MODULE *facebookcacheinvalidation*

EXTENDS *Naturals*

CONSTANTS
KEYS

VARIABLES
database,
Represents *metadata* version stored in the cache
cache,
Represents the version stored in the cache. This is what is used for comparisons,
to allow our model to decouple ACTUAL *metadata* version with STORED version
cacheVersions,
cacheFillStates,
invalidationQueue,
counter

We can still test with the same cache requirements we've been using this whole time
INSTANCE *cacherequirements*

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

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

$CacheFillState \triangleq [$
$state : \{$
"inactive",
"startfillmetadata",
"respondedtometadata", Next: *CacheFillMetadata*
"startfillversion",
"respondedtoversion" Next: *CacheFillVersion*
$\},$
$version : DataVersion]$

$CacheValue \triangleq CacheMiss \cup CacheHit$

$TypeOk \triangleq$
$\land\ database \in [KEYS \to DataVersion]$
$\land\ cache \in [KEYS \to CacheValue]$
Cache versions are typed identically to cache
$\land\ cacheVersions \in [KEYS \to CacheValue]$
$\land\ cacheFillStates \in [KEYS \to CacheFillState]$
$\land\ invalidationQueue \in$ SUBSET $InvalidationMessage$
$\land\ counter \in Nat$

$Init \triangleq$
$\land\ database = [k \in KEYS \mapsto 0]$
cache ($metadata$) and $cacheVersions$ start empty together
$\land\ cache = [k \in KEYS \mapsto [type \mapsto$ "miss"$]]$
$\land\ cacheVersions = [k \in KEYS \mapsto [type \mapsto$ "miss"$]]$

$\land\ cacheFillStates = [k \in KEYS \mapsto [$
$state \mapsto$ "inactive",
$version \mapsto 0]$
$]$
$\land\ invalidationQueue = \{\}$
$\land\ counter = 0$

$DatabaseUpdate(k) \triangleq$
LET $updatedVersion \triangleq database[k] + 1$IN
$\land\ database' = [database$ EXCEPT
$![k] = updatedVersion]$
$\land\ invalidationQueue' =$
$invalidationQueue \cup$
$\{[key \mapsto k,\ version \mapsto updatedVersion]\}$
$\land$ UNCHANGED $\langle cache,\ cacheVersions,\ cacheFillStates,\ counter\rangle$

$CacheStartFillMetadata(k) \triangleq$
Fill only occurs if the cache is unset for that value
$\land\ cache[k] \in CacheMiss$
$\land\ cacheFillStates[k].state =$ "inactive"
$\land\ cacheFillStates' = [cacheFillStates$ EXCEPT $![k].state =$ "startfillmetadata"$]$
$\land$ UNCHANGED $\langle database,\ cache,\ cacheVersions,$
$invalidationQueue,\ counter\rangle$

$DatabaseRespondWithMetadata(k) \triangleq$
$\land\ cacheFillStates[k].state =$ "startfillmetadata"
$\land\ cacheFillStates' = [cacheFillStates$ EXCEPT
$![k].state =$ "respondedtometadata",
$![k].version = database[k]$
$]$
$\land$ UNCHANGED $\langle database,\ cache,\ cacheVersions,$
$invalidationQueue,\ counter\rangle$

