MODULE *cacheinvalidationv2*

EXTENDS *Naturals*

CONSTANTS
KEYS

VARIABLES
database,
cache,
cacheFillStates, *cacheFillStatus*[*key*] = Fill state
invalidationQueue

INSTANCE *cacherequirements*

$vars \triangleq \langle database, cache, cacheFillStates, invalidationQueue \rangle$

$InvalidationMessage \triangleq [key : KEYS, version : DataVersion]$

$CacheFillState \triangleq [state : \{\text{“inactive”}, \text{“started”}, \text{“respondedto”}\}, version : DataVersion]$

$CacheValue \triangleq CacheMiss \cup CacheHit$

$TypeOk \triangleq$
$\land \quad database \in [KEYS \to DataVersion]$
$\land \quad cache \in [KEYS \to CacheValue]$
We track the cache fill state for each key. It is a multipart process
$\land\ cacheFillStates \in [KEYS \to CacheFillState]$
We model *invalidationQueue* as a set, because we cannot guarentee in-order delivery
$\land\ invalidationQueue \in \text{SUBSET } InvalidationMessage$

$Init \triangleq$
$\land\ database = [k \in KEYS \mapsto 0]$
$\land\ cache = [k \in KEYS \mapsto [type \mapsto \text{“miss”}]]$
Cache fill states start inactive
$\land\ cacheFillStates = [k \in KEYS \mapsto [$
$state \mapsto \text{“inactive”},$
Version set to earliest possible version
$version \mapsto 0]$
$]$
The invalidation queue starts empty
$\land\ invalidationQueue = \{\}$

$DatabaseUpdate(k) \triangleq$
LET $updatedVersion \triangleq database[k] + 1$ IN
The version of that key is incremented, representing a write

$\land\ database' = [database \text{ EXCEPT}$
$\qquad\qquad\qquad\qquad ![k] = updatedVersion]$
Adds invalidation message to queue.
We don't need to model a delay in adding message as the cache can
always delay handling message; to similar effect
$\land\ invalidationQueue' = invalidationQueue \cup$
Add updated data to invalidation message
$\qquad\qquad\qquad\qquad \{[key \mapsto k,\ version \mapsto updatedVersion]\}$
$\land$ UNCHANGED $\langle cache,\ cacheFillStates\rangle$

Cache Fill behavior
$CacheStartReadThroughFill(k) \triangleq$
Read through only occurs when the cache is unset for that value
$\land\ cache[k] \in CacheMiss$
One cache fill request at a time
$\land\ cacheFillStates[k].state =$ "inactive"
$\land\ cacheFillStates' = [cacheFillStates \text{ EXCEPT } ![k].state =$ "started"$]$
$\land$ UNCHANGED $\langle database,\ cache,\ invalidationQueue\rangle$

This is the moment the database provides a value for cache fill
$DatabaseRespondToCacheFill(k) \triangleq$
$\land\ cacheFillStates[k].state =$ "started"
$\land\ cacheFillStates' = [cacheFillStates \text{ EXCEPT}$
$\qquad\qquad\qquad\qquad ![k].state =$ "respondedto",
$\qquad\qquad\qquad\qquad ![k].version = database[k]$
$\qquad\qquad\qquad\qquad]$
$\land$ UNCHANGED $\langle database,\ cache,\ invalidationQueue\rangle$

Cache fails to fill
$CacheFailFill(k) \triangleq$
$\land\ cacheFillStates[k].state =$ "respondedto"
Cache fill state is reset, having not filled cache
$\land\ cacheFillStates' = [cacheFillStates \text{ EXCEPT}$
$\qquad\qquad\qquad\qquad ![k].state =$ "inactive",
$\qquad\qquad\qquad\qquad ![k].version = 0$
$\qquad\qquad\qquad\qquad]$
$\land$ UNCHANGED $\langle database,\ cache,\ invalidationQueue\rangle$

