X Simpósio Brasileiro de Arquitetura de Computadores

Performance Evaluation of Checkpointing and Rollback-Recovery Algorithms for Distributed Systems

Sérgio Luis Cechin Ingrid Jansch-Pôrto
{cechin, ingrid}@inf.ufrgs.br

Curso de Pós-Graduação em Ciência da Computação
Instituto de Informática - UFRGS - Caixa Postal 15064
91501-970 Porto Alegre - Brasil

Keywords: Fault Tolerance, Distributed Systems, Rollback Recovery, Synchronous and Asynchronous Checkpointing, Performance Evaluation.

Abstract

In distributed systems, backward recovery has the synchronous and asynchronous approaches as the two main implementation paradigms. In this paper we compare two representative algorithms on these groups and present some theoretical results. Koo & Toueg synchronous algorithm and Juang & Venkatesan asynchronous algorithm have been chosen for this purpose. Our goal is to demonstrate that the advantages and disadvantages between them are mainly related to the characteristics of the applications.

1 Introduction

The difficulties related to process recovery in distributed systems are mainly related to system characteristics: the distributed system consists of autonomous nodes, geographically at different locations, that are connected to each other by a communication network. Each one of these nodes consists of a processor, some private memory which is inaccessible to all other processors, and a private clock. The nodes are loosely coupled, do not have shared memory, and communicate via message passing [JAL94]. The literature on recovery includes several papers with detailed description of different algorithms for checkpointing and rollback recovery algorithms proposed in the scope of distributed systems. This paper proposes the comparison of two representative algorithms of the synchronous and asynchronous approaches from a theoretical model proposed by the author [CEC98]; our goal is to verify the advantages and the disadvantages that are conceptually reported in the literature as support to the alternative proposals and extensions of algorithms.

The algorithms have been proposed to systems whose behavior is based in the following:

- the communication channels are assumed to have infinite buffers, to be error-free, and to deliver messages in the order sent;
- under fault occurrence, processes can fail by stopping (*fail-stop* mode). All other processes are informed of the failure in finite time. Partitions of the communication network do not occur;
- it is admitted the occurrence of orphan or lost messages.

We suppose that the faults will be of transient nature, to eliminate the need of fault treatment and the consequent overhead it would cause. The total time of each message, since it is sent by a process until it is received by the destination process, is t_m. For analysis, t_m have been split up in three phases: packing parameters and transmission, t_{mp}; time period to pass through communication channel, t_{mt}; and unpacking results, t_{mu}.

2 Characteristics of the synchronous algorithm

Koo & Toueg [KOO87] proposed checkpointing and recovery algorithms to restart the system from a consistent state when failures occur. This assumption prevents the *domino effect* as well as livelock problems associated with rollback-recovery. The processes coordinate their checkpointing actions in order to save a consistent set of checkpoints, that is, they set up a consistent global state. During this action of establishing checkpoints, only control messages may be sent through the channel. Consequently, each process stores at most two checkpoints in stable storage, and previous checkpoints may be discarded.

Under recovery, whenever a process rolls back to its checkpoint, it notifies all other processes also to roll back to their respective checkpoints. This action is coordinated to avoid the introduction of livelocks, i.e., situations in which a single failure can cause an infinite number of rollbacks, preventing the systems from making progress [KOO87]. Once synchronized, each process installs its checkpoint state and resumes execution.

The algorithms are based on *two-phase commit protocols*. Checkpointing starts by the invocation of the algorithm by a single process, may be aborted under several conditions and the action is postponed. The rollback recovery algorithm assumes also that a single process invokes the algorithm, but in case of failure, it stays blocked until all processes may roll back. It may be noticed that resuming execution form a consistent state implies discarding all computation that had been executed since the last checkpoint was taken. This fact implies a penalty in performance parameters.

Taking checkpoints is related to parameters of system fault occurrence. An inadequate choice of checkpoints frequency will be related to system performance. Consequently, low frequency (a few checkpoints) will probably result in a big amount of computation undone during a roll back after fault occurrence. If failures often occur between successive checkpoints, this will imply a low frequency checkpointing, which will cause low performance; in other way, checkpoints taken frequently will make the system spend much time in the coordination process, which reduces significantly the performance, even during normal operation (fault-free operation).

