Speculation Meets Checkpointing

Arkadiusz Danilecki and Michał Szychowiak

Institute of Computing Science
Poznań University of Technology
Piotrowo 3a, 60-965 Poznań, Poland
{adanilecki, mszychowiak}@cs.put.poznan.pl

Abstract. This paper describes a checkpointing mechanism destined for Distributed Shared Memory (DSM) systems with speculative prefetching. Speculation is a general technique involving prediction of the future of a computation, namely accesses to shared objects unavailable on the accessing node (*read faults*). Thanks to such predictions objects can be fetched before the actual access operation is performed, resulting, at least potentially, in considerable performance improvement. The proposed mechanism is based on independent incremental checkpointing integrated with a coherence protocol introducing little overhead. It ensures the consistency of checkpoints, allowing fast recovery from failures.

1 Introduction

Modern Distributed Shared Memory (DSM) systems reveal increasing demands of efficiency, reliability and robustness. System developers tend to deliver fast systems which would allow to efficiently parallelize distributed processes. Unfortunately, failures of some system nodes can cause loss of results of the processing and require to restart the computation from the beginning. One of major techniques used to prevent such restarts is *checkpointing*, which consists in periodically saving of the processing state (a *checkpoint*) in order to restore the saved state in case of a further failure. Then, the computation is restarted from the restored checkpoint. Only the checkpoints which represent a consistent global state of the system can be used (the state of a DSM system is usually identified with the content of the memory).

The *communication induced* (or *dependency induced*) checkpointing approach offers simple creation of consistent checkpoints, storing a new checkpoint each time a recovery dependency is created (e.g. interprocess communication), but its overhead turns out to be too prohibitive for general distributed applications. However, this approach has been successfully applied in DSM systems in strict correlation with memory coherence protocols. This correlation allows to reduce the number of actual dependencies and to significantly limit the checkpointing overhead ([2],[6]).

Speculation is a technique intended to improve the efficiency of DSM operations. The speculation methods are required to be very fast, while they do not

V.N. Alexandrov et al. (Eds.): ICCS 2006, Part I, LNCS 3991, pp. 753–760, 2006.

necessary have to make correct predictions, as the cost of the mistakes is usually considered negligible. They include speculative pushes of shared objects to processing nodes before they would actually demand access [7], prefetching of the shared objects [1], or self-invalidation of shared objects [5] among other techniques.

This paper is organized as follows. Section 2 presents a formal definition of the system model and speculation operations. In Section 3 we discuss the concept of a checkpointing mechanism destined for DSM systems with speculation and propose a SpecCkpt protocol. Concluding remarks and future work are proposed in Section 4.

2 DSM System Model

A DSM system is an asynchronous distributed system composed of a finite set of sequential processes P_1, P_2, ..., P_n that can access a finite set O of shared objects. Each P_i is executed on a DSM node n_i composed of a local processor and a volatile local memory used to store shared objects accessed by P_i. Each object consists of several values (*object members*) and *object methods* which read and modify object members (here we adopt the object-oriented approach; however, our work is also applicable to variable-based or page-based shared memory). The concatenation of the values of all members of object $x \in O$ is referred to as *object value* of x. We consider here read-write objects, i.e. each method of x has been classified either as read-only (if it does not change the value of x, and, in case of nested method invocation, all invoked methods are also read-only) or read-and-modify (otherwise). Read access $r_i(x)$ to object x is issued when process P_i invokes a read-only method of object x. Write access $w_i(x)$ to object x is issued when process P_i invokes any other method of x. By $r_i(x)v$ we denote that the read operation returns value v of x, and by $w_i(x)v$ that the write operation stores value v to x.

DSM objects are replicated on distinct hosts to allow concurrent access to the same data. Concurrent processing in an asynchronous system is in general nondeterministic. A consistent state of DSM objects replicated on distinct nodes is maintained by a *coherence protocol* and depends on the assumed *consistency model*. Usually, one replica of every object is distinguished as *master replica*. The set of all replicas of a given object is referred to as *copyset*. The process holding master replica of object x is called x's *owner*. A common approach is to enable the owner an exclusive write access to the object.

