The Performance of Coordinated and Independent Checkpointing

Luis Moura Silva João Gabriel Silva

Departamento Engenharia Informática
Universidade de Coimbra, Polo II
P-3030 - Coimbra
PORTUGAL
Email: luis@dei.uc.pt

Abstract

Checkpointing is a very effective technique to tolerate the occurrence of failures in distributed and parallel applications. The existing algorithms in the literature are basically divided into two main classes: coordinated and independent checkpointing.

This paper presents an experimental study that compares the performance of these two classes of algorithms. The main conclusion of our study is that coordinated checkpointing is more efficient than independent checkpointing and all the arguments against the performance of coordinated algorithms were not verified in practice.

1. Introduction

Checkpointing and rollback-recovery is a very effective technique for recovering the application from partial or total system failures. There are several papers in the literature about checkpointing algorithms for distributed and parallel systems. A comprehensive survey is presented in [1]. Basically, the checkpointing algorithms can be divided in two main classes: independent and coordinated checkpointing.

In the former class of algorithms the application processes are allowed to establish checkpoints in an independent way and no synchronization is enforced between their checkpoint operations. When there is a failure the system will search in stable storage and will try to find some set of local checkpoints that taken together correspond to a consistent state of the application. This requires each process to keep several checkpoints in stable storage, and there is no certainty that a global consistent state can be built. This approach presents two main advantages: (i) first, there is no need to exchange any protocol messages during a checkpoint operation; (ii) if the execution of the processes is completely asynchronous the system can checkpoint one process at a time and this may reduce the bandwidth produced by the checkpoint I/O data. The main disadvantages of this approach are the possibility of occurrence of the *domino-effect* [2] during the rollback operation; and the storage overhead, since several checkpoints have to be maintained in stable storage.

Some algorithms that follow this approach were presented in the literature [3-6] and all of them are prone to the problem of *domino-effect*. The probability of occurrence of the *domino-effect* can be reduced if checkpoints are taken very often, but this will increase the failure-free time overhead. Another way to avoid this problem is to use an additional message-logging scheme that can be based on pessimistic [7] or optimistic logging [8].

In the second approach of coordinated checkpointing, a global checkpoint is taken periodically and the processes have to synchronize between themselves to assure that their set of local checkpoints corresponds to a consistent state of the application [9]. A consistent state is achieved during run-time, while in the independent schemes the determination of a consistent recovery line was left to the recovery phase, which could result in some rollback propagation. In coordinated checkpointing the recovery phase is simple and quite predictable since all the processes roll back to their last committed checkpoints. This approach avoids the *domino-effect* and presents a lower overhead in stable storage. Some overhead is introduced during the checkpoint operation due to the synchronization of

processes and some people argue that independent checkpointing is much better than coordinated checkpointing because of the overhead that is caused by this synchronization.

In this paper, we present the results of an experimental study that was conducted in a commercial parallel machine. We have done an implementation of coordinated and independent checkpointing and the goal was to compare the performance penalty incurred by each scheme.

2. Performance Results

Both schemes of checkpointing were implemented in a run-time library (`CHK-LIB`) that we have developed for the Parix operating system[1] [10]. That library provides a reliable and FIFO communication and a MPI-like programming interface.

The testbed machine was a Parsytec Xplorer, with 8 transputers (T805). In that machine, each processor has 4 Mbytes of main-memory, and all of them have direct access to the host file system. In our particular case, the host machine was a SunSparc2.

2.1 The Application Benchmarks

To evaluate the checkpointing schemes we have used the following application benchmarks:

- **ISING**: This program simulates the behaviour of Spin-glasses.
- **SOR**: successive overrelaxation is an iterative method to solve Laplace's equation on a regular grid.
- **ASP**: solves the All-pairs Shortest Paths problem i.e. it finds the length of the shortest path from any node i to any other node j in a given graph with N nodes by using Floyd's algorithm.
- **NBODY**: this program simulates the evolution of a system of bodies under the influence of gravitational forces.
- **GAUSS**: solves a system of linear equations using the method of Gauss-elimination.
- **TSP**: solves the travelling salesman problem for a dense map of 16 cities.
- **NQUEENS**: counts the number of solutions to the N-queens problem.

[1] Parix is a product of Parsytec Computer GmbH.

2.2 Coordinated Checkpointing

The algorithm of coordinated checkpointing that was selected is described in [11]. We omit its explanation here for lack of space.