3 Characteristics of the asynchronous algorithm

Under the asynchronous approach, checkpoints for each process are taken independently without any synchronization among the processes. We have chosen the Juang e Venkatesan [JUA91] algorithm, which has some similarities to the synchronous approach, allowing easy comparison. Having received a message, the process makes the related computation, changes its state and sends out a set of messages to other processes. This activity of a process is treated as one event: after each event, the process records in its volatile log a triple (with the state of the process before the event, the incoming message that triggered the event, and the set of messages that the process sent in this event). From time to time, a process independently transfer the contents of its volatile log to the stable storage, thereby creating a new checkpoint. The new checkpoint does not destroy the earlier ones. Hence, at any given time, conceptually, each process has a record of its complete behavior from the beginning to its latest checkpoint.[JAL94]. The establishment of these local checkpoints avoids synchronization delays and additional messages overhead during normal operation.

Because of the absence of synchronization, there is no guarantee that a set of local checkpoints taken will be a consistent set of checkpoints. Thus, the recovery algorithm has to search for the most recent consistent set of checkpoints before it can initiate recovery [SIN94]. The volatile log does not persist after the failure of the process: the state of the failed process will be set from the latest event logged in the stable storage; the others may start from their volatile logs. The major disadvantage of this approach is that the *domino effect* is possible when attempting to reach a consistent state; however, it is limited to the last checkpoint stored in stable storage for the failed process. Anyway, as saving actions are not communicated to other processes, all the previous checkpoints taken from the beginning of the application have to be kept in stable storage.

Singhal & Shivaratri [SIN94] stand clearly the key points to performance analysis, which are given in the following: "While synchronous checkpointing simplifies recovery (because a consistent set of checkpoints is readily available), it has the following disadvantages: additional messages are exchanged by the algorithm when it takes a checkpoint; synchronization delays are introduced during normal operations. Then, if failures rarely occur between successive checkpoints, then the synchronous approach places unnecessary burden on the system in the form of additional messages, delays and processing overhead." Under the asynchronous approach, neither additional messages nor synchronization delays disturb the system while there is fault-free operation, the action is restricted to the checkpoint saving. However, this situation may change completely under fault occurrence.

4 General principles and basic suppositions

Some restrictions had been done to simplify the proposed performance analysis; others were necessary to make a consistent comparison of operation parameters concerning both algorithms. Detailed description of these assumptions is presented in [CEC98] and it was omitted here for sake of space:

- the recursive mechanism proposed by Koo & Toueg for establishing and rolling back to checkpoints have not been considered (a broadcast principle is considered);
- by hypotheses, failures have periodical occurrence; the mean period is designated *TBF* (*Time Between Failures*);

- all active processes are considered by the fault-tolerant procedure, even those who have not exchanged messages with the faulty ones;
- although Koo & Toueg algorithm tolerate failures that occur during its execution, we assume that faults do not occur during algorithms execution (checkpointing and rollback recovery).

Performance computation will give the relative amount of time (%) used for application processing considered the total time spent among application and fault-tolerant actions. The equation is given by:

$$Performance = \frac{(TotalTime - FaulToleranceTotalTime)}{TotalTime}$$

As seen before, by hypothesis, failures have a periodical occurrence; the mean value of performance taken from the total time will be equivalent to the one calculated considering as the basic time a period between failures. That is, the expectation of the relative performance (RP) will be:

$$E(RP) = E\left(\frac{(TBF - FaultToleranceTotalTime)}{TBF}\right) = 1 - \frac{E(FaultToleranceTotalTime)}{TBF}$$

The time spent with fault-tolerant activities has been split up in two phases: checkpointing and rollback to a previous checkpoint. Both depend on the considered algorithm.

5 Relative performance of the synchronous algorithm

The equations presented here and in the chapters 6 were developed based on the algorithm's time spent to accomplish their purposes. These equation's development is depicted in the work [CEC98] and it was omitted here for sake of space.

