In order to provide fault tolerance to parallel applications many strategies have been designed using message logging approaches [2][4][3]. Message logging approaches can sustain a much more adverse failure pattern, mainly due to a faster failure recovery. The main disadvantage of message log schemes is that they suffer from high overhead during failure-free executions [6], but they are an scalable solution since only failed processes must rollback, unless the domino effect is not addressed. Usually message logging techniques are used in combination with uncoordinated checkpoint approaches. Uncoordinated approaches are a good solution because there is not need for coordination between processes and there is no dependency on global components that could cause bottlenecks and compromise applications' scalability.

Following these lines, to develop this work we have used the RADIC (Redundant Array of Distributed and Independent Controllers) architecture [7], which uses a pessimistic receiver-based message logging technique in combination with an uncoordinated checkpoint approach in order to give application-transparent and scalable FT support for message passing applications.

In [3] a comparison between a pessimistic and optimistic sender-based logging approaches is presented where both seem to have a comparable performance. Nevertheless, when using sender-based approaches we should consider that in the presence of failures, processes that were not involved in the failure may need to re-send messages to restarted processes, and also garbage collection is complex. The pessimistic receiver based message logging approach of RADIC may be more costly than a sender based approach, but it guarantees that only failed processes will rollback to a previous state, without needing the intervention of other processes during re-execution.

In [8] was proposed a mechanism to reduce the overhead added using the pessimistic receiver-based message logging of RADIC. The technique consists in dividing messages into smaller pieces, so receptors can overlap receiving pieces

with the message logging mechanism. This technique and all the RADIC Architecture has been introduced into Open MPI in order to support message passing applications.

In [9] was presented an algorithm for distributing processes of parallel applications across processing resources paying attention to on-node hardware topologies and memory locales. When it comes to combine the mapping of FT tasks, specifically message logging tasks, with application process mapping, to date, no works have been published to the best of our knowledge.

3 Analyzing Parallel Applications Behavior

Current HPC parallel applications are executed on multicore systems, and the executions usually aim for almost lineal speedup and efficient resource utilization. In Figure 1 we present the three main scenarios possible when mapping applications in multicore systems. It is important to highlight that in this figure we are considering one iteration of a SPMD application. In Figure 1 we decompose the Single Stream in communications and computations operations. The main scenarios are:

1. **Communication bound**: Applications in which the processes are waiting because the communications take more time than the computations belong to this scenario. In Figure 1a we show how a communication bound application behaves (we are using as an example a SPMD application, where all processes do the same thing and each message goes from one process to another in a different core). In this figure we focus on showing how reception times (non-blocking send operations do not delay considerably the execution) can influence highly the execution time of a parallel application.
2. **Balanced Application**: This scenario is the best regarding efficient resource utilization, because the computational elements are working while the communication takes place. However, this behavior is very difficult to obtain because a previous study of the application is needed in order to completely overlap computations and communications (Figure 1b).
3. **Computation Bound**: When operators try to make a good use of the parallel environment they try to maintain the CPU efficiency high. Then in order to avoid the communication bound scenario it is recommended to give more workload per process which usually leads to a computation bound scenario. Figure 1c illustrates this scenario.

When characterizing a parallel application, it is also important to consider the number of processes that will be used, the number of nodes and the memory consumption of each process. This analysis should be done in combination with the analysis of the parallel environment in order to determine resource utilization. In this paper, we have characterized the parallel environment using application kernels and we consider the application phases (repetitive pieces of the parallel execution) that have the biggest weights during application execution.

