HW2 Solutions: CS425 FA18

1. (Solution and Grading by: <Le>)

Seed 2018 FilterResult 3 6 10 16
Seed 2019 FilterResult 4 9 18 7
Seed 2020 FilterResult 5 12 28 16
Seed 2021 FilterResult 6 15 8 17
Seed 2022 FilterResult 7 18 22 16
Seed 2023 FilterResult 8 21 6 19
Seed 2024 FilterResult 9 24 24 0
Seed 2025 FilterResult 10 27 12 29
Seed 2026 FilterResult 11 30 2 16
Seed 2027 FilterResult 12 1 26 31
Seed 2028 FilterResult 13 4 20 16
Seed 2029 FilterResult 14 7 16 9
Seed 2030 FilterResult 15 10 14 16
Filter result 15 10 14 16 exists, current seed 2030 (TERMINATING)

2. (Solution and Grading by: <Faria>)

Regular bloom filter:
Number of hash functions = k = 4
Size of input set = n = 8,10
Size of filter in bits = m = 1024
False positive rate = (1 - e^(-4*n/1024))^4

Orlando's bloom filter:
K = 1 per bloom filter
L = 2^r
m = 1024
False positive rate = (1 - e^(-(n)/1024))^L

Orlando's bloom filter has a lower false positive rate if:
(1 - e^(-4*n/1024))^4 > (1 - e^(-(n)/1024))^L

If X = e^(n/1024), then we have
(1 - X^-4)^4 > (1 - X^- 1)^L
By taking ln on both sides,
4 ln (1-X^-4) > L ln(1-X^-1)

Thus, Orlando's bloom filter has a lower false positive rate if L > 4 ln (1-X^-4) / ln(1-X^-1).

N = 80	N = 10
X = e^(n/1024) = 1.08125780745 Regular Bloom Filter FP: 0.00518834548 Orlando's BF FP: 0.07515118678^L Orlando's BF is better if L > 2.0327764206	X = e^(10/1024) = 1.00981346432 Regular Bloom Filter FP: 0.00000215387 Orlando's BF FP: 0.00971809612^L Orlando's BF is better if L > 2.81590516117

Thus, in both cases, L >= 3.

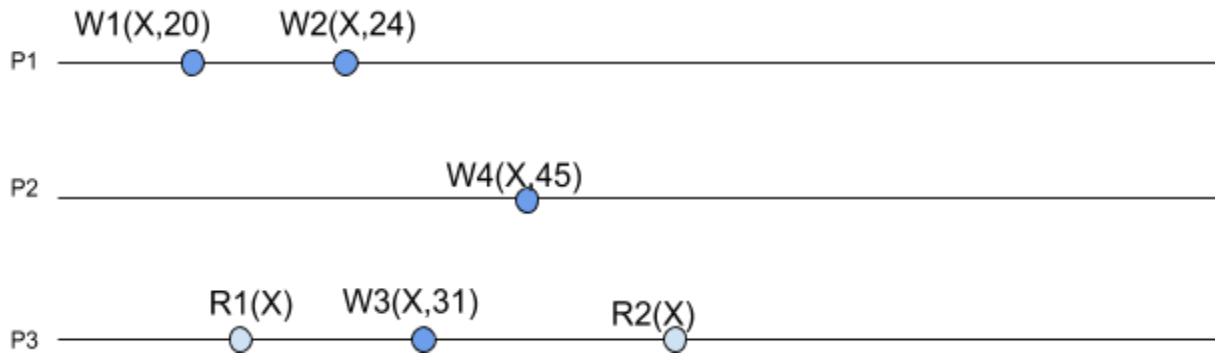
General Comments:
Some students have said that as L = 2^r, and as L > 2, this means that r >= 2 and L >= 4. The assumption is that r needs to be an integer. Students shouldn't make assumptions when the question doesn't say anything to the effect that r is an integer. No marks deducted, however.

3. (Solution and Grading by: <Rahul>)

Linearizability:
The order of execution of operations preserves the global ordering of operations consistent with real time occurence of events.

Sequential consistency:
All operations appear to be executed in some sequential order that is consistent with the order seen at the individual process.

Causal consistency:
This requires that effects are observed only after their respective causes.



Consider the following diagram where Wi implies write and Ri implies read. The values of X which would satisfy the respective consistency models are.

1. **Linearizability** : R2 (45) as the last write according to the global clock is W4.
2. **Sequential consistency** : R2 (24) would be valid under sequential consistency as W2 happens after W1 in process P1.
3. **Causal consistency** : R2(31) causal ordering would be preserved by reading W3.

4. (Solution and Grading by: <Rui>)
   The main problem is that RM might have a clock drift (with respect to the outside world) and will force the entire cluster to drift away for the outside time. This means there will be an issue with external synchronization.

   One solution is bring in a pre-synchronized GPS/atomic clock and use that as the master for synchronization (either via NTP or Cristian's as described in the question). They do not violate the "no internet" rule in the question.
   An alternative solution is to use Berkeley's algorithm. It involves averaging the time across the cluster which could probably cancelling out most of the clock drift.