The speculation introduces special part of the system, called the *predictor*, which is responsible for predicting future actions of the processes (e.g. future read and write accesses) and according reactions. Using speculation, however, an object may be fetched from its owner before the actual read access (i.e. *prefetched*), as a result of prediction. By $p_i(x)$ we will distinguish a prefetch operation on object x resulting from prediction made at process P_i.

3 Speculation and Checkpointing

3.1 Base Protocol

According to our knowledge, the impact of speculation on the checkpointing has not been investigated until now. While properly implemented speculation shall never violate the system consistency, ignoring the specific of speculation may severely damage the efficiency of checkpointing and recovery, as we will show.

We focus on prefetching techniques, but our approach should be easily adaptable to other speculation methods. In such techniques *predictor* anticipates the future read faults and prevents them by fetching respective objects in advance. The prediction may be incorrect in the sense that the process will never actually access the fetched object. Nevertheless, using speculation techniques such as the popular two level predictor MSP ([4]) turns out to increase the efficiency of most DSM applications. Since the predictor uses the underlying coherence protocol, it never violates the consistency of the memory.

Let us now consider the execution shown in Fig. 1. There is a *dependency* between processes P_1 and P_2, since P_2 fetches the value modified by P_1. To ensure the consistency in case of a subsequent failure of process P_1, the system forces P_1 to take a new checkpoint containing the previously modified object x.

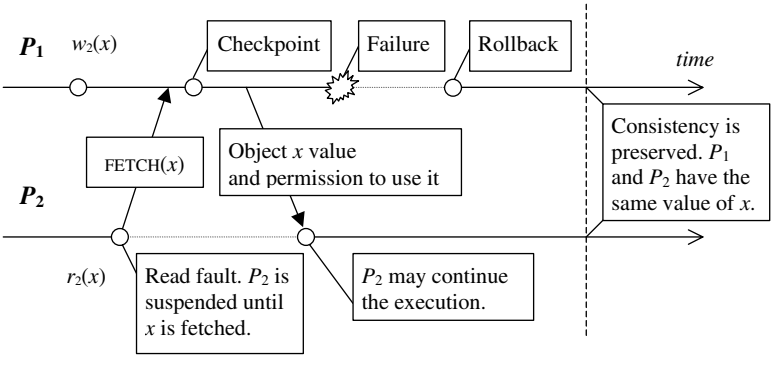

Fig. 1. Scenario without speculation. Real dependency between P_1 and P_2.

However, the situation may significantly change with speculation. In the scenario presented in Fig. 2 the predictor assumes that process P_2 will read the value modified by P_1, so it fetches the object to avoid a further read-fault. Performing that fetch, the system forces process P_1 to take a checkpoint. However, the prediction eventually turns out to be false and P_2 does not actually access x. Therefore, no real dependency was created and checkpoint was unnecessary. Unfortunately, P_1 was unable to determine that the fetch resulted from a false prediction, even if that fetch operation has been known to be speculative.

The problems presented above are summarized as follows:

– Access to objects (fetches) may result from speculation made by predictor and therefore (in case of false prediction) may not result in real dependency;

– Even when an access is marked as speculative, process has no way of determining whether true dependency between processes will ever be created, since it cannot determine whether the prediction is correct.

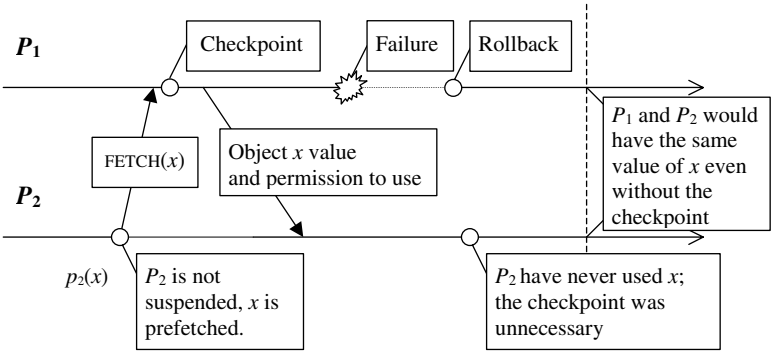

Fig. 2. Scenario with speculation. No dependency between P_1 and P_2.

