Checkpointing Speculative Distributed Shared Memory[⋆]

Arkadiusz Danilecki, Anna Kobusińska, and Michal Szychowiak

Institute of Computing Science,
Poznań University of Technology,
Piotrowo 3a, 60-965 Poznan, Poland
{adanilecki, akobusinska, 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 checkpointing integrated with a coherence protocol for a given consistency model 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 process crashes resulting in a loss of results of the processing and requiring to restart the computation from the beginning. One of major techniques used to prevent such restarts is *checkpointing*. Checkpointing 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).

There are two major approaches to checkpointing: coordinated (synchronous) and independent (asynchronous). Coordinated checkpointing requires expensive synchronization between all (or a part) of the distributed processes in order to ensure the consistency of the saved states. The significant overhead of this approach makes it impractical unless the checkpoint synchronization is correlated with synchronization operations of a coherence protocol ([4]). On the other

⋆ This work has been partially supported by the State Committee for Scientific Research grant no. 3T11C 073 28.

hand, the independent checkpointing does not involve interprocess synchronization but, in general, does not guarantee the consistency. After a failure occurs, a consistent checkpoint must be found among all saved checkpoints, therefore the recovery takes much more time and may require much more recomputation. A variant of the independent checkpoint – *communication induced checkpointing* (or *dependency induced checkpointing*), offers simple creation of consistent checkpoints, storing a new checkpoint each time a recovery dependency is created (e.g. interprocess communication), but its overhead is too prohibitive for general distributed applications. However, this approach has been successfully applied in DMS 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],[10]).

Speculation, on the other hand, is a technique which promises to increase the speed of DSM operations and reduce the gap between DSM systems and message-passing systems. The speculation may involve speculative pushes of shared objects to processing nodes before they would actually demand access [11], prefetching of the shared objects with anticipation that application process would need those objects ([1],[8],[12]) or self invalidation of shared objects to reduce the frequency of "3-hop-misses" ([6],[7]) among other techniques.

The speculation techniques may be argued to be special form of machine learning; it's however a restricted and limited form of learning. The speculation methods are required to be very fast, while they do not necessary have to make correct predictions, as the cost of mistakes is usually considered negligible. Therefore the existing well-known machine learning algorithms are usually not applicable in the DSM.

This paper is organized as follows. Section 2 presents a formal definition of the system model and speculation operations. In Section 3 we propose the conception of a checkpointing mechanism destined for DSM systems with speculation and discuss the proposition. 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. Each write access results in a new object value 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. For the sake of simplicity of the presentation we assume that each write access to an object writes a unique value.

To increase the efficiency of DSM, objects are replicated on distinct hosts, allowing concurrent access to the same data. 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 the *master replica*. 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. However, when no write access to x is performed, the object can have several replicas simultaneously accessible only for reading (shared replicas). 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.

As a result of a read access issued to an object unavailable locally, the object is fetched from its owner and brought to the requester. Using speculation, however, an object may be fetched from its owner also as a result of a prediction before the actual read access (i.e. *prefetched*). By $p_i(x)$ we will distinguish a prefetch operation of object x resulting from prediction made at process P_i.

Dependency of operations is a relation arising between $w_i(x)v$ and any subsequent $r_j(x)v$, i.e. when process P_j uses (reads) a value written previously by P_i. *Local dependency* reflects the order of operations performed by the same single process.

3 Speculation and Checkpointing

According to our knowledge, the impact of speculation on the checkpointing has not been investigated until now. While it seems impossible (or at least, improbable) that properly implemented speculation may danger the consistency of the system and correctness of the checkpointing algorithms, ignoring the existence of speculation in distributed shared memory system may severely damage speed of both making checkpoints and system recovery because it could create false, non-existing dependencies between nodes, as we will try to show.

We will focus on problems resulting from using prefetching techniques, but our approach should be easily adaptable to other speculation methods. In such techniques a special part of the system, called *predictor*, is responsible for anticipating the possible future read faults and preventing 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 ([5]) turns out to increase the efficiency of most DSM applications. Moreover, since the predictor fetches objects using the underlying coherence protocol, it never violates the consistency of the memory.

