Using Speculative Push for Unnecessary Checkpoint Creation Avoidance

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 discusses a way of incorporating speculation techniques into Distributed Shared Memory (DSM) systems with checkpointing mechanism without creating unnecessary checkpoints. 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 pushed to requesting nodes before the actual access operation is performed, resulting, at least potentially, in a considerable performance improvement. This mechanism is a foundation for the proposed SpecCkpt protocol based on independent checkpointing integrated with a coherence protocol for a given consistency model introducing little overhead. It ensures the consistency of checkpoints, at the same time allowing a fast recovery from failures.

1 Introduction

Modern Distributed Shared Memory (DSM) systems reveal increasing demands for efficiency, reliability and robustness. System developers tend to deliver fast systems which would allow to parallelize distributed processes efficiently. 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 the techniques used to prevent such restarts is *checkpointing*. Checkpointing consists in saving the processing state periodically (a *checkpoint*), in order to restore the saved state in case of further failure. Only checkpoints which represent a consistent global state of the system can be used when restarting computation (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 ([7]). On the other hand, the independent checkpointing does not involve interprocess synchronization but, in general, does not guarantee the consistency. After a failure occurs, a

F. Eliassen and A. Montresor (Eds.): DAIS 2006, LNCS 4025, pp. 309–315, 2006.

consistent checkpoint must be found among all saved checkpoints, therefore the recovery takes much more time and may require more recomputation. A variant of the independent checkpoint – *communication induced checkpointing* (or *dependency induced checkpointing*) – offers simple creation of consistent checkpoints, by storing a new checkpoint each time a recovery dependency is created (e.g. interprocess communication). However, its overhead is too prohibitive for general distributed applications. Nevertheless, this approach has been successfully applied in DMS systems in strict correlation with memory coherence protocols. This correlation allows to reduce the number of the actual dependencies and to limit the checkpointing overhead significantly ([9]).

Speculation, on the other hand, is a technique which promises to increase the speed of DSM operations and to reduce the gap between DSM and message-passing systems. The speculation may involve speculative pushes of shared objects to processing nodes, before they actually demand access [10], prefetching of the shared objects with anticipation that the application process would need those objects ([1]) or self invalidation of shared objects to reduce the frequency of "3-hop-misses" ([8]) among other techniques.

This paper is organized as follows. In section 2, we present a formal definition of the system model and speculation operations. The previous work in this field, including the first variant of SpecCkpt protocol, is briefly described in Section 3. Section 4 discusses the ways of combining speculative pushes into DSM systems with checkpointing. Preliminary results are contained within Section 5 Concluding remarks and future work are proposed in Section 6.

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. Here, we consider 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 the 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.

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 a *master replica*. The set of all replicas of a given object is referred to as a *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. 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 a 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 proper reactions.

As a result of a read access issued to an object locally unavailable, the object is fetched from its owner and brought to the requester. Using speculation, however, the object may be fetched from its owner also before the actual read access (i.e. *prefetched*) or forwarded by the owner to potential object users (i.e. *pushed*), as a result of prediction. By $p_i(x)$ we will denote a prefetch operation of object x resulting from the prediction made at process P_i. By $f_{i,j}(x)$ we will denote a push operation of object x from owner P_i to potential object user P_j. The prediction is successful if the pushed or prefetched object is actually used, and the read fault is avoided. If the prefetched or pushed object is never used, the prediction was unsuccessful (and is referred to as *misprediction*).

Dependency of operations is a relation arising between $w_i(x)$ and any subsequent $r_j(x)$, $p_j(x)$ or $f_{i,j}(x)$ i.e. when process P_j uses (reads) a value previously written by P_i. *Local dependency* reflects the order of operations performed by a single process. To ensure system consistency in case of a failure, the system forces the object owner to make a checkpoint each time the dependency arises.

