Homework 4

CS425/ECE428 Fall 2021

Due: Friday, Dec 3 at 11:59 p.m.

This assignment has 4 pages and 5 questions, worth a total of 80 points. Solutions must be submitted via Gradescope. Solutions must be typed, not hand-written, but you may include hand-drawn diagrams.

You must acknowledge any sources you used to arrive at your solutions, other than the course materials and textbook. If you work in a group on homework assignments, please list the names of your collaborators, but make sure to write your own solution.

1. Consider the following interleaving of three transactions. Here, reads and writes to variables are explicitly labeled. The lowercase variables are local to each transaction.

	T1	T2	T3
1:	x = read(A)		
2:		z = read(D)	
3:			w = read(C)
4:			write(B, w+2)
5:	y = read(B)		
6:	write(C, x+y)		
7:		write(A, z+1)	
8:		write(E, z-1)	
9:			write(D, w-2)

   (a) (3 points) Identify all the conflicts in this interleaving. You can just specify the pairs of operation numbers.

   (b) (2 points) Consider an interleaving of *just* T1 and T2 that follows the same ordering as the full interleaving above. I.e.:

T1	T2
x = read(A)	
	z = read(D)
y = read(B)	
write(C, x+y)	
	write(A, z+1)
	write(E, z+1)

   Is this interleaving serially equivalent? Why or why not?

   (c) (3 points) Repeat the part above but with the corresponding interleaving of just T1 and T3, and then with the interleaving of just T2 and T3. Are these interleavings serially equivalent? Why or why not?

   (d) (2 points) Suppose that the variables are initialized to $A = 1, B = 2, C = 3, D = 4, E = 5$. What is the final value of the variables after running T1, T2, T3 using the above full 3-transaction interleaving?

   (e) (3 points) Now consider all six possible serial interleavings of T1, T2, and T3. That is, T1 then T2 then T3; T1 then T3 then T2; ... For each such serial interleaving, compute the final value of the variables A, B, C, D, E with the same initial values as in the previous part.

   (f) (2 points) Explain how you can tell that the given interleaving is not serially equivalent without doing the computations in the previous part.

   (g) (2 points) How would this interleaving be prevented using strict two-phase locking with reader/writer locks?

   (h) (2 points) How would this interleaving be prevented using timestamped concurrency? Assume that T1, T2, T3 have timestamps 1, 2, and 3, respectively.

(i) (4 points) Write down a serially equivalent execution of T1, T2, T3 where all three transactions overlap. (I.e., there is a point where each of T1, T2, T3 has executed at least one operation but none of the transactions have yet completed.) Explain why it is serially equivalent.

2. Consider a Bitcoin network with $N = 100$ nodes. Let us model the propagation of a newly mined block. Note that this will be a simplified model that disregards several complexities of the protocol.

   At time t, there are N_t nodes that have a copy of the block. $N_0 = 1$, which is the block that mines it. Each of the nodes picks a random other node to send the block to.[1] This node already has the block with probability $(N_t - 1)/(N-1)$ and it does not have the block with probability $(N - N_t)/(N-1)$. Therefore, the expected number of nodes that receive the block are $N_t(N - N_t)/(N-1)$. We will therefore use the recurrence:

$$N_{t+1} = \lfloor N_t + N_t(N - N_t)/(N-1) \rfloor$$

   (a) (3 points) Starting with $N_0 = 1$, how many rounds until all nodes receive the block?

   (b) (5 points) Calculate the chance that a chain split occurs. In each round, each node that has not yet received the block will mine a conflicting block with probability $1/(600 \times N)$. You may use a simulation to calculate this if you wish, but make sure to include your simulation code and use enough trials to get an accurate estimate of the true value.

   (c) (2 points) (Unrelated to above) Find some number such that `echo` *`netid n`* `| sha256sum` results in a number with at least 5 leading zeros, with your own netid. E.g.:

```
$ echo nikita 90242 | sha256sum
00000b8556ab757a1a7a6a3ab4b43ff0045975e439593b98a8281f244ab4a772  -
```

3. Consider a bank performing transaction processing. There are three types of operations: (i) `DEPOSIT` *account amount*, which adds the *amount* to the balance of *account* (ii) `WITHDRAW` *account amount*, which subtracts *amount* from the balance of the *account*. The transaction also has a consistency check where a transaction is aborted if *at the end of the transaction* any account balance is negative.

   Consider the following transactions:

```
T1: DEPOSIT A 30; DEPOSIT B 30; DEPOSIT C 50
T2: WITHDRAW A 10; DEPOSIT B 10; WITHDRAW C 60
T3: WITHDRAW A 30; WITHDRAW C 10
T4: WITHDRAW B 40; WITHDRAW C 10
```

   (a) (4 points) If the transactions are executed serially in the order T1, T2, T3, then T4, which of them will be committed and which will be aborted?

   (b) (3 points) What are the final balances of accounts A, B, and C?

[1] In Bitcoin nodes only send blocks to random *neighbors* but we will simplify the analysis here.