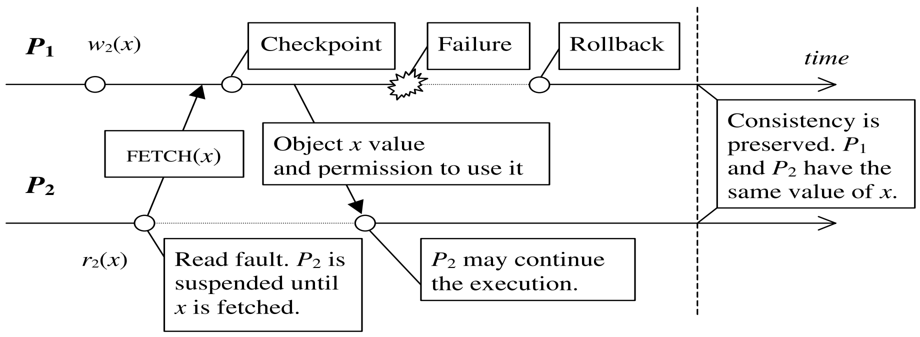
P1
w2(x)
Checkpoint
Failure
Rollback
time
FETCH(x)
Object x value and permission to use it
Consistency is preserved. P1 and P2 have the same value of x.
P2
r2(x)
Read fault. P2 is suspended until x is fetched.
P2 may continue the execution.

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

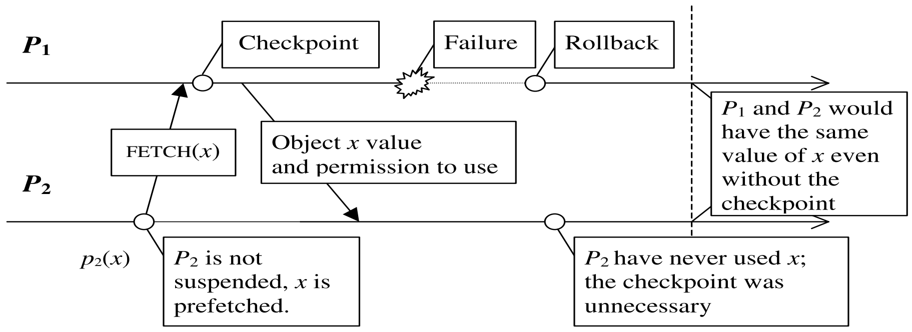
P1
Checkpoint
Failure
Rollback
FETCH(x)
Object x value and permission to use
P1 and P2 would have the same value of x even without the checkpoint
P2
p2(x)
P2 is not suspended, x is prefetched.
P2 have never used x; the checkpoint was unnecessary

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

Let us now consider the hypothetical 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 checkpoint of the previously modified object x (it may be necessary to save also some other objects in the same checkpoint, in order to preserve local dependency of modifications performed by P_1; this is not shown in the figure).

However, the situation may significantly change with use of speculation. In the scenario presented in Fig. 2 the predictor assumes that the application process P_2 will read the value modified by P_1, so it fetches the object ahead into the local memory of P_2, to avoid a further read-fault. Performing that fetch, the system forces process P_1 to take a checkpoint, as in previous example. 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;

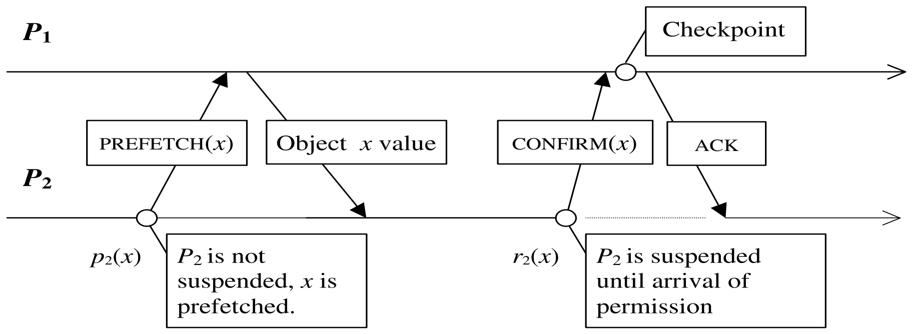
P1
Checkpoint
PREFETCH(x)
Object x value
CONFIRM(x)
ACK
P2
p2(x)
P2 is not suspended, x is prefetched.
r2(x)
P2 is suspended until arrival of permission

Fig. 3. The coherence decoupling

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 (otherwise, it wouldn't be called speculation).