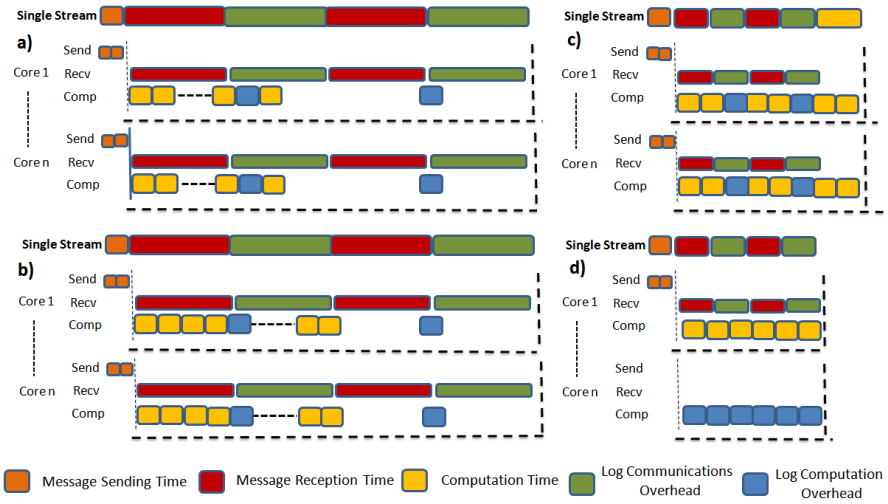


Fig. 2. Parallel Executions Scenarios with Message Logging. a)A communication bound application. b) A balanced application becomes comm. bound. c) A computation bound application stays as it was. d) A computation bound application becomes balanced.

In order to find the most appropriate configuration of the message logging approach, we should analyze how the parallel application and the logging approach coexist in the parallel machine. There will be two parallel applications that will compete for resources, thus it is critical to analyze the influence of FT in application behavior.

4 Analyzing Message Logging Processes Mapping

Most of the impact of a pessimistic receiver-based message logging protocol concentrates on communications and storage (memory or hard disks), but there is also a small impact on computations because FT tasks also need some CPU cycles in order to carry on their work.

For the analysis in this paper we have considered the pessimistic receiver-based message logging protocol used in RADIC. RADIC main components are shown on Figure 3, Protectors' main functions during the protection stage are: establish a heartbeat/watchdog mechanism between nodes (low CPU consumption operation, do not depend on application behavior) and to receive and manage message logs (CPU consumption depends on application) and checkpoints from observers (infrequent operation). All communications between processes go through Observers and each received message is sent to a corresponding logger thread (usually the protector of RADIC is drawn as an equilateral triangle, but in this case we have split it two right triangles to distinguish the main operations).

In order to reduce the impact of the pessimistic receiver-based logging protocol of RADIC we propose to save computational cores for the logger threads (Namely, threads that are in charge of receiving and logging messages of other processes), thus avoiding the competition for CPU between application processes

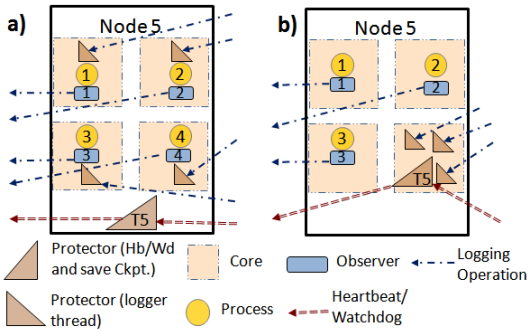


Fig. 3. RADIC Processes Mapping. a)Logging threads equitably distributed among cores. b) Protector with own resources.

and logger threads. According to this protocol every message should be logged in a different computing node (there are no dedicated nodes, but the usage of Spare Nodes is considered in [7]), then there is a considerable increase in the transmission time of each message when RADIC protection is activated. Thus, when executing a parallel application with RADIC the previous scenarios change (Figure 1), because the processes will be waiting a longer time for each message and the computation will be affected by the logger threads.

In order to reduce the overheads of the message logging approach, we analyzed how the application will be affected when introducing this logging technique. In Figure 2a we can observe how in a communication bound application, the difference between communications and computations becomes higher, and the overhead added in computations does not affect the iteration time. A balanced application without message logging will become communication bound when message logging is used (Figure 2b).

In these two scenarios the message logging overheads cannot be hidden, but when it comes to computation bound applications we can manage the mapping of the logger threads so as to distribute the overheads equally among the processes (Figure 2c). Alternatively, we can choose to save some computational cores in each node in order to avoid context switch between application processes and logger threads (Figure 2d).