We define T_{CP} as the checkpointing periodicity for each process (or for the coordinated processes set); T_{CE} is the time required to establish one checkpoint (*ckpt*), and T_{RB} is the time required for a rollback. Then, the following expression gives the relative performance of the synchronous algorithm:

$$E(RP) = \left(1 - \frac{E(T_{CE})}{Tcp}\right)\left(1 - \frac{E(T_{RB})}{TBF}\right)$$

This equation has two terms that contribute for the decline of performance: one is due to the time required to checkpointing procedures; the other is due to the time required for rolling back to a previous state (*ckpt*). The expected checkpointing time is given by the following expression:

$$E(T_{CE}) = T_{FIX} + \left[P_{CE} \times E\left(T_{VAR}^{OK}\right) + \left(1 - P_{CE}\right) \times E\left(T_{VAR}^{NOK}\right)\right],$$

where T_{FIX} is the time required to make a tentative of establishing any checkpoint, P_{PR} is the expectancy of taking a checkpoint, T_{VAR}^{OK} is the overhead for a successful checkpointing procedure while T_{VAR}^{NOK} is the overhead for an unsuccessful checkpointing procedure. These terms of the expected checkpointing time may be calculated by the following equations,

where N is the number of processes:

$T_{FIX} = 3 \cdot t_m + (2 \cdot N - 2) \cdot t_{mp}$, when there is no support for messages broadcast;

$T_{FIX} = 3 \cdot t_m + (3 \cdot N - 4) \cdot t_{mp}$, when there is support for messages broadcast;

$T_{VAR}^{OK} = T_{VAR}^{NOK} = T_{PCE}$, where T_{PCE} is the time required for storing the checkpoint data. In practice, T_{VAR}^{OK} e T_{VAR}^{NOK} are similar but not equal; the value of T_{VAR}^{OK} is bigger as it includes the activation of the checkpoint in the stable memory.

For the time spent in the rollback action, we have the following expression:

$$E(T_{RB}) = T_{CP} \times \left(\frac{2 - P_{CE}}{2 \times P_{CE}}\right) + T_{DET} + \frac{T_{FIX}}{P_{RB}} + \left[\left(\frac{1 - P_{RB}}{P_{RB}}\right) \times T_{VAR}^{NOK} + T_{VAR}^{OK}\right],$$

where P_{RB} is the expectancy of being successful in the rollback action; T_{DET} is the time delay between the failure and the event of error detection; T_{VAR}^{OK} is the overhead for a successful rollback recovery and T_{VAR}^{NOK} is the overhead for an unsuccessful rollback procedure (at least, one more will be necessary).

The computed results are shown by graphic representation. In figure 1 are plotted the curves of performance versus (a) the failure periodicity *TBF*, and (b) the checkpointing periodicity T_{CP}.

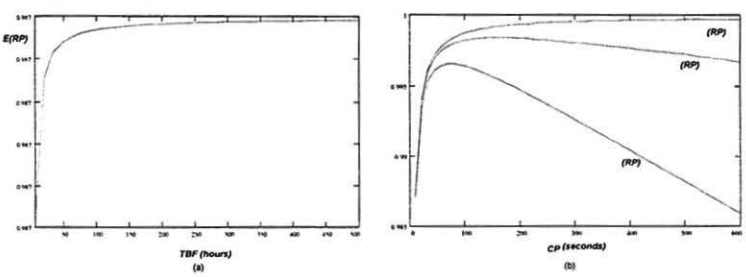

Figure 1 - Performance curves for the synchronous algorithm.

The performance exhibits an asymptotic increase with increasing *TBF*. The asymptote value may be calculated by the boundary value of the performance equation, with the increase of *TBF* to infinite:

$$\lim_{TBF \to \infty} E(RP) = 1 - \frac{E(T_{CE})}{T_{CP}}$$

In figure 1(b), there are three performance curves, each plotted for a different *TBF* value. The $E_1(DR)$ curve corresponds to a smaller *TBF* value while $E_3(DR)$ curve corresponds to a bigger *TBF* value. All the curves have a peak value, which may be computed when the differential of the performance equation is made equal to zero. As result, we have:

$$T_{CP_{max}} = \sqrt{\left[E(T_{CE}) \times (TBF - K2)\right] / K1}$$

$$\text{where } K1 = \left(\frac{2 - P_{CE}}{2 \times P_{CE}}\right) \text{ and } K2 = T_{DET} + \frac{T_{FIX}}{P_{RB}} + \left[\left(\frac{1 - P_{RB}}{P_{RB}}\right) \times T_{VAR}^{NOK} + T_{VAR}^{OK}\right].$$