A possible solution is to introduce a new replica state and decouple access requests for objects into two phases: prefetch and confirmation (Fig. 3). A speculative prefetch operation is explicitly distinguished from a coherence operation of a read access. The prefetched object replica is set into state PREFETCHED on the requesting node, and PRESEND on the owner. Further read access performed on the requesting node requires to ask for acknowledgment of accessing the object (message CONFIRM). On reception of this message the owner takes a checkpoint of the object, if necessary (e.g. the checkpoint could been taken already before reception of CONFIRM request as a result of some operations issued by other processes), and answers with a permission message (ACK).

Please note that ACK message does not contain the value the requested object (since this value has been formerly prefetched and is available for the requesting node). Therefore the overhead of the confirmation operation is in general lower than of a read-fault.

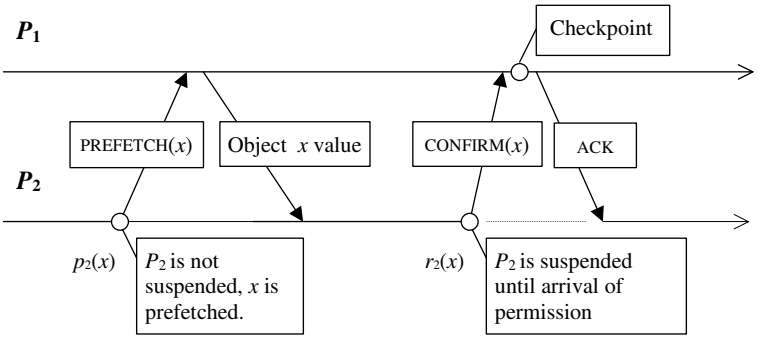

Fig. 3. Coherence decoupling

If the master replica of the considered object has been modified after a prefetch but before the corresponding confirmation it is up to the coherence protocol to decide about the acknowledgment (reading outdated values may be allowed depending on the consistency model). The coherency protocol may force the invalidation of a prefetched object before the confirmation. This invalidation will be performed exactly as for objects fetched by nonspeculative operations. Since there is no difference between those two types of operations from the point of view of the coherence, only minor modifications of coherence protocols will be necessary. The only significant difference concerns the checkpointing operations.

Our approach avoids unnecessary taking of checkpoints after a prefetch (when no real dependency is created). The checkpoint is postponed until an actual dependency is revealed on the confirmation request.

3.2 Protocol Improvement

Addressing the Protocol Efficiency. There are several possible ways to further increase the protocol efficiency. It is possible to perform a consolidated checkpoint of an entire group of objects (i.e. *burst checkpoint* [2]). This may significantly reduce the cumulative checkpointing overhead.

For instance, at the moment of further confirmation the prefetched object demanding confirmation may have already been checkpointed (during some previous burst checkpoint) and no new checkpoint will then be required. In such situation, no checkpoint overhead will be perceived by the application neither on prefetch, nor on actual read access to the prefetched object.

Another possible optimization is to send confirmations to all prefetched replicas directly after every checkpoint. The improvement of the efficiency is achieved by avoiding the need of confirmation during a further access to the replica prefetched earlier.

Addressing the Protocol Correctness. Let us consider a recovery situation presented in Fig. 4. After the value 1 of x has been checkpointed, it is modified again, to 2. Process P_2 prefetches the modified value of x from P_1. Then, P_1 fails and recovers, restoring the checkpointed value $x = 1$. Please note that the confirmation requested by P_2 cannot be granted, as it concerns a value of x that became inconsistent after the recovery.