Metadata updated in cache
$CacheFillMetadata(k) \triangleq$
$\land\ cacheFillStates[k].state =$ "respondedtometadata"
$\land\ cacheFillStates' = [cacheFillStates$ EXCEPT
$![k].state =$ "inactive"

```
                        ]
    Represents cache metadata being updated
    Does not check version
    ∧ cache' = [cache EXCEPT
                    ![k] = [
                        type ↦ "hit",
                        version ↦ cacheFillStates[k].version
                    ]
                ]
    ∧ UNCHANGED ⟨database, cacheVersions, invalidationQueue, counter⟩

CacheStartFillVersion(k) ≜
    Fill only occurs if the cacheVersion is unset for that value
    ∧ cacheVersions[k] ∈ CacheMiss
    ∧ cacheFillStates[k].state = "inactive"
    ∧ cacheFillStates' = [cacheFillStates EXCEPT ![k].state = "startfillversion"]
    ∧ UNCHANGED ⟨database, cache, cacheVersions,
                 invalidationQueue, counter⟩

DatabaseRespondWithVersion(k) ≜
    ∧ cacheFillStates[k].state = "startfillversion"
    ∧ cacheFillStates' = [cacheFillStates EXCEPT
                            ![k].state = "respondedtoversion",
                            ![k].version = database[k]
                         ]
    ∧ UNCHANGED ⟨database, cache, cacheVersions,
                 invalidationQueue, counter⟩

 Version updated in cache
CacheFillVersion(k) ≜
    ∧ cacheFillStates[k].state = "respondedtoversion"
    Fill empty versions
    ∧   ∨ cacheVersions[k] ∈ CacheMiss
        or newer versions
        ∨ ∧ cacheVersions[k] ∈ CacheHit
          ∧ cacheVersions[k].version < cacheFillStates[k].version

    ∧ cacheFillStates' = [cacheFillStates EXCEPT
                            ![k].state = "inactive"
                         ]
    Represents cache versions being updated
    ∧ cacheVersions' = [cacheVersions EXCEPT
                            ![k] = [
                                type ↦ "hit",
                                version ↦ cacheFillStates[k].version
```

```
                                ]
                            ]
    ∧ UNCHANGED ⟨database, invalidationQueue, cache, counter⟩

CacheIgnoreFillVersion(k) ≜
    ∧ cacheFillStates[k].state = "respondedtoversion"
     If we have a newer version in cache, ignore fill
    ∧ ∧ cacheVersions[k] ∈ CacheHit
      ∧ cacheVersions[k].version ≥ cacheFillStates[k].version
    ∧ cacheFillStates' = [cacheFillStates EXCEPT   Reset to 0
                            ![k].state = "inactive",
                            ![k].version = 0
                          ]
    ∧ counter' = counter + 1
    ∧ UNCHANGED ⟨cache, cacheVersions,
                 database, invalidationQueue⟩


CacheFailFill(k) ≜
    ∧ cacheFillStates[k].state ∈ {"respondedtometadata", "respondedtoversion"}

    ∧ cacheFillStates' = [cacheFillStates EXCEPT
                            ![k].state = "inactive",
                            ![k].version = 0
                          ]
    ∧ counter' = counter + 1
    ∧ UNCHANGED ⟨database, cache, cacheVersions,
                 invalidationQueue⟩

CacheEvict(k) ≜
    ∧ cache[k] ∈ CacheHit
    ∧ cacheFillStates[k].state = "inactive"
    ∧ cache' = [cache EXCEPT ![k] = [type ↦ "miss"]]
    ∧ cacheVersions' = [cache EXCEPT ![k] = [type ↦ "miss"]]
    ∧ counter' = counter + 1
    ∧ UNCHANGED ⟨database, cacheFillStates,
                 invalidationQueue⟩
```

Invalidation message handling

```
UpdateFromInvalidationMessage ≜
    ∃ message ∈ invalidationQueue :
          Can update with no version
      ∧ ∨ ∧ cache[message.key] ∈ CacheHit
          ∧ cacheVersions[message.key] ∈ CacheMiss
          or with greater or equal version
        ∨ ∧ cacheVersions[message.key] ∈ CacheHit
```

$\land\ cacheVersions[message.key].version \leq message.version$

Kills pending fill request

$\land\ cacheFillStates[message.key].state = \text{“inactive”}$

Unlike fills from the database, the invalidation message contains both version and *metadata*.

$\land\ cache' = [cache \text{ EXCEPT}$
$\quad ![message.key] = [$
$\quad\quad type \mapsto \text{“hit”},$
$\quad\quad version \mapsto message.version$
$\quad]]$
$\land\ cacheVersions' = [cache \text{ EXCEPT}$
$\quad ![message.key] = [$
$\quad\quad type \mapsto \text{“hit”},$
$\quad\quad version \mapsto message.version$
$\quad]]$
$\land\ invalidationQueue' = invalidationQueue \setminus \{message\}$
$\land\ \text{UNCHANGED}\ \langle cacheFillStates,\ database,\ counter\rangle$

