lockopt code review

Michael Davidsaver

June 25, 2015

Goals

- Enhance concurrency of EPICS Process Database
- Multi-locking
  - Allow locking of several lock sets
  - Like a temporary DB_LINK
  - Allow atomic get/put to an arbitrary set of records in one process
- Reduce contention
  - Eliminate global locks
  - Reduce coupling of otherwise independent scan threads
  - lockSetModifyLock and timeListLock biggest offenders

Problems

- Dynamic DB_LINK re-targetting
  - Records are the entry point
  - Association between Records and lock set can change
- Recursive dbScanLock()
- dbLockSetGblLock() and dbPutFieldLink
  - Solved w/ link parsing change

Design Considerations

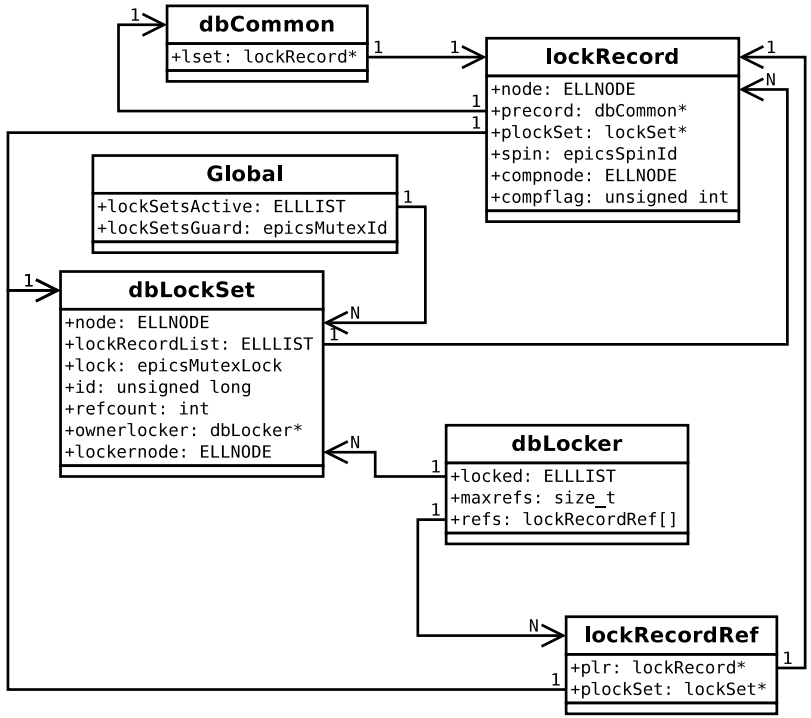
- dbScanLock()
  - Hot code path, must be fast
- DB_LINK re-targetting
  - Assumed to be occasional
  - Shouldn't have global effects
  - Avoid temperary allocation during lock set split operation
- Multi-locking
  - Deterministic?

Algorithm Choices

- Multi-locking possibilities
- Deadlock detection
  - Ownership tracking
  - Uses owner priority breaks deadlocks
  - Loser unlocks and re-tries
  - Requires disable preemption? (all existing implimentations)
- Global ordering
  - Pre-defined locking order
  - Pointer address (or counter/creation time)
  - Rollover?
    - New lock is in the middle of the order
- Investigated deadlock detection first, but ultimately used global ordering

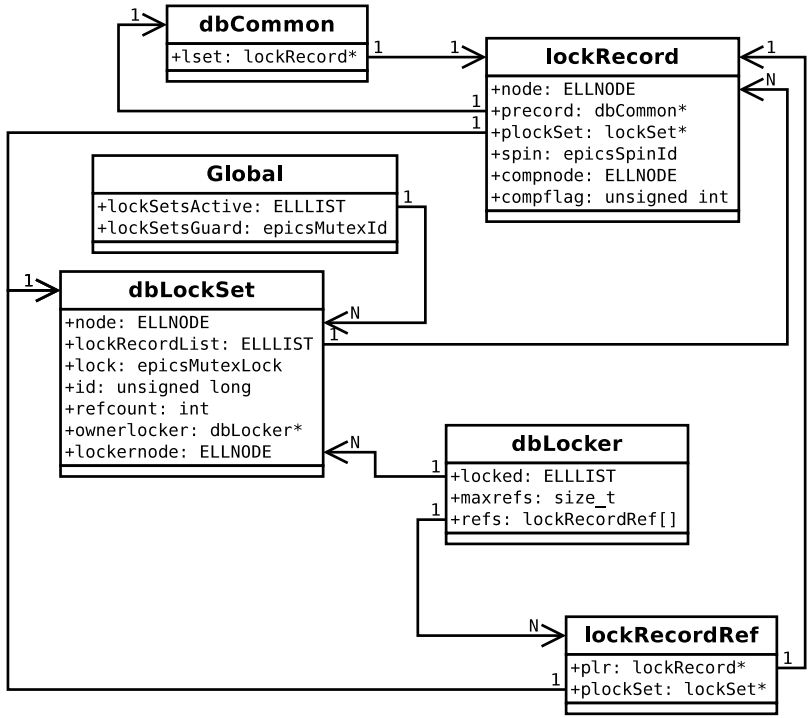
dbScanLock(dbCommon*)