In order to ensure the consistency, the recovered process P_1 might simply invalidate every replica prefetched from P_1 and not confirmed yet or, alternatively, refuse all confirmation requests received after the recovery. While those two solutions do prevent system from becoming inconsistent, they are far from being optimal. The first approach may unnecessarily invalidate prefetched replicas which were consistent (an unconfirmed replica may be perfectly consistent, as presented in Fig. 5), or invalidate prefetched replicas which would never be used anyway (therefore introducing unnecessary communication costs). The second approach is even worse, since it basically turns off the whole prefetching mechanism after the first failure. Optimal solution should both prevent the system from becoming inconsistent and allow the confirmation of all prefetched replicas that do not violate the consistency.

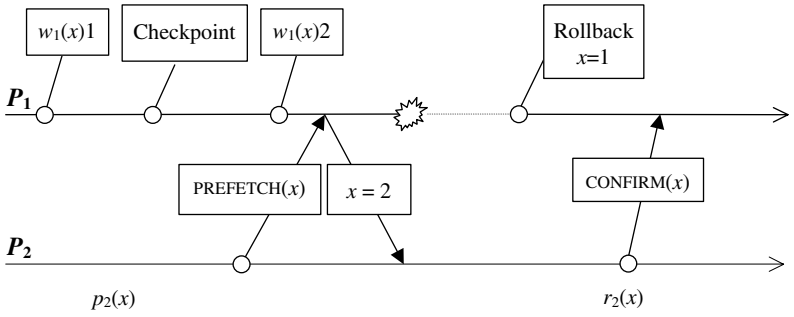

Fig. 4. Possible coherence problems with node failures

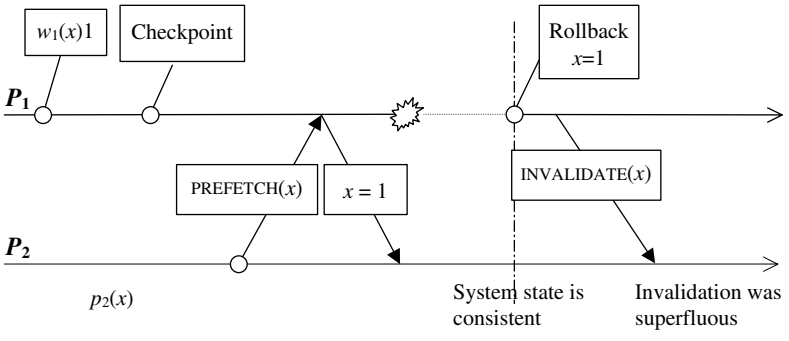

Fig. 5. Unnecessary invalidations of all prefetched replicas after the owner recovery

One intuitive and simple (and wrong, as we will soon show) solution is to use *version numbers*, increased after each meaningful change of the object (by meaningful we understand the first object modification after each checkpoint). Version numbers are stored in the checkpoints. Only replicas with version number equal to *master version number* (the version number of the master replica, restored from checkpoint after the recovery) can be confirmed, and all other confirmations would be refused.

However, this approach may result in inconsistent state after the recovery. Let us consider a simple example illustrated by Fig. 6. Owner P_1 modifies the object x, therefore increasing the version number $v(x)$ from m to $m + 1$. This version of x would be prefetched by another process P_2. Please note, that this version is not checkpointed. After the recovery, master replica of x would be rolled back to version m, and a subsequent change would again increase the version number to $m + 1$ When process P_2 would then ask for confirmation of his prefetched replica, it would appear that he has the correct version of the object, and the confirmation will be granted, possibly wrongfully.

Therefore, we investigate other possibilities, discussed in depth in [3]. Here we present one which solves all the problems described above.