6 Relative performance of the asynchronous algorithm

Similarly to the procedure related to performance equation of the synchronous algorithm, now for the asynchronous case we also have two main terms in the performance equation: one is due to the time required for checkpointing procedures and the other is due to the time required for rolling back to a previous state, according to the following expression:

$$E(RP) = \left(1 - \frac{T_{VCE}}{1/\lambda} - \frac{T_{PCE}}{T_{CP}}\right)\left(1 - \frac{E(T_{RB})}{TBF}\right),$$

where T_{VCE} is the time required to store the state of a process in the volatile storage, T_{PCE} is the time required to store the state of a process in the stable storage and λ is the mean number of received messages.

We may notice in the expression of the *relative performance* that the different activities that contribute to performance degradation appear in that equation divided by the periodicity they occur: the time required to write in the volatile storage is divided by the messages period; the time required to write in the stable storage is divided by checkpointing period; while the time required for rolling back to a checkpoint is divided by the failure periodicity. The time spent with storage actions in both volatile and stable memory depend on the hardware characteristics; for our purposes, these parameters will be considered as a mathematical relation between them.

The time required for recovery actions may be computed by the following expression:

$$E(T_{RB}) = \frac{T_{CP}}{2} + T_{DET} + T_{BRC} + N \times \left(T_{TM} + T_{PM}\right),$$

where T_{DET} is the time delay between the failure and the event of error detection; T_{BRC} is the time used for the faulty process to broadcast this occurrence to the other processes in the system; T_{TM} is the time spent during each iteration to search for the most recent consistent set of checkpoints; T_{PM} is the time required to process the messages during each iteration. As may be observed in the equation, N iterations are necessary to obtain consistent checkpoints. The expression to compute T_{TM} is the following:

$$T_{TM} = N \times \left[t_m + (N-2) \times t_{mp}\right]$$

In figure 2, are plotted the curves for the relative performance versus (a) the failure periodicity *TBF*, and (b) the checkpointing periodicity T_{CP}. The curves of figure 2(b) has been plotted for different values of *TBF*. The $E_1(RP)$ curve corresponds to a smaller *TBF* value while $E_3(RP)$ curve corresponds to a bigger *TBF* value.

The graphical form of the performance curves is basically the same obtained for the synchronous algorithm. The performance exhibits an asymptotic increase with increasing *TBF*. The asymptote value may be calculated by the boundary value of the performance equation, with the increase of *TBF* to infinite:

$$\lim_{TBF\to\infty} E(RP) = 1 - \frac{T_{VCE}}{1/\lambda} - \frac{T_{PCE}}{T_{CP}}.$$

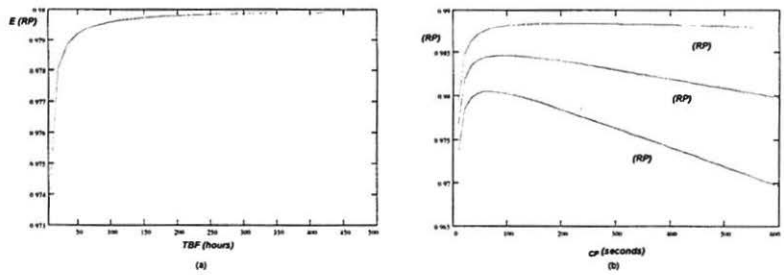

Figure 2 - Performance curves for the asynchronous algorithm

From these results, we may verify that even with a low failure rate (a big *TBF*), the performance will have a maximum value. This value depends on the time spent to store the checkpoints - both volatile and stable - and on the periodicity of these storage actions: that is, it depends on the incoming messages rate and on the transfer rate of these information to the stable memory.