Cache incorporates the data
$CacheCompleteFill(k) \triangleq$
$\land\ cacheFillStates[k].state =$ "respondedto"
Either the cache is empty for that key
$\land\ \lor\ cache[k] \in CacheMiss$
or we are filling a newer version
$\qquad \lor\ \land\ cache[k] \notin CacheMiss$
$\qquad\quad \land\ cache[k].version < cacheFillStates[k].version$

```
    ∧ cacheFillStates′ = [cacheFillStates EXCEPT  Reset to 0
                            ![k].state = "inactive",
                            ![k].version = 0
                          ]
    ∧ cache′ = [cache EXCEPT
                    ![k] = [
                        cache value is now a hit
                        type ↦ "hit",
                        set to whatever came back in response
                        version ↦ cacheFillStates[k].version
                    ]
                ]
    ∧ UNCHANGED ⟨database, invalidationQueue⟩

CacheIgnoreFill(k) ≜
    ∧ cacheFillStates[k].state = "respondedto"
    If we have a newer version in cache, ignore fill
    ∧ ∧ cache[k] ∈ CacheHit
      ∧ cache[k].version ≥ cacheFillStates[k].version
    ∧ cacheFillStates′ = [cacheFillStates EXCEPT  Reset to 0
                            ![k].state = "inactive",
                            ![k].version = 0
                          ]
    Don't update cache
    ∧ UNCHANGED ⟨cache, database, invalidationQueue⟩


CacheHandleInvalidationMessage ≜
    ∧ ∃ message ∈ invalidationQueue :  Deque invalidation queue in any order
        Key must be in cache
      ∧ ∧ cache[message.key] ∈ CacheHit
        Message needs to be newer then the cache
        ∧ cache[message.key].version < message.version
      Update item in cache
      ∧ cache′ = [cache EXCEPT
                    ![message.key] = [
                        type ↦ "hit",
                        Update to version in invalidation message
                        version ↦ message.version
                    ]]
      Remove message from queue because handled
      ∧ invalidationQueue′ = invalidationQueue \ {message}

    ∧ UNCHANGED ⟨cacheFillStates, database⟩

CacheIgnoreInvalidationMessage ≜
```

$\land \exists\, message \in invalidationQueue :$ Deque invalidation queue in any order

Ignore invalidation messages for messages not in cache

$\land \lor cache[message.key] \in CacheMiss$

Or when the cache already has the same or larger version

$\lor \land cache[message.key] \notin CacheMiss$

$\land cache[message.key].version \geq message.version$

Remove message from queue to ignore

$\land invalidationQueue' = invalidationQueue \setminus \{message\}$

Don't update cache

$\land$ UNCHANGED $\langle cacheFillStates,\ database,\ cache\rangle$

Cache eviction model is unchanged

$CacheEvict(k) \triangleq$

$\land cache[k] \in CacheHit$

$\land cache' = [cache$ EXCEPT $![k] = [type \mapsto$ "miss"$]]$

$\land$ UNCHANGED $\langle database,\ cacheFillStates,\ invalidationQueue\rangle$

Fairness: Normally no operation is guarenteed to happen, it just may. however that means, for example, that the cache could just stop reading forever. And so it would never update. Now that doesn't seem reasonable.

$CacheFairness \triangleq$

$\exists\, k \in KEYS :$

Cache fill process will be allowed to complete

$\lor CacheStartReadThroughFill(k)$

$\lor DatabaseRespondToCacheFill(k)$ Write

$\lor CacheCompleteFill(k)$

$\lor CacheIgnoreFill(k)$

Cache will be allowed to process invalidation messages

$\lor CacheHandleInvalidationMessage$

$\lor CacheIgnoreInvalidationMessage$

Specification

$Next \triangleq$

$\exists\, k \in KEYS :$

Database states

$\lor DatabaseUpdate(k)$

Cache states

$\lor CacheStartReadThroughFill(k)$

$\lor DatabaseRespondToCacheFill(k)$

$\lor CacheCompleteFill(k)$

$\lor CacheIgnoreFill(k)$

$\lor CacheHandleInvalidationMessage$

$\lor$ *CacheIgnoreInvalidationMessage*
$\lor$ *CacheEvict*(k)

Cache fairness is included as part of the specification of system behavior.
This is just how the system works.

$Spec \triangleq Init \land \Box[Next]_{vars} \land \mathrm{WF}_{vars}(CacheFairness)$

* Modification History
* Last modified *Wed Jun* 15 12:49:34 *MST* 2022 by *elliotswart*
* Created *Tue Jun* 14 20:36:02 *MST* 2022 by *elliotswart*