We decided to implement four different variations of the coordinated checkpointing algorithm, namely:

- **`Coord_NB`**: this scheme uses a non-blocking checkpointing protocol. While the state of a process is being saved to stable storage the process remains blocked. It resumes at the end of that operation but does not need to wait for the rest of the protocol that will be executed in the background.
- **`Coord_NBM`**: it uses a non-blocking checkpointing protocol but the checkpoint of each process is firstly saved to a local buffer in main memory. When this memory copy operation is over the application process is allowed to resume with its computation. The checkpointer thread will be responsible for writing the checkpoint data into the stable storage and for the rest of the protocol. These operations are done in a concurrent way.
- **`Coord_NBMS`**: it is an extension of the previous scheme (`Coord_NBM`). The only difference is that it uses a checkpointing staggering technique in order to reduce the contention in the access to the stable storage. The processors are organized in a virtual ring and execute a token-based protocol: a process that has the token is allowed to write the checkpoint to stable storage. When that operation is finished it sends the token to the next one that will write its checkpoint. The operation continues until the last processor writes its checkpoint. At this point, the checkpoint coordinator can initiate the commit phase. With this token-based protocol, only one processor has access to stable storage at a time, thus reducing the potential bottleneck of using a global disk.

From our experimental study we concluded that the use of those two optimization techniques - *main-memory checkpointing* and *checkpoint staggering* - was very effective in the reduction of the performance overhead.

2.3 Independent Checkpointing

In this approach, each process takes its checkpoint at its own will and there is no synchronization involved among the processes. While in the previous approach there was only one process that had the freedom to initiate a global checkpoint, in this approach every processor is able to perform a checkpoint, but this is just a local operation.

The potential advantage of independent against coordinated checkpointing is the simplicity of the protocol since there is no need to exchange control messages during normal execution. However, this checkpointing approach is prone to the *domino-effect*. It remains to be seen if the performance of this approach is better or not than the coordinated checkpointing approach. Two different schemes were implemented:

- **`Indep`**: every process takes its checkpoint in an autonomous and independent way. There is no need to synchronize with the others. In each node of the system, while the checkpointer thread is saving the checkpoint data to stable storage the application process remains blocked, and is only resumed when that operation is finished.
- **`Indep_M`**: this is similar to the previous scheme but this one uses a main-memory checkpointing technique.

2.4 Experimental Results

In Table 1 we present the overhead per checkpoint when using both schemes of independent checkpointing, and we compare with three of the coordinated checkpointing schemes.

It is fair to compare `Coord_NB` with `Indep` and `Coord_NBM/Coord_NBMS` with `Indep_M`. These three latter schemes make use of main-memory checkpointing while the former ones do not. We can see in Table 1 that `Indep` did not perform better than `Coord_NB`. In 15 of the cases it performed worse, while in the remaining 6 cases it was better. This was an unexpected result.

Applications	Coord NB	Indep	Coord NBM	Indep M	Coord NBMS
ISING 256	6.102	6.923	2.549	3.123	0.639
ISING 512	7.363	8.265	2.354	2.133	0.814
ISING 768	9.985	10.392	2.901	2.373	0.788
ISING 1024	13.222	13.829	1.857	2.052	0.132
ISING 1280	17.304	18.401	2.898	1.339	0.253
ISING 1536	22.769	23.165	------	------	------
ISING 1792	28.694	26.982	------	------	------
ISING 2048	35.345	31.847	------	------	------
SOR 256	6.912	7.600	3.507	4.242	1.099
SOR 512	10.396	10.939	5.151	4.404	2.427
SOR 768	16.453	17.569	7.049	6.634	4.707
SOR 1024	24.841	25.681	------	------	------
SOR 1280	36.065	37.625	------	------	------
GAUSS 512	10.475	10.714	5.429	3.526	1.143
GAUSS 1024	22.050	21.305	10.478	6.086	1.915
ASP 512	7.809	8.171	3.247	2.449	0.916
ASP 1024	14.005	13.503	4.155	2.533	1.513
NBODY	4.396	5.635	0.985	0.201	0.270
TSP	0.976	0.942	0.233	0.222	0.242
NQUEENS	0.795	0.725	0.145	0.145	0.138

Table 1: Performance of independent versus coordinated checkpointing.

Curiously, the opposite scenario was achieved with `Coord_NBM` and `Indep_M`: when comparing these two schemes we can see that the independent algorithm performed better in 12 of the cases, while in just 3 cases the performance of `Coord_NBM` was better than `Indep_M`.

From these two comparisons we can take two main conclusions: first of all, the overhead for synchronizing the processes in a coordinated checkpoint is not a relevant factor. It is clear that the major contribution to the overhead is the checkpoint saving operation. Secondly, when we do not use a technique like main-memory checkpointing it seems advantageous to interrupt the application with a global coordinated checkpoint scheme, rather than interrupting each processor in an autonomous way. However, when the checkpoint of each process is taken in a concurrent way the independent checkpointing scheme is fairly better. In this case, the contention in the global disk is lower and the autonomous saving of the checkpoints would make the difference.

