Scribe notes 3-27-12

Last Time

Last time we looked at building a set using linked list implemented by 4 types of locking

There will be 3 new types of locking we will look at

1. Optimistic Synchronization
2. Lazy Synchronization
3. Lock Free Synchronization

What are the downsides of fine grain locking?

- It uses a lot of locks
- Searching needs to acquire locks, this is especially bad because programs tend to read more than they write

What do we want to achieve with these new types of locks?

- We want to free readers from having to acquire locks
- Eliminate having one lock act as a bottle neck

Slide 84

When the data structure is actually modified is when the locks are acquired.

Before committing a change make sure that everything you did was okay. I.e. Validation phase

Slides 85->87

In order to add c. the blue thread needs to acquire the locks on b and d and then switch the pointers.

Slides 88 -> 94

These slides show what can go wrong if the orange thread tries to delete b while the blue thread is in the process of adding c to the linked list.

If the blue thread adds c and then the orange thread comes and deletes b then c is unreachable from the head.

Slide 95

In order to prevent this the blue thread needs to enter its validate phase and make sure that adding c worked by:

1. Checking that c is still reachable from the head (shown on this slide or slide 107)

2. Check that c still points to d (shown on slide 101 or 108)

Slide 97

The orange slide could also make add b' which would violate the second check in the validation phase.

Slide 102

Linearization Point: the exact moment when an action is finished

Slide 109

Return false means c does not exist

Slide 115

We would like to minimize the need for 2 traversals and make sure that out list will perform better and fail infrequently

Optimistic list is not wait-free for add/remove

Optimistic list is lock-free

Optimistic list has global progress

Slide 116

The need to traverse the list twice means that the optimistic list doesn't scale.

Slide 118

Lazy synchronization:

Removal needs to identify the node to remove and swing the pointers

If a node needs to be removed, the thread only marks 1 bit which marks the node as deleted. It becomes someone else's job to remove the node and swing the pointers.

By doing this we eliminate the need for scanning the list twice.

Slide 119

How to do the physical delete is a design decision.

Slide120 ->124

Green means is present

Red means is deleted

Slide126

If validation fails then that thread needs to fix everything up.

Slide134

A bit can be set atomically where as a pointer cannot be switched atomically since it requires a read and a write.

Slide 136

While doing a search through the list the reader can move the pointer as it traverses the list when it encounters a node which is marked as deleted.

Slide 140

Question: Can we get add and remove to be lock free?

If yes then we have a lock free data structure.