Modern day OLTP:

Main memory problem
No-disk stalls in a Xact
Do not allow user-stalls in a Xact (Aunt Millie will go out for lunch)
--- hence no stalls.

A heavy Xact is 200 record touches – less than 1 Msec.

Why not run Xact to completion – single threaded! No latches, no issues, no nothing. Basically TS order !!!

Problem: multiprocessor support – we will come back to this.

Ok to ask for Xact classes in advance (no ad-hoc updates in an OLTP system).

Look at the xacts classes…..

They might commute: if so run with no locking

They might never conflict – if so run with no locking.

Might be only two classes that conflict (Ti and Tj). Run everybody else with no controls. Serialize Ti and Tj (with timestamp techniques or something else)

If a transaction is alive for nanoseconds (processor transactional memory) or microseconds (modern OLTP), then interesting to rerun Carey simulations (which assumed disk not main memory).

Contracts/Saga

Vacation in San Diego

T1: get a plane ticket
T2: get a hotel
T3: get a rental car
T4: tickets to San Diego zoo

Oops – get sick – can't go. Want to "unwind" whole "workflow". Want something bigger than a Xact which can be reversed. Notion of Sagas and Contracts. Need compensation actions, which will reverse a xact. Can't abort after a commit.

Crash recovery
Never lose my data ever. Surest recipe to get fired on the spot.

Scenarios:

1) transaction aborts (back him out)
2) transaction deadlocks, and is picked as a victim (ditto)
3) transaction violates an integrity constraint (ditto)

OS fails (rule of thumb – MVS crashes once a year, Linux once a month, Windows once a week or more) -- reload OS, reload DBMS, undo losers, redo winners

DBMS fails

bohrbugs (repeatable). These are knocked out quickly by a good QA process. If you are buying a DBMS, get clear on how serious the vendor is about QA (typically not very) -- don't run a DBMS in production until it is "mature" -- like 1-2 years after release

heisen bugs (not repeatable) timing problems, race conditions, … Unbelievably hard to find. Usually put engineers on airplanes.

Disk crash: modern disks fail every 5 years or so. Usually start to see disk errors (redo reads or writes in advance). In any case, take a dump periodically, weekly full with daily partials. Must roll forward from the last partial, redoing history using a log.

Bad, bad, bad, bad: unrecoverable failures (corrupted the log) -- "up the creek"

App fails:
Not an issue in this class

Comm. Failures:
We will come back to these when we deal with multi-processor issues

Disaster -- Machine room fails (fire, flood, earthquake, 9/11, …)

1970's solution to disasters:

Write a log (journal) of changes
Spool the log to tape

Hire iron mountain to put the tapes under the mountain
Buy IBM hardware – they were heroic in getting you back up in small numbers of days

1980's solution
Hire Comdisco to put the tapes at their machine room in Atlanta
Send your system programmers to Atlanta, restore the log tapes, divert comm to Atlanta
Back up in small numbers of hours
(average CIO conducts a disaster drill more than once a year)

2000's solution (for some)
Run a dedicated "hot standby" in Atlanta
Fail over in seconds to a few minutes

Driven by plummeting cost of hardware and the increasing cost of downtime (thousands of dollars per minute)

In most cases, you tend to lose a few transactions. Too costly to lower the probability to zero. Write disgruntled users a check!!!

Disk-intact recovery

1) undo uncommitted transactions
2) redo committed transactions

write this stuff in a log

depends on buffer pool tactics.

Buffer pool implements "steal" – can write to disk a dirty page of an uncommitted xact – when it needs the slot for something else. All systems do this. Requires the before image in a log to perform undo

Buffer pool implements "no force" – do not require dirty blocks to be forced at commit time – takes too long. Everybody does this. Requires the after image to perform redo

Hence write (before image, after image) in a log. Must write the log before writing the data. Otherwise screwed. Hence WAL.

Options for the log:

Can use a physical log. Whenever the bits change, write a log record.

Insert could cause logging of 4K worth of data

Can use a logical log – record the SQL command (nobody does this – too slow -- and won't work for undo)

Can use something in between – e.g. insert (record) on page-X

Can log B-tree updates
Physical: means 8K bytes for a page splitter
Logical: do nothing – side effect of SQL
In between: insert (key, block – X)

Most of these have been used at one time or another.

6.830 / 6.814 Database Systems

Fall 2010