Finally, if we take a look at the two last columns we can see that `Coord_NBMS` performs much better than `Indep_M`. This result means that we can achieve better performance with coordinated checkpointing in both cases. While the `Coord_NBMS` scheme made use of two optimization techniques (i.e. main-memory and checkpoint staggering) the `Indep_M` scheme only used main-memory checkpointing.

The next two Tables show more clearly the comparison of performance between coordinated and independent checkpointing. Table 2 presents the execution time of those applications (SOR and ISING were executed for 100 iterations, while NBODY simulated 10 iterations). The first column displays the normal execution time, i.e. the time without checkpointing. The other columns present the execution of those four checkpointing schemes (`Coord_NB` vs `Indep`; `Coord_NBMS` vs `Indep_M`). All the applications were checkpointed 3 times during their execution.

The interval between checkpoints was different for every application. It is presented (in seconds) in the first column of the Table. It changed from 1 up to 7 minutes, depending on the normal execution of the application. In this Table we present the performance overhead of

those checkpointing schemes when the number of checkpoints was 3.

	NORMAL	COORD NB	INDEP	COORD NBMS	INDEP M
ISING 1024	314.660	354.328	356.148	318.058	320.818
ISING 2048	1240.934	1346.969	1336.475	-------	-------
SOR 768	176.404	225.764	229.113	190.525	196.308
SOR 1280	479.873	588.069	592.750	-------	-------
GAUSS 1024	311.357	377.508	375.273	317.103	329.615
ASP 1024	987.454	1029.469	1027.964	991.993	995.053
NBODY 4000	643.094	656.283	660.000	643.906	643.698
TSP	352.461	355.389	355.228	353.187	353.129
NQUEENS	586.250	588.637	588.425	586.665	586.687

Table 2: Execution times of the checkpointing schemes.

Table 3 clearly shows that in the overall both coordinated checkpointing schemes perform better than their independent checkpointing counterparts, although the difference is not very significant. We were expecting that independent checkpointing could perform better than coordinated checkpointing because each process is able to write its checkpoint to stable storage with little interference from other processes. However, the performance results did not follow our expectations.

	INTERVAL CHKP (SEC)	COORD NB	INDEP	COORD NBMS	INDEP M
TSP	117	0.83 %	0.78 %	0.20 %	0.18 %
NQUEENS	195	0.40 %	0.37 %	0.07 %	0.07 %
SOR 768	59	27.98 %	29.88 %	8.00 %	11.28 %
ISING 1024	105	12.60 %	13.18 %	1.08 %	1.95 %
GAUSS 1024	104	21.24 %	20.52 %	1.84 %	5.86 %
ASP 1024	329	4.25 %	4.10 %	0.46 %	0.77 %
SOR 1280	160	22.54 %	23.52 %	-------	-------
ISING 2048	414	8.54 %	7.70 %	-------	-------
NBODY 4000	214	2.05 %	2.62 %	0.12 %	0.09 %

Table 3: Performance overhead of the checkpointing schemes.

In the first coordinated checkpointing scheme -`Coord_NB`- all the processes attempt to write their checkpoints at essentially the same time, increasing the load on the stable storage server and slowing its response. It is now clear that the main overhead of coordinated checkpointing is not the process synchronization, but is due to the nearly simultaneous occurrence of all checkpoints, which is likely to result in contention for the communication network and the stable storage.

In the other scheme -`Coord_NBMS`- we used a checkpoint staggering technique to reduce the contention on stable storage. In Table 3 we observe a reduction factor of 4 up to 17 in the performance overhead. However, checkpoint staggering was only an effective solution when used together with the other optimization technique: main-memory checkpointing.

To summarize the results we have obtained, we could say that coordinated checkpointing leads to a better failure-free performance, requires less space in stable storage, is more easy to implement and obtains a predictable rollback distance. The major contribution to the overhead in this approach is due to the checkpoint saving, while the cost of synchronization is actually insignificant. In our experiments we did not observe any significant performance benefit from using independent checkpointing. Besides, this approach implies a large storage overhead and is prone to the *domino-effect*. Several checkpoints have to be kept in stable storage, even if the recovery system makes use of some garbage collection algorithm to discard useless checkpoints [12]. With these two advantages, coordinated checkpointing is definitely the best choice.

3. Related Work

Some implementation results of checkpointing and rollback-recovery were also reported in [13]. Some optimization techniques, like incremental and copy-on-write checkpointing, were used to reduce the performance overhead. That study has shown that the synchronization of the individual processes to form a consistent global checkpoint added little overhead. The overhead of writing data to stable storage clearly dominates. It was shown that the performance overhead of consistent checkpointing is about the same as the overhead of independent checkpointing.

Other experimental results for coordinated checkpointing were presented in [14-17]. However, in all these studies no comparison was presented between the performance of coordinated and independent checkpointing.

