Problem 3: Performance of Optimistic and Pessimistic Offline Locking

Suppose a database contains N distinct objects.

Each user transaction against this database works as follows:

> First it chooses an object at random and reads the object's value, v_{old}. It performs some calculation based on v_{old}. Then with probability p_w it updates the object, changing its value to v_{new}. If the above actions succeed the transaction completes successfully. Otherwise (either the chosen object could not be read or we decided to update it and failed) the transaction is retried on a (different) randomly chosen object. The transaction continues retrying until it succeeds.

Admittedly this model of "infinite retries" is somewhat unrealistic, but it is relatively easy to analyze.

Suppose the database server is infinitely fast, but the communication system and the client are not: sending a message between the client and server (in either direction) takes expected time T_x, and the processing performed by the client (to decide whether to update the object and compute its new value) takes expected time T_p.

We will consider implementations using *optimistic* and *pessimistic* offline locking, as discussed in Lecture 20. Assume that all failures are the result of contention for objects (i.e., other failures are sufficiently unlikely that we can ignore them for this analysis). Thus

> In the optimistic implementation, a failure occurs because a client reads the value of some object and later attempts to update it, but some other client has updated the object in the meantime.
>
> In the pessimistic implementation clients hold locks on objects while processing them, so a failure occurs when a client attempts to read (and lock) an object on which another client currently holds the lock.

Even on an infinitely fast database system, these two schemes have different performance.

(a) Suppose the site is in steady state executing n transactions per second (TPS). Assuming n is small compared to N, approximately how many objects are updated per second? Why do we need the assumption that $n << N$?

(b) Again assuming $n << N$, what is the approximate probability $p_u(\Delta t)$ that an individual object chosen at random will be updated during a short time interval of length Δt? The answer is a function of n (the TPS value). Why does it not depend on which implementation (optimistic or pessimistic) is being used?

(c) We now consider L_o, the expected latency for a client transaction to complete using optimistic offline locking. Let p_{fo} be the probability that a failure will be detected during an update step of the transaction (this is the only point at which a failure can be detected using optimistic locking). You will derive an expression for p_{fo} in the next part. The latency satisfies the following equation:

$$\begin{aligned} L_o &= (2T_x + T_p) + (1 - p_w)(0) \\ &\quad + p_w(2T_x + p_{fo}L_o) \end{aligned}$$

Explain why this equation holds. Note that $2T_x$ is an RPC round-trip time, and note that L_o appears on both sides of the equation.

(d) Derive a formula for p_{fo}. *Hint:* Consider how a failure is detected, and use the answer to part (b).

(e) Now consider L_p, the expected client transaction latency using pessimistic locking. Let p_{fp} be the probability that a failure is be detected during a read-and-lock step of the transaction (this is the only point at which a failure can be detected using pessimistic locking). By analogy with part (c), derive an equation for L_p in terms of p_{fp}.

(f) Now derive a formula for p_{fp}. *Hint:* You know N (the total number of objects), n (the number of TPS that complete), and total length of time each completed transaction holds a lock.

(g) Under what conditions (on N, n and p_w) is does optimistic offline locking preferable to pessimistic?