- Lock a single record/lock set
- Entry point is **dbCommon***
- Assoc. with **dbLockSet*** may change
- Must traverse this assoc. to lock



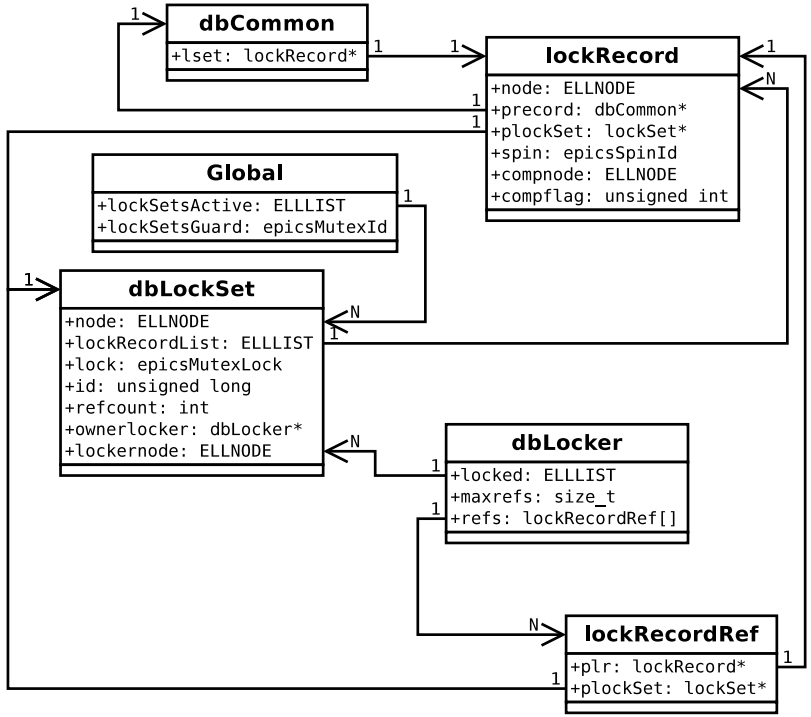
dbScanLock(dbCommon*)

- At present lockSetModifyLock guards assoc. of all records
- Replace with per-record spinlock
- But can't lock mutex while holding spinlock



dbScanLock(dbCommon*)

- At present lockSetModifyLock guards assoc. of all records
- Replace with per-record spinlock
- But can't lock mutex while holding spinlock
- Add a reference counter to dbLockSet



dbLockSet association rules

- Association between lockRecord and dbLockSet
  - dbLockSet* lockRecord::plockSet
- Guarded by spinlock (lockRecord) and mutex (dbLockSet)
- Read
  - Either is locked
- Change
  - Both are locked

dbScanLock(dbCommon *precord)

```
    int cnt;
    lockRecord *lr = precord->lset;
    lockSet *ls = dbLockGetRef(lr);
retry:
    epicsMutexMustLock(ls->lock);
    epicsSpinLock(lr->spin);
    if(ls!=lr->plockSet) {
        lockSet *ls2 = lr->plockSet;
        epicsAtomicIncrIntT(&ls2->refcount);
        epicsSpinUnlock(lr->spin);
        epicsMutexUnlock(ls->lock);
        dbLockDecRef(ls);
        ls = ls2;
        goto retry;
    }
    epicsSpinUnlock(lr->spin);
    epicsAtomicDecrIntT(&ls->refcount);
```