The curve of performance versus T_{CP} have a peak value, which may be computed when the differential of the performance equation is made equal to zero. Then we have:

$$T_{CP_{max}} = \sqrt{\left[T_{PCE} \times (TBF - K2)\right] \Big/ \left[K1 \times \left(1 - \frac{T_{VCE}}{1/\lambda}\right)\right]}$$

where $K1 = \frac{1}{2}$ and $K2 = T_{DET} + T_{BRC} + N \times \left(T_{TM} + T_{PM}\right)$.

7 Analysis and Conclusions

The performance equations for both algorithms are determined by the two main phases designated as: checkpointing factor (CE) and rollback factor (RB), with the following equations: $SP = SF_{CE} \cdot SF_{RB}$ for the synchronous algorithm and $AP = AF_{CE} \cdot AF_{RB}$ for the asynchronous one. The four component factors are as in the following:

$$SF_{CE} = 1 - \frac{NSF_{CE}}{T_{CP}}, SF_{RB} = 1 - \frac{NSF_{RB}}{TBF}, AF_{CE} = 1 - \frac{T_{VCE}}{1/\lambda} - \frac{T_{PCE}}{T_{CP}} \text{ and } AF_{RB} = 1 - \frac{NAF_{RB}}{TBF}.$$

with: $NSF_{CE} = S_1 N + S_2$, $NSF_{RB} = S_3 N + S_4$, $NAF_{RB} = A_2 N^3 + A_3 N^2 + A_4 N + A_5$

The terms A_i and S_i are independent of N and TBF. They are defined using communication (messages) time, volatile and stable checkpoint write time, probability of checkpointing or rolling back (used in synchronous algorithm model), fault detection time and periodicity of checkpointing: tentative checkpoints in synchronous algorithm and stable checkpoints in asynchronous algorithm.

Beyond the above, for our purposes, *TBF*, *Tcp*, T_{VCE} and T_{PCE} will be considered as independent of *N*.

In general, it is supposed that the increase of *TBF* (lower failure rate) encourage the use of asynchronous algorithms. However, it is a simplified qualitative point-of-view. It is necessary to explore what factors cooperate for this result and how they do that.

The increase of *TBF* does not affect the performance factor related to checkpointing: both SF_{CE} and AF_{CE} do not depend on *TBF*. Otherwise, the increase of *TBF* affects the performance factor related to the rollback but not with the same intensity in both algorithms. This behavior may be observed by determining how this factors change (through differentials) and comparing them. The differentials of AF_{RB} and SF_{RB} are given by:

$$\frac{d}{dTBF} SF_{RB} = \frac{NSF_{RB}}{TBF^2} \qquad \frac{d}{dTBF} AF_{RB} = \frac{NAF_{RB}}{TBF^2}$$

For all valid values of *TBF* and *N*,these differentials are positive values. Then, for having a significant effect of *TBF* on performance, the differential of factors should be greater. It is necessary that $fdd(N) = NAF_{RB} - NSF_{RB} > 0$. Replacing these factors by their equations, we have the following expression:

$$fdd(N) = A_2 N^3 + A_3 N^2 + (A_4 - S_3)N + (A_5 - S_4)$$

This equation result is positive for all valid values of N (i.e., for $N \geq 2$). Then, the fault-tolerance cost imposed by the asynchronous algorithm decreases faster than for the synchronous one.

In spite of the conclusion above and comparing the rollback costs for both algorithms, we may verify that the following relation always occur: $SF_{RB} > AF_{RB}$. It can be demonstrated by replacing the factors by their component factors. The result is $NAF_{RB} > NSF_{RB}$, that is true for all valid values of *N*, as shown above. Hence, the asynchronous rollback cost will be always greater than for the synchronous case. However, when *TBF* increases, the rollback cost rate for the asynchronous algorithm decreases faster than for the synchronous one, both spreading to 1 when *TBF* spreads to infinite.

Next point we will examine is when fault rate is low ($TBF \rightarrow \infty$), the fault-tolerance cost through recovery depends on the checkpointing activity.