4. Below is a list of 128 nodes and their 32-bit ID specified in hex and decimal

Hexadecimal:

0271f7e5	051c3474	06fd9230	077e9532	0874e37d	09da6a01	0b14eab9	0d3671ea
0d8ea9bb	1452f156	154e3073	16632b1c	1b004f11	1cdf7c10	1d0fcd17	252c7750
2949d3af	29904e42	2d8f8345	2f0bb425	2fdf22c5	36a153cf	378aa66f	38c7f02c
3b403605	3f6462cd	414dd380	417e4d3a	4241cc88	42cd3bda	449cf8b1	4673c47e
46c2b3ce	4aa139c4	4b2dd547	4d4a537f	4e78adba	4e94ec32	4ffbf451	5252d6bd
53b97550	54cf8e1a	5c3477de	5d0a981c	5ddd7d7e	5e7f885d	601e02e3	62c2a9e0
66cdadff	673a8811	6890c79a	690ca3be	6927e4cf	69d971bc	6b4f81af	6bd7973a
6f0f09f9	70be1b30	73c68a3e	746e0df2	75590e12	78b8991f	78dc974c	791b40d1
81076d4a	81852191	82b4544c	87828ed1	8826367a	8a0d6bc7	8a7dc915	8b8a10d3
8ef06a66	8fa97eaa	9058438b	96fe0b06	9a437b40	9b83586a	9e887088	9ea42a42
9eea232f	9eef19e9	9fb56c3f	a18e07c6	a20af51f	a3a7ddc0	a6160a54	a81593c0
ace69076	b098241c	b2ae3134	b4135d63	b7ba8f93	b94919c8	ba052703	ba1342d0
bc6600c2	bd0a2417	bf015ef4	c8289ed2	cc484247	ccc15960	cdf675e0	d0a3916a
d1d6a00e	d38e2a77	d53d73bc	d690274a	d8084341	db1778be	dc579a99	dcf3ae76
df41085f	df5db463	e24cf6c3	e2c969b2	e58813f8	e8d724e2	ec838ca1	ee0d25c7
efdc2e2c	f1ca7e2a	f2d51880	f3f4eca9	f5fefe85	f63c195a	fd39eaa5	feecd1d5

Decimal:

41023461	85734516	117281328	125736242	141878141	165308929	185920185	221671914
227453371	340980054	357445747	375597852	453005073	484408336	487574807	623671120
692704175	697323074	764379973	789296165	803152581	916542415	931833455	952627244
994063877	1063543501	1095619456	1098796346	1111608456	1120746458	1151137969	1181992062
1187165134	1252080068	1261294919	1296716671	1316531642	1318382642	1341912145	1381160637
1404663120	1422888474	1546942430	1560975388	1574796670	1585416285	1612579555	1656924640
1724755455	1731889169	1754318746	1762436030	1764222159	1775858108	1800372655	1809291066
1863256569	1891506992	1942391358	1953369586	1968770578	2025363743	2027722572	2031829201
2164747594	2172985745	2192856140	2273480401	2284205690	2316135367	2323499285	2341081299
2398120550	2410249898	2421703563	2533231366	2588113728	2609076330	2659741832	2661558850
2666144559	2666469865	2679467071	2710439878	2718627103	2745687488	2786462292	2819986368
2900791414	2962760732	2997760308	3021167971	3082456979	3108575688	3120899843	3121824464
3160801474	3171558423	3204538100	3358105298	3427287623	3435223392	3455481312	3500380522
3520503822	3549309559	3577574332	3599771466	3624420161	3675748542	3696728729	3706957430
3745581151	3747460195	3796694723	3804850610	3850900472	3906413794	3968044193	3993839047
4024184364	4056579626	4074051712	4092914857	4127129221	4131133786	4248431269	4276933077

You may also download these IDs in JSON format at:
`https://courses.grainger.illinois.edu/cs425/fa2021/assets/hw/chord-ids.json`.

(a) (9 points) List the fingers of the following nodes:

- 484408336 (0x1cdf7c10)
- 1095619456 (0x414dd380)
- 3500380522 (0xd0a3916a)

(b) (1 point) How many *distinct* fingers would each node have if all nodes were equally spaced?

(c) (2 points) Which of the above 3 nodes will store the most keys, on expectation? Which will store the fewest?

(d) (5 points) List the set of nodes that will be contacted if node 3500380522 (0xd0a3916a) searches for key 0x12345678?

(e) (5 points) Suppose a power outage took out all nodes with ids that are a perfect multiple of 3, and no stabilization has been run. What nodes would be contacted by the same search? When the Chord routing algorithm encounters a node that has failed, it tries using the next smallest finger entry, and so on, until it finds one that is alive. If this doesn't work, it will use its successor, and then the successor's successor, etc.

5. Consider two transactions shown below:

   T1: read A; write B; write A; read C; write E
   T2: read C; write D; read A; read E; write B

   (a) (4 points) Write a non-serial interleaving of T1 and T2 that would be feasible using strict two-phase locking with reader/writer locks.

   (b) (4 points) Write down a partial interleaving of T1 and T2 that would lead to a deadlock if strict two-phase locking with reader/writer locks are used. State what lock and in which mode is being requested by each transaction.

   (c) (2 points) Write down an interleaving of T1 and T2 that is serially equivalent but impossible with strict two-phase locking (assuming reader/writer locks), Explain why it is impossible with strict two-phase locking.

   (d) (4 points) Write down a (potentially partial) interleaving of T1 and T2 that would cause T1 to be aborted if timestamped ordering were used. Assume that T1 and T2 have transaction timestamps 1 and 2, respectively, and show how the timestamps are updated. Explain why the abort happens.

   (e) (4 points) Write down a non-serial interleaving of T1 and T2 that could happen if timestamped ordering were used where both T1 and T2 successfully commit. T1 and T2 should have the transaction timestamps of 1 and 2, respectively. Show how timestamps are updated in your execution.