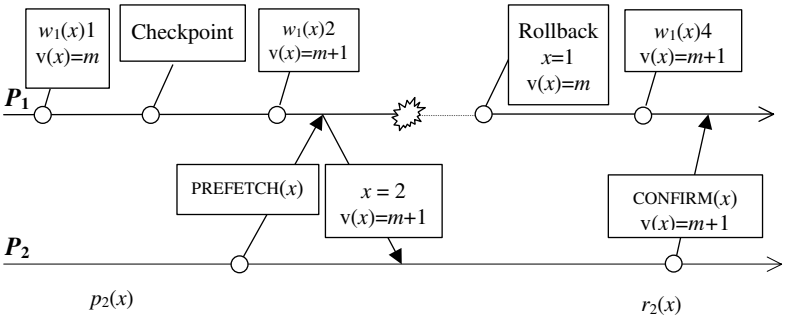

Fig. 6. Possible consistency violation in the approach with version number

The proposed checkpointing protocol, SpecCkpt, combines version numbers and the approach with invalidation of all prefetched replicas on recovery. Owners maintain a version number associated with the objects. After the recovery, the owner sends an invalidation request containing the version number restored from the checkpoint. The receiving processes invalidate the prefetched replicas only if their local version numbers are larger than the one received in the invalidation message. This approach keeps the system consistent by invalidating those and only those replicas which could violate it. The only small vice is the additional communication costs, which may be unnecessary if the invalidated prefetched replicas would never to be used anyway (i.e on mispredication).

Finally, let us present a remark about the optimization with confirmation of all prefetched replicas on every checkpoint. When using this approach, if the replica is in the prefetched state, it might be safely assumed that it is not consistent with the version of the object restored from the checkpoint. Therefore, it's enough to simply invalidate all prefetched replicas on recovery.

4 Conclusions

This paper describes an approach to checkpointing shared objects with speculation. We recognize the false dependencies and unnecessary checkpoints related to speculative operations on shared objects. We propose the operation decoupling which allows to decrease the frequency of checkpoints. Moreover, we describe additional mechanisms reducing the checkpointing overhead and enabling fast recovery. Practical verification of an implementation of SpecCkpt protocol is currently performed.

There are at least two directions in which our approach could be further studied and extended. First is to integrate the implementation of the proposed checkpointing technique with a particular coherence model. Second direction is to seek the optimizations for increasing positive effects of speculation.

Since our approach is very general, it still allows several optimizations concerning distinct characteristics of the protocol.

In the presented protocol, when the owner refuses to confirm the prefetch, the prefetched object is invalidated. In the optimized version the current value of the object may be sent along with ACK message.

In many typical scientific applications there are program loops which produce strictly defined sequence of requests. Commonly employed in such cases is grouping the objects accessed in the loop into blocks, fetching (or prefetching) them together. Access to the first object from such group may signal that the program loop started again and other objects from this group will also be fetched subsequently. Therefore, it appears useful to confirm the whole group on access to the first object.

References

1. Bianchini, R., Pinto, R., Amorim, C.L.: Data Prefetching for Software DSMs. Int. Conf. on Supercomputing, Melbourne, Australia (1998)
2. Brzeziski, J., Szychowiak, M.: Replication of Checkpoints in Recoverable DSM Systems. 21^{st} Int. Conf. on Parallel and Distributed Computing and Networks PDCN'2003, Innsbruck, Austria (2003)
3. Danilecki, A., Szychowiak, M.: Checkpointing Speculative DSM Systems. Technical Report RA-021/05, Institute of Computing Science, Poznań University of Technology, Poznań, Poland (2005)
4. Lai, A-C., Babak Falsafi, B.: Memory Sharing Predictor: The Key to a Speculative Coherent DSM. 26^{th} Int. Symp. on Computer Architecture (ISCA 26), Atlanta, Georgia (1999) 172–183
5. Lai, A-C., Babak Falsafi, B.: Selective, Accurate, and Timely Self-Invalidation Using Last-Touch Prediction. 27^{th} Int. Symp. on Computer Architecture (ISCA 27), Vancouver, BC, Canada (2000) 139–148
6. Park, T., Yeom, H.Y.: A Low Overhead Logging Scheme for Fast Recovery in Distributed Shared Memory Systems. Journal of Supercomputing Vo.15. No.3. (2002) 295–320
7. Rajwar, R., Kagi, A., Goodman, J. R.: Inferential Queueing and Speculative Push. Int. Journal of Parallel Programming (IJPP) Vo. 32. No. 3 (2004) 273–284