Dvé Appendix: Coherence Protocol Specification and Verification

1. Summary

In this appendix, we describe how we model and verify the allow-based and deny-based protocol variants. We also provide the complete specification of the allow-based replica directory controller including transient states and actions, as a sample specification.

2. Assumptions

The cache hierarchy is strictly inclusive and all of the generated protocols use a strict hierarchical intervention-forwarding communication pattern. This means a last-level cache (LLC), for a request associated with the replica directory (replica-LLC), communicates only with the replica directory and never communicates directly with the home directory. On receipt of a request, the replica directory issues a request to the home directory if required. Similarly, responses from the home directory are sent to the replica directory first, which then forwards it to the LLC. In addition, an ordered network is used to ensure point-to-point ordering.

3. Verification

We have verified the complete specifications of the allow-based and the deny-based protocols using the Murϕ model checker for safety and deadlock-freedom. A protocol is deemed to be safe if it enforces the Single-Writer-Multiple-Reader invariant (at any time there exists a single writer or multiple readers) and the data-value-invariant (a read always returns the value of the most recent write) across all of the caches, and the 2 replicas. Note that these two invariants implicitly ensure that the data and replica are in sync.

We first verified the protocols for our system model in our evaluation: i.e., 2 sockets, with one replica directory (R_D) and one home directory (H_D) serving requests to a single LLC each.

In addition to the reference system discussed in the paper, we have also verified an allow-based and a deny-based protocol implementation for a system with 4 sockets and 2 replicas: i.e., one replica directory (R_D) and one home directory (H_D) which each manage two LLCs.

4. Allow-based Replica Directory Spec

The complete specification of an allow-based replica directory with a single LLC is given in tables 1 and 2.

Table 1 lists the requests and responses from the LLC to the replica directory. In addition to the LLC state machine transition events/messages, an evict event (second column – R_D, evict) is defined for the replica directory. This evict event is required because the the replica directory might hold an entry for a block which is not cached in the LLC. If the replica directory gets a request for an untracked cache block when it has no free entries, it must be able to evict another entry. As long as the LLC maintains a copy of the cache block, the replica directory remains in state S. An entry in the S^{NS} (Shared-NoSharers) state implies that the replica directory can serve a GetS request without consulting the home directory again. Since this is a single LLC implementation we used two different states (S and S^{NS}) to track if the LLC maintains a shared copy of the cache block or not. However, for an implementation with multiple LLCs, we use a sharer vector implementation with a single state S. Note that if a dirty cache block is either evicted from the LLC or is requested by another LLC with shared permissions, it is written back to the replica and the home node's memory.

Table 2 lists remote requests and responses from the home directory. All messages apart from Fwd_WB are part of a standard MSI protocol. The Fwd_WB message is sent from the home directory, if its local LLC writes back a dirty cache block, thus requiring a writeback to the replica memory.

5. Notation

- Receive message: ?Message
- Send message: !Message(Destination, *Payload) *Payload is optional
- Next state transition: → Next State
- If a message arrives in a state where the table entry is grey it is considered to be stalled.
- Write to replicate memory : RepMemWr
- Read from replicate memory: RepMemRd
- Last level cache: LLC
- Replica Directory: R_D
- Home Directory: H_D

Table 1: Allow-Based Protocol: Eviction and LLC requests

	R_D	LLC						
State	evict	?GetS	?GetX	?PutS	?PutX	?Inv_Ack	?GetS_Ack	?GetX_Ack
I		!GetS(H_D) $\rightarrow I^S$	!GetX(H_D) $\rightarrow I^M$					
S			!GetX(H_D) $\rightarrow S^M$	!Put_Ack(LLC) $\rightarrow S^{NS}$				
S^{NS}	!PutS(H_D) $\rightarrow S^I$	RepMemRd !GetS_Ack(LLC, Data) $\rightarrow$ S	!GetX(H_D) $\rightarrow S^M$					
S^F						!Inv_Ack(H_D) $\rightarrow$ I		
S^{IF}						!Inv_Ack(H_D) $\rightarrow S^I$		
S^{MF}						!Inv_Ack(H_D) $\rightarrow I^M$		
M					RepMemWr !PutX(H_D, Data) $\rightarrow M^I$			
M^F							RepMemWr !GetS_Ack(H_D, Data) $\rightarrow$ S	RepMemWr !GetX_Ack(H_D, Data) $\rightarrow$ I
M^{IF}							RepMemWr !GetS_Ack(H_D, Data) $\rightarrow M^{IS}$	RepMemWr !GetX_Ack(H_D, Data) $\rightarrow M^{II}$

Table 2: Allow-Based Protocol: Home Directory forwarded requests and responses

	H_D						
State	?Inv	?Fwd_GetS	?Fwd_GetX	?GetS_Ack	?GetX_Ack	?Put_Ack	?Fwd_WB
I							RepMemWr !WB_Ack(H_D)
I^S				RepMemWr !GetS_Ack(LLC, Data) $\rightarrow$ S			RepMemWr !WB_Ack(H_D)
I^M					RepMemWr !GetX_Ack(LLC, Data) $\rightarrow$ M		RepMemWr !WB_Ack(H_D)
S	!Inv(LLC) $\rightarrow S^{IF}$						
S^{NS}	!Inv_Ack(H_D) $\rightarrow$ I						
S^I	!Inv_Ack(H_D)					!Put_Ack(LLC) $\rightarrow$ I	RepMemWr !WB_Ack(H_D)
S^M	!Inv(LLC) $\rightarrow S^{MF}$				RepMemRd !GetX_Ack(LLC, Data) $\rightarrow$ M		
M		!Fwd_GetS(LLC) $\rightarrow S^F$	!Fwd_GetX(LLC) $\rightarrow M^F$				
M^I		!Fwd_GetS(LLC) $\rightarrow S^{IF}$	!Fwd_GetX(LLC) $\rightarrow M^{IF}$			!Put_Ack(LLC) $\rightarrow$ S	
M^{II}						!Put_Ack(LLC) $\rightarrow$ I	RepMemWr !WB_Ack(H_D)
M^{IS}	!Inv(LLC) $\rightarrow S^{IF}$					!Put_Ack(LLC) $\rightarrow$ S	