3 Previous Work

In the previous work on reliable speculative DSM systems ([4], [3]), it has been shown that a naive implementation of the speculation and, more specifically, prefetching may reduce DSM efficiency by introducing false dependencies, which in turn increase the number of unnecessary checkpoints. Since prefetches are seen by the object owner as read accesses, they create dependencies. However, because they are speculative operations, the object owner has no way to determine whether a prefetched object will actually be used (prediction was successful) or will be invalidated before using or never accessed by requesting node (on misprediction).

These problems are summarized as follows:

- An access to objects (fetches) may result from the speculation made by the predictor and therefore (in case of a false prediction), may not create a real dependency;
- Even when the access is explicitly marked as speculative, the process has no way of determining whether a true dependency between processes will ever be created, since it cannot determine whether the prediction is correct (otherwise, it wouldn't be a speculation).

To avoid the creation of unnecessary checkpoints, the changes to the underlying checkpointing protocol have been proposed. They consist in introducing a new replica state (PREFETCHED) and operation decoupling. In the proposed SpecCkpt protocol, prefetched objects are put into a special PREFETCHED state. The access to prefetch object would then require getting confirmation from the owner. This confirmation would be granted or not, depending on the underlying coherence protocol.

This protocol avoids unnecessary checkpoints at the cost of reducing positive speculation effects (even a successful prediction needs a confirmation message). We have verified our protocol using a simulated DSM system (see sec. 5) and we found our results somewhat dissatisfying. In some of the tested application our protocol behaved surprisingly bad, so we decided to search for another techniques.

4 Speculative Push

Another way of avoiding the creation of unnecessary checkpoints may be using a different speculation technique, namely speculative push. In prefetching, speculation is triggered by the node which has the object in the INVALID state; in the speculative push, it is the object's owner who triggers the speculation, pushing the object to potential requesters. It may be observed that in the prefetching, the requesting node has no way of determining whether the object owner has already checkpointed the object. So, it does not know whether the prefetch request would force the object owner to take a checkpoint, or not. In the latter technique however, the object's owner has obviously a full knowledge of its own local state and therefore is able to determine when the speculation results in making a checkpoint. Usually, the object's owner forwards the object to anticipated requesters after it finishes the object modifications. However, this may involve creating a checkpoint, which may be unnecessary (because the push may result from misprediction).

Our proposal is to trigger a speculative push of the object each time the object is checkpointed. Therefore, even if the push is unnecessary and creates a false dependency, it does not introduce additional costs (since the page is already checkpointed).

We have tested this proposal for sequential consistency model and MSI (Modified, Shared, Invalid) coherence protocol. The implemented algorithm is described as follows:

- Between the checkpoints, the object's owner records remote accesses to the objects in the SHARED state. Those will be potential candidates for data forwarding. Essentially, it is the object *copyset*.
- On local write access, the owner invalidates the copyset but keeps the list of the potential candidates (*possible copyset*).
- If the owner has the object in the MODIFIED state and receives a read request, it checkpoints the page before answering. The requester is removed from possible copyset list but added to the copyset list.

- After the owner has successfully checkpointed the object, it forwards the object to all potential requesters and clears the possible copyset list. Such new replicas are put into a special PUSHED state (and the respective nodes are put into the master replica copyset).
- Request from a node is ignored if the owner has already pushed the object to that node.
- The access to a replica in the PUSHED state is treated as the access to replica in the SHARED state, with the exception when a confirmation is sent to the owner. The only purpose of this confirmation is to provide feedback for the owner, so that it could add the node again to the possible copyset list.

5 Preliminary Results

The simulation was performed with the use of a backend [5] to Augmint simulator [2] and a set of applications from the SPLASH-2 suite [6]. The obtained results should be treated rather as indications of the trends, not the final results. Compared to the results found in literature, our simulation consequently tends to be too optimistic about the positive effects of prediction, probably because of the simplified simulator architecture (the impact of increased network traffic, costs of the owner searching, the cost of a single checkpoint is modelled as a constant).

