Lecture 17: Logging

Hari Balakrishnan
6.033 Spring 2015

Based on Section 9.3 and some material from Sam Madden
Transaction example

GET(x) – read value of x from store (database)
PUT(x,v) – write v to x in store (database)

xfer(F, T, amt):
    PUT(F, GET(F) – amt)
    PUT(T, GET(T) + amt)

tid = BEGIN_TRANSACTION
xfer(from, to, amount)
if read(from) < 0:
    print “Not enough funds”
    ABORT
else:
    COMMIT
    “Do it all”
Assumption for today

No concurrent transactions

Focus on crash recovery and ABORT to implement all-or-nothing atomicity and durability for transactions
Log

Append-only data structure: NEVER OVERWRITE OR ERASE!

<table>
<thead>
<tr>
<th>type:</th>
<th>CHANGE</th>
<th>type:</th>
<th>OUTCOME</th>
<th>type:</th>
<th>CHANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>tid:</td>
<td>9979</td>
<td>tid:</td>
<td>9974</td>
<td>tid:</td>
<td>9979</td>
</tr>
<tr>
<td>redo_action:</td>
<td>new: 90</td>
<td>status:</td>
<td>COMMITTED</td>
<td>redo_action:</td>
<td>new: 40</td>
</tr>
<tr>
<td></td>
<td>PUT(debit_account, $90)</td>
<td></td>
<td></td>
<td></td>
<td>PUT(credit_account, $40)</td>
</tr>
<tr>
<td>undo_action:</td>
<td>old: 120</td>
<td></td>
<td></td>
<td>undo_action:</td>
<td>old: 10</td>
</tr>
<tr>
<td></td>
<td>PUT(debit_account, $120)</td>
<td></td>
<td></td>
<td></td>
<td>PUT(credit_account, $10)</td>
</tr>
</tbody>
</table>

tid : “transaction identifier”, aka “action identifier”
GET (read) with just the log

GET(x):  # global log
    commits = {}
    for record r in log[len(log)-1] .. log[0]:
        if (r.status == COMMITTED):
            commits = commits + r.tid
        if (r.type == CHANGE) and
           (r.tid in commits) and
           (r.var == x):
            return r.new_val
GET (read) your own PUTs (writes)

GET(x):
    commits = {}  
    for record r in reversed(log):  # backward scan
        if (r.status == COMMITTED):
            commits = commits + r.tid
        if (r.type == CHANGE) and
            (r.tid in commits or r.tid=cur_tid) and
            (r.var == x):
            return r.new_val

+ Crash recovery is fast! Don’t have to do anything
+ PUTs are fast! Just append to log
- GETs are SLOW: have to scan log backwards
Cell Storage + Log

Append-only data structure: NEVER OVERWRITE OR ERASE!

WRITE_NEW_VALUE

READ_CURRENT_VALUE
Read / write with cell storage

GET(x):
    return cell_read(var)

PUT(x, value):
    old_x = GET(x)  WRONG!
    cell_write(x, value)
    log.append(tid, CHANGE,
                x, old=old_x, new=value)

WRITE-AHEAD LOGGING (WAL)

PUT(x, value):  LOG TO STABLE STORAGE FIRST
    log.append(tid, CHANGE,
                x, old=read(x), new=value)
    cell_write(x, value)
1. Volatile cell writes (in-mem DB)

Recovering cell storage from log

```python
recover(log):
    done = {}
    for record r in reversed(log):  # backward scan
        if r.status == COMMITTED:
            winners = winners + r.tid
    for record r in log:  # forward scan
        if r.type == CHANGE and r.tid in winners:
            cell_write(r.var, r.new_val)  # redo
```
2. Non-volatile cell writes: Recovering cell storage from log

```python
recover(log):
    winners = {}
    for record r in reversed(log):  # backward scan
        if r.status == COMMITTED:
            winners = winners + r.tid
        if r.type == CHANGE and r.tid not in winners:
            cell_write(r.var, r.old_val)  # undo
```
3. Cached read / write

GET(x):
    if x not in cache:
        # may evict another from cache to cell store
        cache[x] = cell_read(x)
    return cache[x]

PUT(x, value):
    log.append(cur_tid, CHANGE,
                x, old=read(x), new=value)
    # may evict another from cache to cell store
    cache[x] = value
3. Recovery for cached database

recover(log):
    done = {}
    for record r in reversed(log):  # backward scan
        if r.type == COMMITTED:
            done = done + r.tid
        if r.type == CHANGE and r.tid not in done:
            cell_write(r.var, r.old_val)  # undo
    for record r in log:  # forward scan
        if r.type == CHANGE and r.tid in done:
            cell_write(r.var, r.new_val)  # redo
Abort (all three cases)

abort():  # ABORT current transaction, cur_tid
for record r in reversed(log)
  if (r.tid == cur_tid)
    if r.type == CHANGE:
      PUT(r.var, r.old_val)  # undo
    if r.type == BEGIN
      break
log.append(cur_tid, ABORTED)  # optional
# to avoid undo’ing an already-aborted transaction