Homework 4

CS425/ECE428 Spring 2026

Due: Friday, April 17 at 11:59 p.m.

1. **Blockchains**

   In a system using a blockchain for distributed consensus, in order to add a block to a chain, a participating node must solve the following puzzle: it must find a value x such that its hash, $H(x||seed)$, is less than T. The hash function is such that a given value of x can uniformly map to any integer in $[0, 2^{256} - 1]$. Assume T is set to 2^{230}.

   (a) (1 point) What is the probability that a given value of x, randomly chosen by the participating node, is a winning solution to the puzzle (i.e. $H(x||seed) < T$)?

   (b) (2 points) Assume a participating node adopts the standard strategy for solving the puzzle: it randomly picks a value x and checks if it is the winning solution. It keeps repeating this step, until a winning solution is found. Further assume that, for simplicity, the strategy is memoryless (unoptimized), in the sense that a value of x that has already been checked can get re-checked if it is randomly selected again. If the node can compute and check hashes at the rate of 2^{12} values per second, what is the probability that it finds a winning solution within 5 hours? (You may round your answer to five decimal places.)

   (c) (6 points) Assume that there are 2000 participating nodes in the system. Of these, 1500 nodes can compute and check hashes at the rate of 2^{12} values per second (lets call these set of nodes Group A). The remaining 500 nodes can compute and check hashes at the rate of 2^7 values per second (lets call these set of nodes Group B). Each node starts solving the puzzle at exactly the same time.

   (i) What is the probability that at least one node in the system finds a winning solution *within 1 minute*?

   (ii) What is the probability that a winning solution is found within 1 minute by at least one node in Group A and by no nodes in Group B?

   (iii) What is the probability that a winning solution is found within 1 minute by at least one node in Group B and by no nodes in Group A?

   (You may round your answer to five decimal places.)

2. **Concurrency Control: Two-phase locking, Deadlocks, and Timestamped Ordering**

   Consider the following two transactions, each with five operations:

	$T1$	$T2$
1	read D	write C
2	read B	read B
3	write A	read A
4	write E	write D
5	read F	read E

   Clarification comment: Various sub-parts of this question require you to consider or construct different *interleavings* of the above operations, where the operations are executed in the order represented by the interleaving. Assume that an operation has been executed when the read/write call returns (even if it is after a tentative write). A locking or waiting based mechanism can execute an operation only after any required lock has been acquired or waiting conditions have been met.

   (a) (2 points) Write down all the conflicting pairs of operations across the two transactions. (You can refer to each operation as $Tn.m$; e.g., $T1.1$ is "read D", $T1.2$ is "read B", and so on).

   (b) (2 points) Is the following interleaving of operations across $T1$ and $T2$ serially equivalent? Explain why or why not.

$T1$	$T2$
	write C
read D	
read B	
	read B
	read A
	write D
write A	
write E	
read F	
	read E

(c) (4 points) Is there a *non-serial* interleaving of operations across $T1$ and $T2$, that could result from using strict two-phase locking (with read-write locks), and is equivalent in effect to a serial execution of $T1$ followed by $T2$? If yes, provide an example. If not, explain why.

(d) (4 points) Is there a *non-serial* interleaving of operations across $T1$ and $T2$, that could result from using strict two-phase locking (with read-write locks), and is equivalent in effect to a serial execution of $T2$ followed by $T1$? If yes, provide an example. If not, explain why.

(e) (4 points) Is there a *non-serial* interleaving of operations across $T1$ and $T2$, that is equivalent in effect to a serial execution of $T1$ followed by $T2$, and is impossible to achieve using strict two-phase locking (with read-write locks)? If yes, provide an example and explain what makes it impossible to achieve with strict two-phase locking (with read-write locks). If not, explain why.

(f) (4 points) Write down a *partial* interleaving of the operations across $T1$ and $T2$ that is compliant with strict two-phase locking (with read-write locks) and leads to a deadlock. List which lock (and in which mode – read or write) will be waited upon by each transaction in your deadlock. (The interleaving you provide will be partial in the sense that once you encounter a deadlock, neither transaction can continue executing its operations.)

When presenting your example for the sub-parts below, clearly indicate which timestamp value (picked between 1 or 2) gets assigned to each transaction in your example. In other words, when constructing your example, you are free to assign the lower timestamp 1 either to $T1$ or to $T2$ (and consequently a higher timestamp of 2 to the other transaction) – you must clearly indicate which timestamp you assign to which transaction. Also mention the relevant state maintained by the objects for timestamped ordering. You can use the example format like "write X (X.committedTS = 1, X.RTS = [1,2], X.TW=[2])" to indicate the state maintained by the object (i.e. timestamps for reads and tentative writes) after an operation has been executed or a transaction has been committed. Assume the committed timestamp for each object is 0 before either of these two transactions are initiated. While different variants of timestamp ordering exist, for consistency, please follow the timestamp ordering rules discussed in class.

(g) (4 points) Write down an interleaving of the operations across $T1$ and $T2$ that is equivalent in effect to a serial execution of $T1$ followed by $T2$, that could result from using *timestamped ordering*.

(h) (4 points) Write down a *partial* interleaving of the operations across $T1$ and $T2$, that could result from using *timestamped ordering* and would cause one of the transactions to be aborted.

(i) (3 points) Is there a *non-serial* interleaving of operations across $T1$ and $T2$, that is equivalent in effect to a serial execution of $T1$ followed by $T2$, and is impossible to achieve using the timestamp ordering rules discussed in class? If yes, provide an example. If not, explain why.