A possible solution is introduction of a new replica state and decoupling of access requests for objects into two phases: prefetch and confirmation (Fig. 3). A similar idea of coherence decoupling has been proposed in [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 merely ask for acknowledgement 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 in the meantime 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 a read-fault.

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 acknowledgement (reading outdated values may be disallowed depending on the consistency model). Also the coherency protocol may involve invalidation of a prefetched object before the confirmation. This invalidation will be performed for prefetched objects exactly as for all object fetched by nonspeculative operations. Therefore, there is no difference between those two types of operations from the point of view of the coherence (thus, 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). To reduce the checkpoint overhead many checkpointing protocols perform a consolidated checkpoint of an entire group of objects (*burst checkpoint* [2]). It is possible to include also the prefetched objects in such a burst checkpoint. This allows to further reduce the checkpointing overhead, since the prefetched object may already be checkpointed at the moment of confirmation and no additional checkpoint will be required. In such a situation, there will be no checkpoint overhead perceived by the application neither on prefetch, nor on actual read access to the prefetched object.

Finally, let us consider a recovery situation presented in Fig. 4. After the value 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 following confirmation cannot be granted, as it concerns a value of x that became inconsistent after the recovery. The simplest solution could be to invalidate all replicas of x prefetched by other processes. This Invalidation can be performed on recovery of the owner.

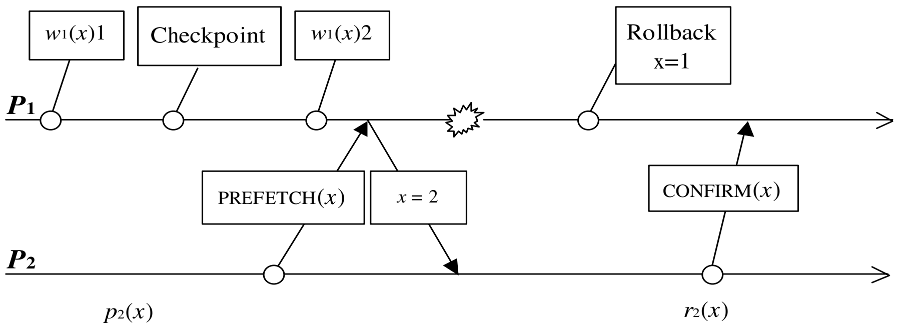

Fig. 4. Possible coherence problems with node failures

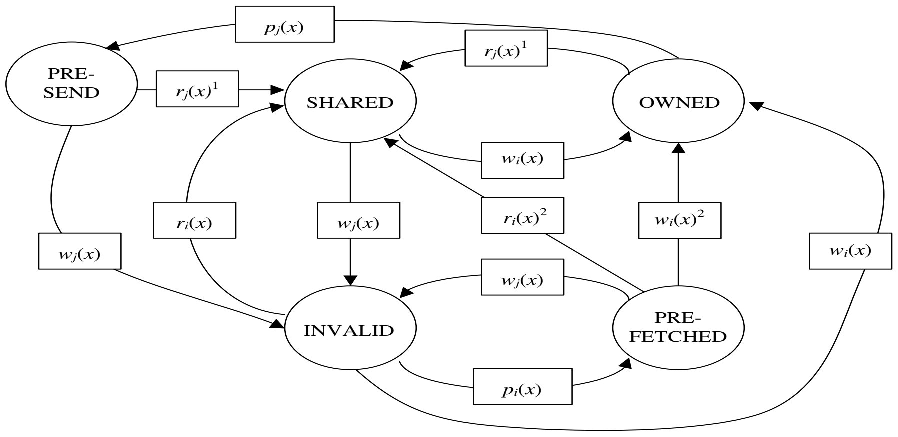

Fig. 5. The example state diagram of the protocol for sequential consistency model

The state diagram for replica of object x at process P_i is presented in Fig. 5. The assumed consistency model is sequential consistency [9]. The superscript indexes at the diagram denote that: ^{1}operation requires a checkpoint, ^{2} operation requires a confirmation.