$FailUpdateInvalidationMessageEvictKey\ \triangleq$
$\exists\, message \in invalidationQueue :$
Can update with no version
$\land\ \lor\ \land\ cache[message.key] \in CacheHit$
$\quad\ \land\ cacheVersions[message.key] \in CacheMiss$
or with greater version
$\quad\ \lor\ \land\ cacheVersions[message.key] \in CacheHit$
$\quad\quad\ \land\ cacheVersions[message.key].version < message.version$
Kills pending fill request
$\land\ cacheFillStates[message.key].state = \text{“inactive”}$
Key is evicted from cache, to allow fresh cache fill
$\land\ cache' =$
$\quad [cache \text{ EXCEPT}$
$\quad\quad ![message.key] = [type \mapsto \text{“miss”}]]$
$\land\ cacheVersions' =$
$\quad [cacheVersions \text{ EXCEPT}$
$\quad\quad ![message.key] = [type \mapsto \text{“miss”}]]$
$\land\ invalidationQueue' = invalidationQueue \setminus \{message\}$
$\land\ \text{UNCHANGED}\ \langle cacheFillStates,\ database,\ counter\rangle$

$FailUpdateInvalidationMessageIgnore\ \triangleq$
$\exists\, message \in invalidationQueue :$
If message version is lower or equal than cache version, do nothing
$\land\ cacheVersions[message.key] \in CacheHit$
$\land\ cacheVersions[message.key].version \geq message.version$

$\land\ counter' = counter + 1$
$\land\ invalidationQueue' = invalidationQueue \setminus \{message\}$
$\land$ UNCHANGED $\langle cacheFillStates,\ database,$
$cache,\ cacheVersions\rangle$

$IgnoreInvalidationMessage\ \triangleq$
$\exists\, message \in invalidationQueue :$
Ignore invalidation messages if a key is not in cache
$\land\ \lor\ \land\ cache[message.key] \in CacheMiss$
and a fill is not occurring
$\land\ cacheFillStates[message.key].state =$ "inactive"
or when the cache already has a larger version
$\lor\ \land\ cacheVersions[message.key] \in CacheHit$
$\land\ cacheVersions[message.key].version > message.version$
$\land\ invalidationQueue' = invalidationQueue \setminus \{message\}$
$\land\ counter' = counter + 1$
Don't update cache
$\land$ UNCHANGED $\langle cacheFillStates,\ database,\ cache,\ cacheVersions\rangle$

$CacheFairness\ \triangleq$
$\lor \exists\, k \in KEYS :$
$\lor\ CacheStartFillMetadata(k)$
$\lor\ DatabaseRespondWithMetadata(k)$
$\lor\ CacheFillMetadata(k)$
$\lor\ CacheStartFillVersion(k)$
$\lor\ DatabaseRespondWithVersion(k)$
$\lor\ CacheFillVersion(k)$
$\lor\ CacheIgnoreFillVersion(k)$
$\lor\ UpdateFromInvalidationMessage$
$\lor\ FailUpdateInvalidationMessageEvictKey$
$\lor\ FailUpdateInvalidationMessageIgnore$
$\lor\ IgnoreInvalidationMessage$

Specification

$Next\ \triangleq$
$\lor \exists\, k \in KEYS :$
Database states
$\lor\ DatabaseUpdate(k)$
Cache states
$\lor\ CacheStartFillMetadata(k)$
$\lor\ DatabaseRespondWithMetadata(k)$
$\lor\ CacheFillMetadata(k)$
$\lor\ CacheStartFillVersion(k)$
$\lor\ DatabaseRespondWithVersion(k)$

$\lor$ $CacheFillVersion(k)$
$\lor$ $CacheIgnoreFillVersion(k)$
$\lor$ $CacheEvict(k)$
$\lor$ $CacheFailFill(k)$
$\lor$ $UpdateFromInvalidationMessage$
$\lor$ $FailUpdateInvalidationMessageEvictKey$
$\lor$ $FailUpdateInvalidationMessageIgnore$
$\lor$ $IgnoreInvalidationMessage$

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

$CounterBound \triangleq counter \leq 2$

* Modification History
* Last modified *Thu Jun* 16 16:19:54 *MST* 2022 by *elliotswart*
* Created *Tue Jun* 14 20:36:02 *MST* 2022 by *elliotswart*