dbLocker operations

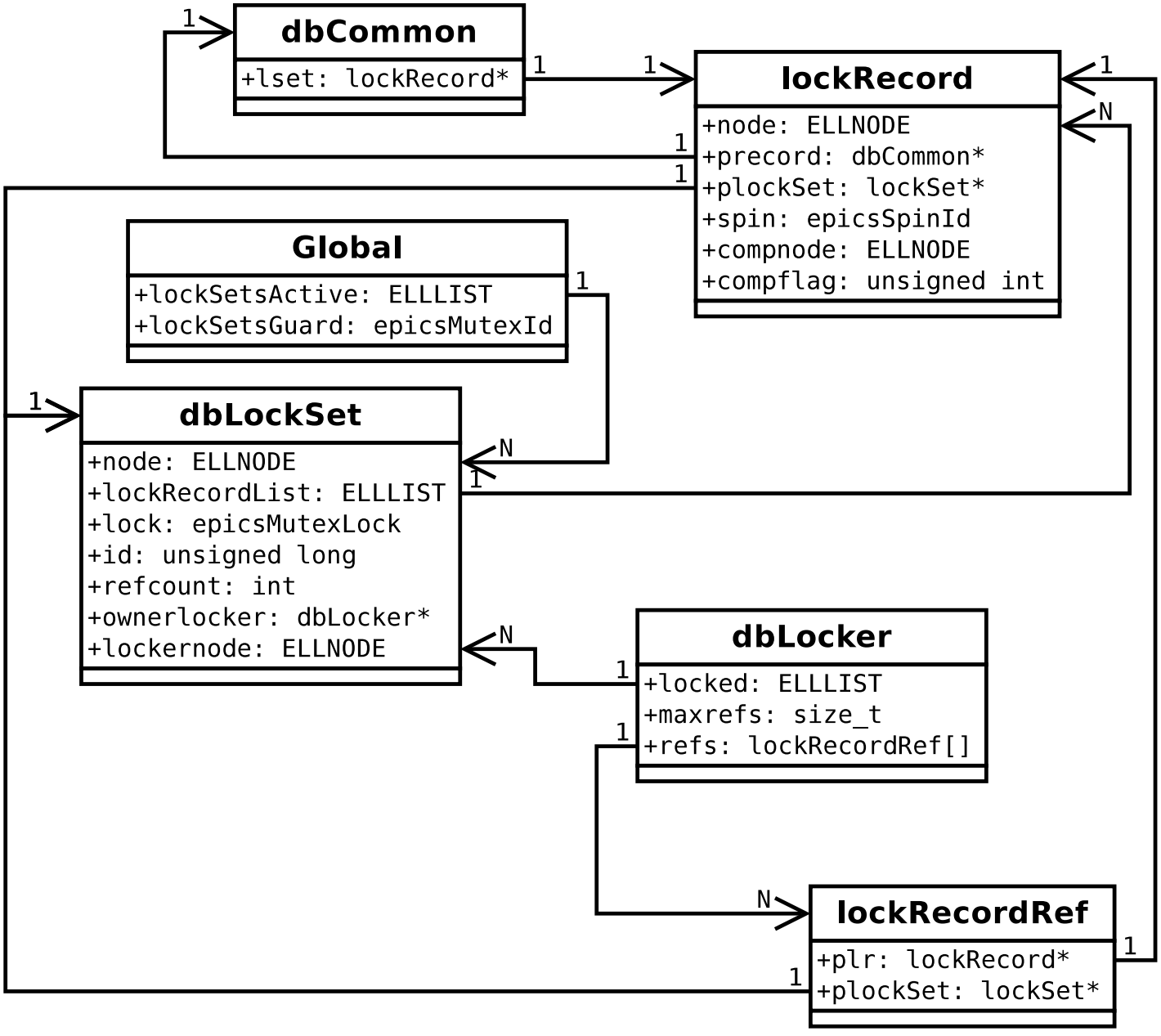
- Functions
  - dbLockerAlloc(struct dbCommon *precs[], ...
  - dbLockerFree(dbLocker *)
  - dbScanLockMany(dbLocker*)
  - dbScanUnlockMany(dbLocker*)
- Operates on a list/array of records
- Maintains/locks records in sorted order
  - By increasing dbLockSet*
  - qsort() [opporunity for improvement]
- Keeps a cache of record to lockset association

dbScanLockMany(dbLocker*)

1. Check cache of lockRecord to dbLockSet associations
   1.1 Cache is lockRecordRef::plockSet
   1.2 Uses global counter as optimization to detect if no associations changed
2. Lock all cached dbLockSets
3. Check that associations didn't change
   3.1 Unlock and retry if any did

- TODO: Check each assoc. as soon as locked

dbScanLockMany(dbLocker*)



Reference counting rules

- int dbLockSet::refcount
- Owners increment by one
  - +1 for lockRecord
    - dbLockSet* lockRecord::plockSet
  - +1 for dbLocker cache
    - dbLockSet* lockRecordRef::plockSet
  - +1 for dbLocker locked
    - ELLIST dbLocker::locked

dbLockSet merge/split

- Merge two lock sets
  - Lock both with dbScanLockMany()
  - Concatinate dbLockSet::lockRecordList. O(0)
  - Inner loop to lock each spinlock to change assoc. O(N)
- Split one lock set
  - Remove one DB_LINK between two records
  - Partitioned? (graph)
  - Reverse link tracking (dest. to src.)
  - Traverse the graph. O(N)