When SF_{CE} is expanded, we obtain: $SF_{CE} = 1 - \dfrac{2 \cdot t_{mp} \cdot N + 3 \cdot t_m - 2 \cdot t_{mp} + T_{PCE}}{Tcp}$

However, while this equation was obtained using $t_{mp} > t_{mt}$ for all messages, checkpointing invitation messages sent to all processes by an initiator process may found an overloaded channel. In this case, $t_{mp} > t_{mt}$ will not be true, and the equation should be changed to consider this other situation, only for these messages. Using t_c as the time spent for these messages and considering a high overload degree, we obtain the following expression:

$$SF_{CE} = 1 - \frac{t_c \cdot (N-1) + T_{PCE}}{Tcp}.$$

This equation shows that, while the asynchronous factor does not depend on *N*, the synchronous algorithm depends linearly with *N*.

The equation obtained previously for SF_{CE} expresses that this factor decreases with *N* and its slope is t_c, while AF_{CE} is independent of *N*. Typical curves for these factors are presented in figure 3, where is plotted a curve for AF_{CE} and three curves for SF_{CE} with

different values to t_c. It is important to observe the slope of *SP(N)*: with low values, the difference among the performances considered for both algorithms will be significant only for big values of *N*. The explanation is in the following. Consider the difference between *AP(N)* e *SP(N)*:

$$DIF = AP(N) - SP(N) = \left(\frac{t_c}{Tcp}\right) \cdot N - \left(\frac{t_c}{Tcp} + \frac{\lambda \cdot T_{PCE}}{R}\right).$$

For a non-overloaded network ($t_c = 19\mu s$), the equation is: $DIF \cong (19 \times 10^{-7}) \cdot N - (10^{-2})$. In this case, for a 10% difference ($DIF = 0{,}1$), *N* should be greater than 57894 processes.

For an overloaded network, with $t_c = 100ms$ for instance, the expression is:

$$DIF \cong (10^{-2}) \cdot N - (10^{-2}).$$

In this case, the same difference would be reached only for $N > 11$.

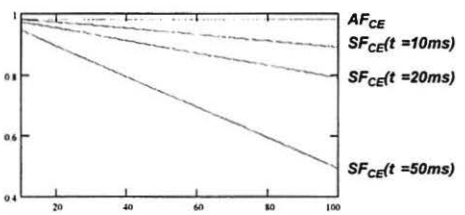

Figure 3- FS_{PR} and FA_{PR} ($TBF \rightarrow \infty$)

It was shown that increasing the time between failures makes the performance rate to grow faster for the asynchronous algorithm than for the synchronous one. However, they do not touch each other as they are asymptotic.

Otherwise, the synchronous checkpointing cost grows with the number of processes while the asynchronous one remains unchanged.

When both factors are considered, we should look for the network overload. The checkpointing invite messages of the synchronous algorithm splits up the processing time in two phases: one due to the application and other due to the algorithm actions. Passing from one phase to the other has a cost: the system have to wait for clearing the channel. The more intensive was the application message exchange before the procedure, the longer will be the time necessary to clear all messages and finally to reach and process the checkpointing invitation message.

The analysis of *N* e *TBF* shows that the asynchronous algorithm become better (with respect to synchronous) when the application message rate is increased, . However, it should be followed by a low failure rate.

References

[CEC98] Cechin, S. L. *On the theoretical performance evaluation of two rollback-recovery synchronous and asynchronous algorithms*. CPGCC / UFRGS. April, 1998 (Report TI nº 729) - In Portuguese.

[JAL94] Jalote, P. *Fault Tolerance in Distributed Systems*. New Jersey: Prentice-Hall, 1994.

[JUA91] Juang, T.; Venkatesan, S. Crash Recovery with Little Overhead. Int'l. Conf. on Distributed Computing Systems. Proceedings. May 1991. pp.454-461.

[KOO87] Koo, R; Toueg, S. Checkpointing and Rollback-Recovery for Distributed Systems. *IEEE Trans. on Software Engineering*, v.SE-13(1):23-31, Jan. 1987.

[SIN94] Singhal, M.; Shivaratri, N. *Advanced Concepts in Operating Systems*. New York: McGraw-Hill, 1994.