Considering that many parallel applications are executed with balanced per-core workload, our default proposal is to distribute the overhead in computation produced by the logger threads among application processes residing in a node (Figure 3a). Moreover, we characterize the parallel application in order to find the computation-communication ratio, and if the application is computation bound, we analyze the overheads produced in computations. If these overheads make the application behave as in Figure 2c, we propose saving cores in each node in order to assign them to the logger threads, obtaining the behavior showed in Figure 2d. Figure 3b shows how we assign the logger threads and other proctectors' functionalities when using own resources for them.

As saving cores may make the initial mapping change, we also analyze if the new mapping does not negatively affect the execution, resulting in a worse performance than the default option.

Another important aspect that we analyze is the per-process memory consumption. This is significant because we have the option of storing the message log in memory instead of hard disks as this allows us to avoid bigger delays when storing messages. When we put less processes per node, we can save more memory for the message log, thus there is more time to overlap the flush-to-disk operation with receptions of new messages to log. Also, we can use longer checkpoint intervals if we consider an event-triggered checkpoint mechanism where a full message log buffer triggers a checkpoint.

5 Experimental Validation

The main approach presented in this paper focus on resource assignation to decrease logging overheads and save memory for FT tasks. In this section we present experimental evaluation that has been carried out in order to probe our hypothesis.

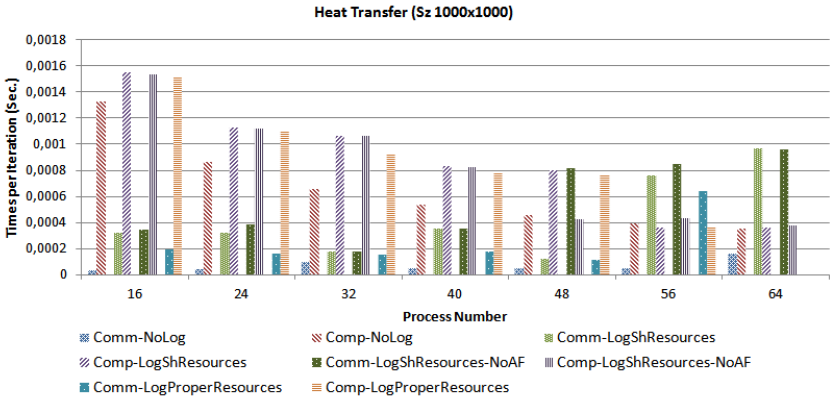
The experiments and characterizations have been made using a Dell PowerEdge M600 with 8 nodes, each node with 2 quad-core Intel® Xeon® E5430 running at 2.66 GHz. Each node has 16 GB of main memory and a dual embedded Broadcom® NetXtreme IITM 5708 Gigabit Ethernet. RADIC features have been integrated into Open MPI 1.7.

Most of the overhead added by a logging protocol affects communications. In order to lower the impact of a message logging technique we can assign more work per process which allow us to hide the overheads in communications (Figure 2c). However, if there are no available computational resources for the fault tolerance tasks, the overheads in computations could become relevant. Moreover, if we are executing a parallel application where memory consumption per process is high, there will be no room for the FT mechanisms.

When executing a parallel application with FT support is desirable to store checkpoints and message logs in main memory avoiding the file system, thus allowing FT mechanisms to execute faster. Also, if we consider an event triggered checkpoint mechanism where checkpoints take place when a message-log-buffer in memory is full and we save memory by executing less application processes per node we can use a bigger message-log-buffer, thus the checkpoint interval could be bigger.

Our testbed here is composed by two SPMD applications: a Heat Transfer application and a Laplace Solver. Both applications allow overlapping between the computation steps and the communication steps as was shown in Figure 2 and are written using non-blocking communications. The computation and communication times per iteration showed in bars in Figure 4 and Figure 5 are obtained by executing a few iterations of the parallel applications observing all processes and then selecting the slowest one for each number of processes. The execution times have been obtained using 10000 iterations.