4 Conclusions

This paper describes an approach to checkpointing shared objects with use of speculation. We recognize the false dependencies and unnecessary checkpoints related to speculation operations on the 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 indispensable modifications of the coherency operations after a failure and recovery.

There are at least three directions in which our approach could be studied and extended. First, to consider the implementation of proposed technique with using concrete coherence model and checkpointing algorithm. Second, to seek the optimizations for increasing positive effects of speculation. Third, to find a way to resolve issues with restarting processes.

Intuitively, there may be many possible optimizations which could be applied to the proposed checkpointing technique. Since our approach is very general, the implementation for a specific coherence model may exploit distinct features of underlying protocols. An obvious optimization might allow to use the prefetched object even before arrival of the permission.

In our approach, if object owner refuses to confirm the prefetch, the prefetched object is invalidated. Another optimization might fetch the current value of the object with ACK message sent back to the requester.

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.

Requiring the object owner to deny all confirmation request after the failure may seem to be too harsh. A different solution would allow the object owner to refuse confirmation only for objects prefetched before crash, and acknowledge objects prefetched after the recovery.

References

1. Bianchini, R., Pinto, R., Amorim, C. L.: Data Prefetching for Software DSMs. Proc. International Conference on Supercomputing, Melbourne, Australia (1998)
2. Brzeziński, J., Szychowiak, M.: Replication of Checkpoints in Recoverable DSM Systems. Proc 21st Int'l Conference on Parallel and Distributed Computing and Networks PDCN'2003, Innsbruck (2003)

3. Chang, J., Huh, J., Desikan, R., Burger, D., Sohi, G.: Using Coherent Value Speculation to Improve Multiprocessor Performance. Proc 30th Annual International Symposium on Computer Architecture, San Diego, California (2003)
4. Kongmunvattana, A., Tanchatchawal, S., Tzeng, N.-F.: Coherence-Based Coordinated Checkpointing for Software Distributed Shared Memory Systems. Proc. 20th Conference on Distributed Computing Systems (2000) 556-563
5. Lai, A-C., Babak Falsafi, B.: Memory Sharing Predictor: The Key to a Speculative Coherent DSM. Proceedings of the 26th International Symposium on Computer Architecture (ISCA 26), Atlanta, Georgia (1999) 172-183
6. Lai, A-C., Babak Falsafi, B.: Selective, Accurate, and Timely Self-Invalidation Using Last-Touch Prediction. Proceedings of the 27th International Symposium on Computer Architecture (ISCA 27), Vancouver, BC, Canada (2000) 139-148
7. Lebeck, A., R., Wood, D.: Dynamic Self-Invalidation: Reducing Coherence Overhead in Shared-Memory Multiprocessors. Proceedings of the 22nd Annual International Symposium on Computer Architecture, Santa Margherita, Italy (1995) 48-59
8. Lee, S-K., Yun, H-C., Lee, J., Maeng, S.: Adaptive Prefetching Technique for Shared Virtual Memory. First IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2001). Brisbane, Australia (2001) 521-526
9. Raynal M., Schiper A.: A Suite of Formal Definitions for Consistency Criteria in Distributed Shared Memories, Technical Report PI-968, IRISA Rennes (1995).
10. 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
11. Rajwar, R., Kagi, A., Goodman, J. R.: Inferential Queueing and Speculative Push. International Journal of Parallel Programming (IJPP) Vo. 32. No. 3 (2004) 273-284
12. Speight, E., Burtscher, M.: Delphi: Prediction-Based Page Prefetching to Improve the Performance of Shared Virtual Memory Systems. 2002 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), Monte Carlo Resort, Nevada (June 2002) 49-55