Homework 5

CS425/ECE428 Spring 2019

Due: Wednesday, May 1 at 11:55 p.m.

1. Optimistic Concurrency ..... *18 points*

Consider the interleaving of 3 transactions. t_i here is used to represent temporary variables internal to the transaction, while X and Y are objects that are read and updated by the transactions.

	T1	T2	T3
1	$t_1 = X$		
2		$t_2 = Y$	
3		$X = t_2 + 1$	
4			$t_3 = X$
5			$X = t_3 + 2$
6	$t_4 = Y$		
7	$Y = t_1 + t_4 + 3$		
8		$Y = t_2 + 4$	
9			$t_5 = Y$
10			$Y = t_3 + t_5 + 5$

(a) (2 points) Is this interleaving serially equivalent? Why or why not?

(b) (2 points) Would this interleaving be possible with 2-phase locking, using either exclusive or reader/writer locks? Why or why not?

(c) (3 points) Suppose that timestamped optimistic concurrency was used. Write down for each step in the interleaving the updated read and write timestamp of the object being read or written. Assume that T1, T2, and T3 have timestamps 1, 2, and 3, respectively, and that the initial timestamps of each object are 0.

For the following parts, consider a modification of the transaction interleaving above:

	T1	T2	T3
1	$t_1 = X$		
2		$t_2 = Y$	
3		$X = t_2 + 1$	
4			$t_3 = X$
5			$X = t_3 + 2$
6	$t_4 = Y$		
7	$\boxed{X = t_1 + t_4 + 3}$		
8		$Y = t_2 + 4$	
9			$t_5 = Y$
10			$Y = t_3 + t_5 + 5$

(d) (2 points) Is this interleaving serially equivalent? Why or why not?

(e) (2 points) Explain what would happen in this interleaving using timestamped optimistic concurrency. Be detailed.

(f) (4 points) What would happen in this example if multi-version optimistic concurrency was used? Write down for each operation which version of X and Y would be read or written, assuming transactions T1. T2, T3 had timestamps 1, 2, and 3, respectively.

(g) (3 points) Repeat the previous part assuming that T1, T2, T3 had timestamps 3, 2, 1, respectively.

2. Bloom Filter ..... *5 points*

Consider a Bloom filter with 16 slots that uses 2 hash functions. We will use a version of SHA256 based on your netid. To hash `x`, you will need to run:

```
$ echo -n <netid> <x> | openssl sha256
```

E.g.:

```
$ echo -n nikita 1 | openssl sha256
369ca22d9a6484a2109492a23ac355721b240132288436273fa16ce245ab04b0
```

If you prefer, you can use Python3 to calculate it:

```
$ python3
Python 3.7.0 (default, Aug 22 2018, 15:22:33)
[Clang 9.1.0 (clang-902.0.39.2)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import hashlib
>>> hashlib.sha256(b"nikita 1").hexdigest()
'369ca22d9a6484a2109492a23ac355721b240132288436273fa16ce245ab04b0'
```

We will use the first hex digit of the output (3 in this example) as the first hash function, and the second digit (6) as the second hash function. Be sure to use your own netid in place of `nikita` here.

(a) (3 points) Add the strings '1', '3', '5', ..., '19' to this Bloom filter and write down which bits have been set. It's not too hard to do manually but you may want to write a quick script.

(b) (2 points) Test the resulting Bloom filter for the values '2', '4', '6', ..., '20'. Do you get any false positives, and if so, what are they?

3. Cassandra Quorums ..... *5 points*

Consider a Cassandra key that is replicated at 5 nodes in the local data center, L_1, L_2, L_3, L_4, L_5, three nodes in remote data center 1, R_1, R_2, R_3, and 3 nodes remote data center 2, S_1, S_2, S_3.

(a) (3 points) What is the minimum number of nodes that need to respond for a request configured with LOCAL_QUORUM, EACH_QUORUM, and QUORUM? (There are 3 subparts here)

(b) (2 points) Consider all possible combinations of read and write policies selected out of the quorum options above. (E.g., EACH_QUORUM for read, LOCAL_QUORUM for write.) Which combinations would *not* result in consistent reads? Write down an example read and write set for each combination that would result in inconsistency.

4. MapReduce ..... *11 points*

Implement the following computations using a chain of one or more MapReduce operations.

(a) (2 points) Given a directed graph $G = (V, E)$, compute the indegree of each vertex. Input pairs (k, v) where k is a graph vertex and v is a list of its out-neighbors; i.e., for each $x \in v$, (k, x) is a directed edge in E. Output pairs (k, v) where k is a graph vertex and v is the number of nodes that have k as its neighbors; i.e., $v = |\{x|(x, k) \in E\}|$

(b) (3 points) Compute a list of all nodes reachable from a vertex in exactly two hops. Input is as above. Output is (k, v) where k is a graph vertex and v is a list of nodes reachable in two hops; i.e., $v = \{x|\exists y$ such that $(k, y) \in E$ and $(y, v) \in E\}$.

(c) (3 points) Compute the size of a 4-neighborhood of each vertex; i.e., all the nodes reachable from from a vertex in at most 4 hops. Input as above; output is (k, v) where k is a graph vertex and v is the number of nodes reachable in at most 4 hops.

(d) (3 points) Compute the product of two matrices, $M^{(1)}$ an $M^{(2)}$. Input is (k, v) where k is a triple (i, j, n) such that $M^{(n)}_{i,j} = v$. Output is (k, v) where k is a pair (i, j) such that $M'_{i,j} = v$, where $M' = M^{(1)} \times M^{(2)}$.