In this experiments we have only considered the overlapped times (communication and computations) because they represent the higher portion of both applications. We have discarded the delays caused by desynchronization and



Overheads With Barriers			
# Process	Ovh-ShResources-NoAF	Ovh-ShResources	Ovh-ProperResources
16	55,25	51,75	46,55
24	69,42	61,96	56,94
32	44,06	41,24	28,94
40	116,03	86,14	71,52
48	123,38	96,99	75,58
56	129,86	99,10	75,40
64	54,58	43,28	

Overheads Without Barriers			
# Process	Ovh-ShResources-NoAF	Ovh-ShResources	Ovh-ProperResources
16	35,85	35,00	36,57
24	56,07	48,34	45,47
32	69,48	59,40	53,98
40	85,41	78,13	66,55
48	93,09	78,62	65,14
56	103,10	91,09	65,97
64	180,31	145,58	

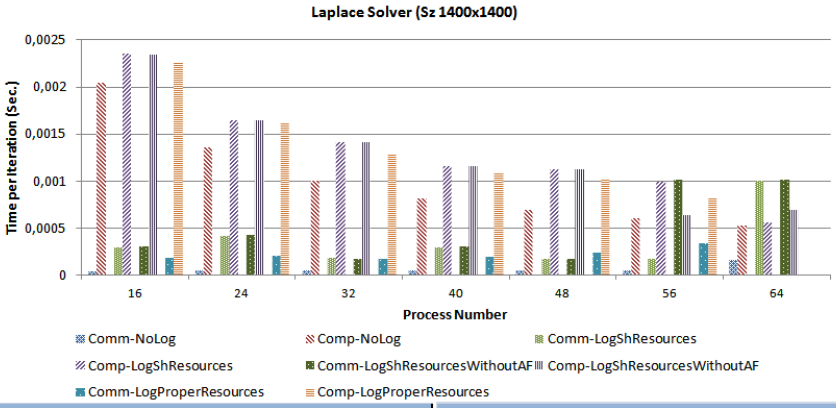
Fig. 4. Characterization results and Overhead Analysis of the Heat Transfer Application.

the computation time spent in computing the edges of each sub-matrix before sending it to the neighbors.

For both applications we have measure communication and computation times with the following options:

1. Without using message logging (Comm-NoLog and Comp-NoLog).
2. With message logging using all available cores in each node and giving affinity to each logger thread in order to ensure an equally distributed overhead in computation among all application processes (Comm-LogShResources and Comp-LogShResources).
3. With message logging using all available cores in each node without giving affinity to each logger thread (Comm-LogShResources-NoAF and Comp-LogShResources-NoAF).
4. With message logging saving one core per node and assigning all logger threads to the core not used by application processes (Comm-LogProperResources and Comp-LogProperResources).

With the purpose of measuring the communication and computation times of each application, we have inserted a barrier (MPI_Barrier) that allow us to properly measure them. The tables of Figure 4 and Figure 5 show the overhead in percentage introduced by each message logging approach with the barriers and also without them. The executions without barriers are faster than the execution with barriers and we present both overheads in order to prove that the measures taken are consistent when removing them and executing the original versions.



Overheads With Barriers				Overheads Without Barriers			
# Process	Ovh-ShResources-NoAF	Ovh-ShResources	Ovh-ProperResources	# Process	Ovh-ShResources-NoAF	Ovh-ShResources	Ovh-ProperResources
16	39,91	38,79	38,25	16	24,24	24,41	26,25
24	55,79	50,99	50,02	24	40,31	39,60	35,52
32	56,33	37,68	25,74	32	50,28	47,66	42,94
40	75,11	65,36	51,97	40	56,07	57,63	44,82
48	89,70	79,68	65,60	48	80,68	69,55	58,41
56	101,65	82,75	61,83	56	83,45	73,34	52,41
64	87,01	74,10		64	179,03	138,30	

Fig. 5. Characterization results and Overhead Analysis of the Laplace Solver.