4. Conclusions

The main conclusion from our experimental study is that a non-blocking coordinated checkpointing scheme is definitely better than independent checkpointing. It is *domino-effect* free, introduces a minimal storage overhead and in most of our applications it presented a lower performance overhead. The cost of checkpointing is actually dominated by the cost of writing the local checkpoints to stable

storage. The overhead of synchronizing the checkpoints is negligible and presents a minor contribution to the overall performance cost.

The use of optimization techniques like main-memory checkpointing and checkpoint staggering was able to reduce considerably the performance overhead of our coordinated algorithm. Whenever possible the system should try to perform the checkpoint concurrently with the computation, and use an ordering technique to reduce the congestion in stable storage. More details about this study can be obtained in [18].

Acknowledgments

This work was partially supported by the Portuguese *Ministério da Ciência e Tecnologia*, the European Union through the R&D Unit 326/94 (CISUC) and the project PRAXIS XXI 2/2.1/TIT/1625/95 (PARQUANTUM).

References

[1] E.N.Elnozahy, D.B.Johnson, Y.M.Wang. "A Survey of Rollback-Recovery Protocols in Message Passing Systems", Technical Report CMU-CS-96-181, School of Computer Science, Carnegie Mellon University, October 1996

[2] B.Randell. "System Structure for Software Fault-Tolerance", IEEE Trans. on Software on Software Engineering, Vol. SE-1 (2), pp. 226-232, June 1975

[3] B.Bhargava, S.R.Lian. "Independent Checkpointing and Concurrent Rollback for Recovery in Distributed Systems", Proc. 7th Symposium on Reliable Distributed Systems, pp. 3-12, 1988

[4] W.G.Wood. "A Decentralised Recovery Control Protocol", Proc. 11h Int. Fault-Tolerant Computing Symposium, FTCS-11, pp. 159-164, 1981

[5] P.M.Merlin, B.Randell. "Consistent State Restoration in Distributed Systems", Proc. 8th Int. Fault-Tolerant Computing Symposium, FTCS-8, pp. 129-134, July 1978

[6] J.A.McDermid. "Checkpointing and Error Recovery in Distributed Systems", Proc. 2nd Int. Conf. on Distributed Computing Systems, pp. 271-282, 1981

[7] A.Borg, J.Baumback, S.Glazer. "A Message System Supporting Fault-Tolerance", Proc. 9th Symposium on Operating Systems Principles, pp. 90-99, October 1983

[8] R.E.Strom, S.A.Yemini. "Optimistic Recovery: an Asynchronous Approach to Fault-Tolerance in Distributed Systems", Proc. 14th Int. Fault-Tolerant Computing Symposium, FTCS-14, pp. 374-379, 1984

[9] K.M.Chandy, L.Lamport. "Distributed Snapshots: Determining Global States of Distributed Systems", ACM Transactions on Computer Systems, Vol. 3, No. 1, pp. 63-75, February 1985

[10] "Parix 1.2: Software Documentation", Parsytec Computer GmbH, March 1993

[11] L.M.Silva, J.G.Silva. "Global Checkpointing for Distributed Programs", Proc. 11th Symposium on Reliable Distributed Programs, Houston USA, pp. 155-162, October 1992

[12] Y.M.Wang, P.Y.Chung, I.J.Lin, W.K.Fuchs. "Checkpoint Space Reclamation for Uncoordinated Checkpointing in Message-Passing Systems", IEEE Trans. on Parallel and Distributed Systems, Vol. 6, No. 5, May 1995

[13] E.N.Elnozahy, D.B.Johnson, W.Zwaenepoel. "The Performance of Consistent Checkpointing", Proc. 11th Symposium on Reliable Distributed Systems, pp. 39-47, 1992

[14] M.F.Kaashoek, R.Michiels, H.Bal, A.Tanenbaum. "Transparent Fault-Tolerance in Parallel Orca Programs", Symposium on Experiences with Distributed and Multiprocessor Systems III, pp. 297-312, March 1992

[15] K.Li, J.F.Naughton, J.S.Plank. "Real-Time Concurrent Checkpoint for Parallel Programs", Proc. 2nd ACM Sigplan Symposium in Principles and Practice of Parallel Programming, pp. 79-88, March 1990

[16] G.Muller, M.Hue, N.Peyrouze. "Performance of Consistent Checkpointing in a Modular Operating System: Results of the FTM Experiment", Lecture Notes in Computer Science, Vol. 852, Proc. European Dependable Computing Conference, EDCC-1, Springer-Verlag, pp. 491-508, October 1994

[17] J.S.Plank, K.Li. "ickp - A Consistent Checkpointer for Multicomputers", IEEE Parallel and Distributed Technology, vol. 2 (2), pp. 62-67, Summer 1994

[18] L.M.Silva, "Checkpointing Mechanisms for Scientific Parallel Applications", PhD Thesis presented at the Univ. of Coimbra, Portugal, January 1997, ISBN 972-97189-0-3