15441 Paper Summaries, November 11, 2001

SUMAN KUMAR NATH(sknath@cs.cmu.edu)

On Optimistic Methods for Concurrency Control

- *One-line Summary* : Instead of using locks, optimistically do transaction then check at end if can commit results, and retry if not.
- *Main Points*
  - Locking is bad because (i) lock maintenance has high overhead, (ii) lack of general-purpose deadlock free locking protocols, (iii) concurrency is significantly lowered when it is necessary to leave congested node locked while accessing other nodes in disk, and (iv) occasional need to hold on to locks until end of transaction etc.
  - *Optimistic Idea* :
    - Reading a value/pointer can never cause loss of integrity; so reads are completely unrestricted, although returning a value from a query is considered equivalent to a write.
    - Writes are severely restricted. Have read phase, validation phase, then write phase. During read, all writes are on local copies. Validation ensures consistency across active transactions. Write phase is when local copies get written to global DB.
  - *Concurrency control* procedures maintain sets of objects accessed by transaction. Maintain read and write set for transaction.
  - *Serial equivalence* : if there are individual transactions that are executed concurrently, have serial equivalence if there exists some serial sequence of the transactions that produce the final data structure. serial equivalence is sufficient (but not necessary) for integrity of DB.
  - *Validation* of serial equivalence: assign each transaction a transaction number (TN), and explicitly force serializability. Transactions don't interfere if satisfy any of the three:
    1. t_1 finishes before t_2 starts.
    2. t_1 doesn't write anything t_2 reads and t_1 finishes writing before t_2 starts.
    3. t_1 doesn't write anything t_2 reads or writes and t_1 starts validating before t_2 starts validating.
- *Flaws* :
  - I don't believe conflicts are as rare as they suggest.(The authors could have some experiments with a realistic workload to establish their assumptions about this – which they didn't do). There is distinct non-randomness in realistic access patterns; these correlated accesses have much higher probability of conflicting.
  - It is possible to have starvation as opposed to deadlock with degenerate access patterns and optimistic access control. This is never mentioned.
  - Very expensive as amount of stuff changed in a transaction increases. Unclear how to handle when set of things changed exceeds physical memory.
  - Doesn't notice two big advantages of method: Avoidance of livelock (someone has to make progress to keep someone else from failing); Cleanly deals with query processes failing w/o having a long delay as failure detection runs.
- *Take home message* : In the scenarios where the worst case situations might arise only very rarely, it makes more sense to use a more relaxed scheme to achieve higher throughput, making sure that the worst case situations are appropriately detected and handled. *Make the common case simpler.*