In Figure 4 we can observe how the computation times when using the version with own resources is lower. Even when the application becomes communication bound (56 processes) the logging version with own resources behaves better than the other versions. We do not show results of the version with own resources with 64 processes because our test environment has 64 cores, and we have to save 8 cores (1 per node) for the logger threads.

The tables of Figure 4 reflect what we have observed when characterizing the application, using message logging with own resources for the logger threads introduces less overhead in almost all cases (except with 16 cores without barriers). At best, we have reduced 25% overhead when comparing the own resources version with the version with shared resources and affinity. We can also observe that when increasing the number of processes without increasing the problem size, the overhead added becomes bigger.

Figure 5 shows the execution of the Laplace Solver. As was in the previous experiment, here we can observer how the computation times are lower when using the version with own resources.

The tables of Figure 5 reflect again what we have observed when characterizing the application, using message logging with own resources for the logger threads introduces less overhead in almost all cases (except with 16 cores without barriers).At best, we have reduced 20% overhead when comparing the own resources version with the version with shared resources and affinity.

As we have observed, in both applications the computation time of the versions with FT with own resources is lower than the versions with shared resources, but is not equal to the version without message logging. This is because

when logging is activated and a process call MPI_Irecv, this process should save the request, re-send the message to its logger thread and free the request when the message was totally received, thus there is an slight increase in computation.

6 Conclusions

The main contribution of this paper consists on analyzing possible configurations of the pessimistic receiver-based logging approach in order to find the most suitable according to application behavior. This is done by characterizing the parallel application (or a small kernel of it) obtaining the computation and communication times and the disturbance caused by the logging approaches. Our initial results have demonstrated that saving resources for the FT tasks reduces overheads and also allows us to save memory for a message log buffer. In our experimental validation we have obtained 25% overhead reduction at best.

Future work will extend the analysis made in this paper to a bigger set of applications. We will focus on obtaining traces of parallel applications and use them to find the FT configuration that will be more suitable to them. We will also analyze the relationship between message sizes and logging overheads, in order to determine the number of resources that should be save for FT tasks, because with bigger message sizes delays could increase.

References

1. Nielsen, I., Janssen, C.L.: Multicore challenges and benefits for high performance scientific computing. Sci. Program. (2008) 277–285
2. Bouteiller, A., Herault, T., Bosilca, G., Dongarra, J.J.: Correlated set coordination in fault tolerant message logging protocols. (2011) 51–64
3. Bouteiller, A., Ropars, T., Bosilca, G., Morin, C., Dongarra, J.: Reasons for a Pessimistic or Optimistic Message Logging Protocol in MPI Uncoordinated Failure Recovery. (2009) 229–236
4. Bouteiller, A., Bosilca, G., Dongarra, J.: Redesigning the message logging model for high performance. Concurr. Comput. : Pract. Exper. (2010) 2196–2211
5. Fialho, L., Rexachs, D., Luque, E.: What is missing in current checkpoint interval models? 2012 IEEE 32nd International Conference on Distributed Computing Systems (2011) 322–332
6. Lemarinier, P., Bouteiller, A., Herault, T., Krawezik, G., Cappello, F.: Improved message logging versus improved coordinated checkpointing for fault tolerant mpi. 2012 IEEE International Conference on Cluster Computing (2004) 115–124
7. Meyer, H., Rexachs, D., Luque, E.: Radic: A fault tolerant middleware with automatic management of spare nodes. The 2012 International Conference on Parallel and Distributed Processing Techniques and Applications, July 16-19, Las Vegas, USA (2012) 17–23
8. Santos, G., Fialho, L., Rexachs, D., Luque, E.: Increasing the availability provided by radic with low overhead. IEEE International Conference on Cluster Computing and Workshops, 2009. CLUSTER '09. (2009) 1–8
9. Hursey, J., Squyres, J., Dontje, T.: Locality-aware parallel process mapping for multi-core hpc systems. IEEE International Conference on Cluster Computing (2011) 527–531