Due to the space limits, we present only some of the results here. The following basic set of prefetch techniques was used for comparison: simple stride-based prefetch; prefetching pages which were recently invalidated; prefetching the same set of pages which were used before attempting a barrier; prefetching neighboring pages; and combination of all those techniques.

Table 1. The results of the simulation with standard application inputs and 8 processes, using the best prefetch technique

Application name	(a)	(b)	(c)	(d)	(e)	(f)	(g)
barnes	46	97	97	38	101	100	95
fft	11	98	97	54	103	101	100
lu	36	87	85	43	108	102	95
fmm	33	100	100	36	103	100	97

Different prefetching techniques prove to yield the best effects for different applications. We decided to choose those which were the most and the least efficient for every application and compare the reduction of execution time against the base checkpointing protocol without speculation. Relative reduction of execution time is shown in table 1. First, we evaluated the prefetching used without any modifications of the base MSI checkpointing protocol (column b and e). Then, we compared it to SpecCkpt protocol using the same techniques (column c and f). The misprediction ratio for those techniques can be found in columns

a and *d*. Finally, the *g* column represents the relative execution times achieved with SpecCkpt protocol using speculative pushes.

From the whole set of the obtained results, we concluded that the most influential factor is the misprediction ratio. If the misprediction ratio is low, then usually our SpeckCkpt protocol, using either prefetches or pushes, is outperformed by the base protocol. However, for different applications different techniques turned out to have a low misprediction ratio. It would be, in general, impossible for a DSM system to determine the best suited technique in advance. Therefore, in real systems, higher misprediction ratio is to be expected.

6 Conclusions

This paper proposed the use of speculative pushes instead of speculative prefetches in DSM systems with checkpointing. Since the object owner is able to determine whether the push will result in checkpoint or not, it may decide when the pushes do not introduce additional significant costs resulting from the checkpoints.

We intend to implement our protocols first in another simulator (to validate our results) and then in the real linux-based DSM system. We are in the process of validating and preparing tests of a few other ideas of improving our protocols, namely prefetch delaying and speculative checkpoints.

References

1. Bianchini, R., Pinto, R., Amorim, C. L.: Data Prefetching for Software DSMs. Proc. Int. Conference on Supercomputing, Melbourne, Australia (1998)
2. Carbajal, J., Michael, M., Nguyen, A-T., Torrellas, J., Sharma, A.: Augmint: A Multiprocessor Simulation Environment for Intel x86 Architectures. CSRD Technical Report 1463, March 1996
3. Danilecki A., Szychowiak M.: Checkpointing Speculative Distributed Shared Memory. To appear in Proc. 6^{th} Int. Conference on Parallel Processing and Applied Mathematics PPAM'2005, Poznan 2005
4. Danilecki A., Szychowiak M.: Checkpointing Speculative DSM Systems, Research Report RA-021/05, Institute of Computing Science, Poznan University of Technology, 2005.
5. Danilecki A., Szychowiak M., Kobusinski J.: Simplified DSM simulation with the use of the Augmint backend, Research Report RA-04/06, Institute of Computing Science, Poznan University of Technology, 2006.
6. Gupta, A., Ohara, M., Singh, J., Torrie, E., Woo, S., The SPLASH2 Programs: Characterization and Methodological Considerations. Proc. 22^{nd} Int. Symposium on Computer Architecture (ISCA 1995), May 1995
7. Kongmunvattana, A., Tanchatchawal, S., Tzeng, N.-F.: Coherence-Based Coordinated Checkpointing for Software Distributed Shared Memory Systems. Proc. 20^{th} Conference on Distributed Computing Systems (2000) 556–563
8. Lai, A-C., Babak Falsafi, B.: Selective, Accurate, and Timely Self-Invalidation Using Last-Touch Prediction. Proc. 27^{th} Int. Symposium on Computer Architecture (ISCA 27), Vancouver, BC, Canada (2000) 139–148

9. 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
10. 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