Lecture 20

Note Title 3/27/2008

No class Tues April 1.

Optimistic Concurrency control is to use timestamp version numbers on objects and detect conflicts at ~~tact~~ commit time

read(d_1, v_1, t_1)
read(d_2, v_2, t_2)
write(d_3, v_1+v_2, t_3)

at end check that d_1 still has version t_1, d_2 has t_2
d_3 has t_3

Why?

Obstruction-free: a thread executing in isolation makes progress.
- livelock: threads continually abort each other and system makes no progress.

- advantage: it's allowable for any xact to abort any other xact at any time.

- Idea for implementation: "wrap" an ordinary object in a transactional object by holding a reference to the ordinary object.

To use such an Object, we "open" it

to create:
```
Counter    counter = new Counter(0)
TMObject   myCounter = new TMObject(counter)
```

to use:
```
Counter    localCounter = (Counter) myCounter.open(WRITE)
```

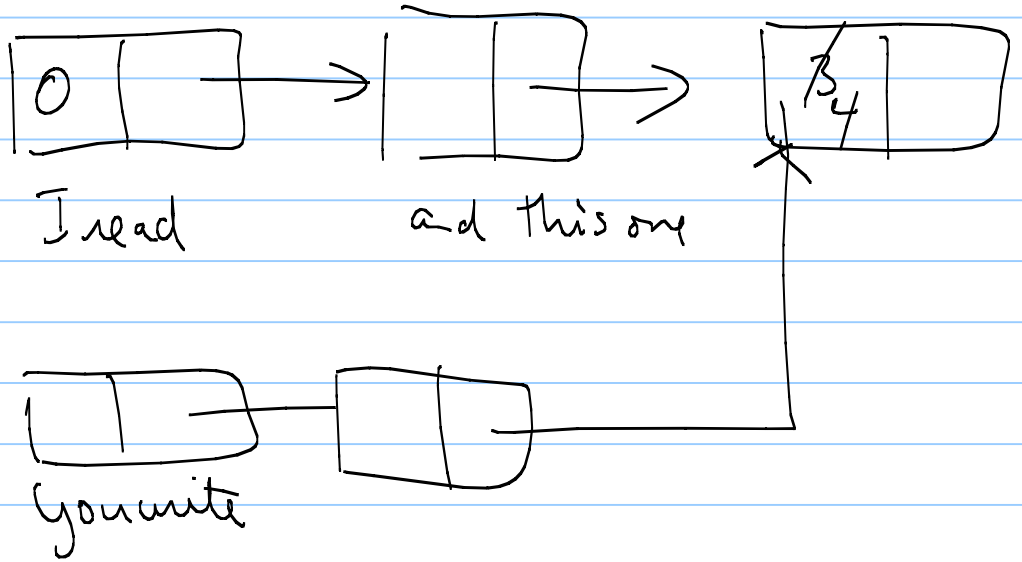
localCounter is a copy

localCounter.inc()

commitTransaction() — does tests like those I described for opt. conc. control.

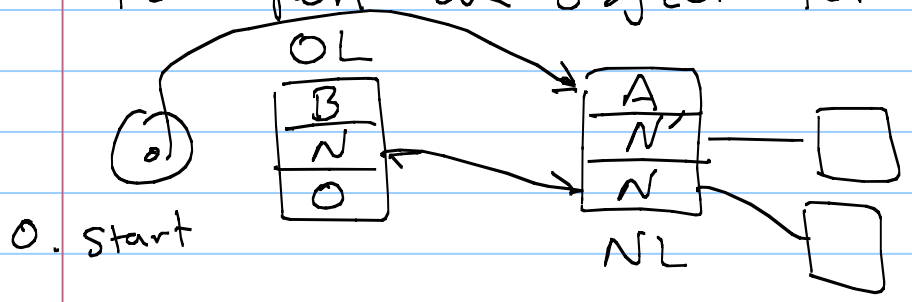
Refer to Figures 1 and 2 in the paper for how to use the DSTM primitives.

Why does consistency have to be checked at each open()? Otherwise a transaction (that would have to be aborted) could read conflicting values committed by different transactions; this could lead to null pointer dereference or other nastiness; so the decision is made to abort as soon as the need is



Refer to Figure 3 for the representation of transactional objects and figures 4 and 5 for what happens when an object is opened for writing in the two cases that the most recent opener for write has committed or aborted. If the most recent opener is still active then we either need to abort it or wait awhile and try again (contention policy management decision).

To open an object for writing in transaction A



OL = start

if OL.B. committed {

NL.old = OL.new

NL.new = OL.new.clone()

CAS (start, OL, NL)

else if OL.B. aborted

NL.old = OL.old

NL.new = OL.old.clone()

CAS(start, OL, NL)

else // OL.B is active

For read:

OL = start

if OL.B.active : B.abort

else if OL.B.committed

v = OL.new

else v = OL.old

curr Thread.vtable +=

(O , v)

abort B

Validation at commit or open:

- for read objects: are all the (o, v) pairs still current
- for written objects — check if current Xact has been aborted.

Next Thursday

Composable Memory Transactions

Harris, Marlow, Peyton-Jones, Herlihy

Refer to Figure 6 for performance analysis. The top graph is not very interesting in my opinion. The most striking thing about the lower graph is how much, much, (much*) worse in ALL of the concurrent versions are than the single-threaded test case.

The lesson that I take is that performance improvement from parallelism is doomed when operations on shared data represent too large a fraction of the total operations performed, regardless of whether one uses locking or STM methods. It is interesting that the STM methods do perform better than the locking methods beginning at a modest level of concurrency.