5. (Solution and Grading by: <Rui>)
   Solution:
   From Christian's algorithm, error <= (RTT - min1 - min2) / 2 where
   min1 = client's APP to NIC time + fixed transmission time + server's NIC to APP time
   min2 = server's APP to NIC time + fixed transmission time + client's NIC to APP time

   If we denote:

the fixed transmission time is assumed as 0 here

The unknown client's NIC to APP time as y,

Then we could get error bound as:

error <= (3.99 - (0.01 + 0.33) - (333.3/1000 + y)) / 2

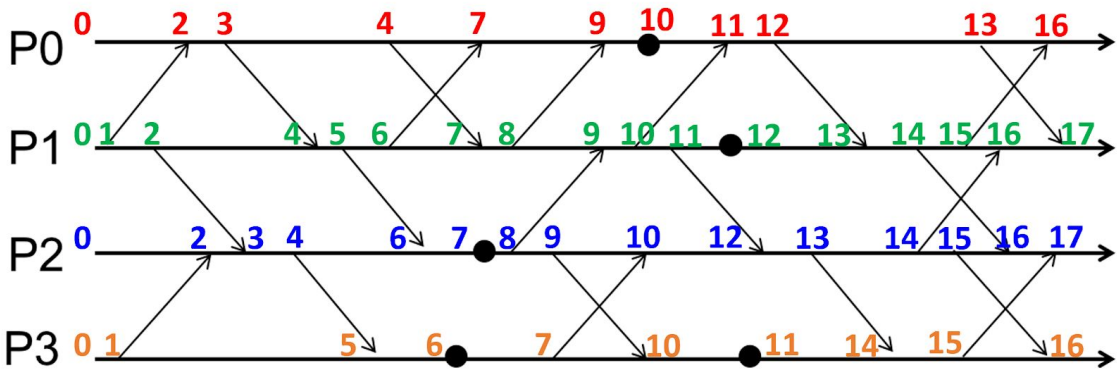
= (3.99 - 0.01 - 0.33 - 0.3333 - y) / 2

<= (3.99 - 0.01 - 0.33 - 0.3333) / 2

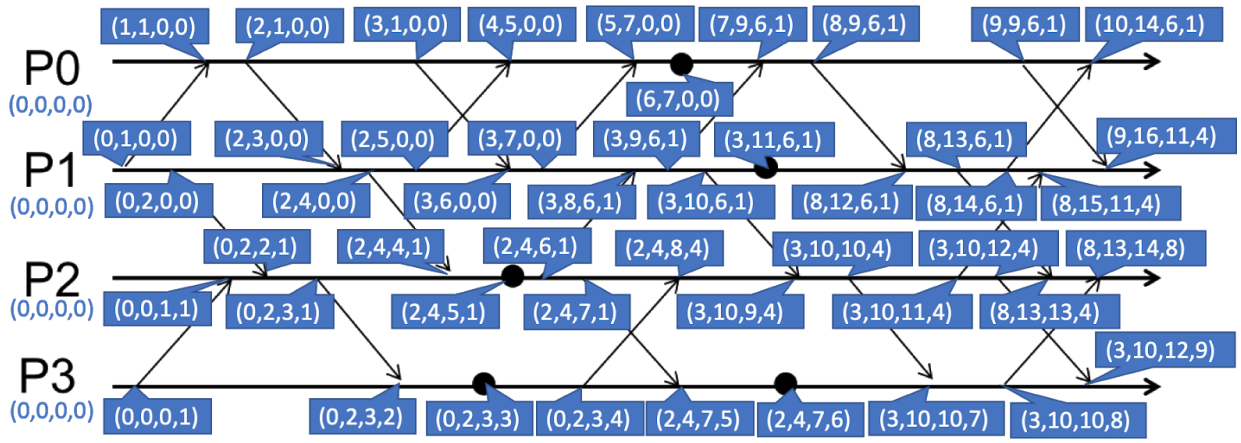
= 1.65835 ms

(Note: It's also good to assume non-zero transmission time (e.g. notated as x), but no difference in the result.)

6. (Solution and Grading by: <Shegufta>)



7. (Solution and Grading by: Beomyeol)



8. (Solution and Grading by: <Zhuolun Xiang>)

The algorithm is correct.

On each instruction, send and receive, the timestamp will be incremented by at least $F(e)+1$, which can guarantee the correctness of the algorithm. In fact any positive increment can guarantee this.

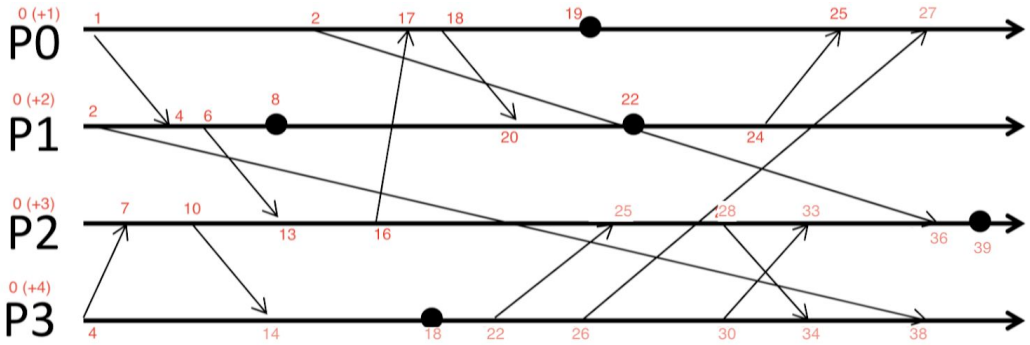
More formally: We need to prove that for any two events a, b such that $a \rightarrow b$, their timestamp satisfies $ts(a) < ts(b)$. We say a, b are neighbor events if 1) a, b are both events at some process and b happens right after a, or 2) a is a send event and b is the corresponding receive event. When a, b are neighbor events, $ts(a) < ts(b)$ since $ts(b)$ will be incremented by at least $F(e)+1$. When a, b are not neighbor events, there must be a sequence of events $a \rightarrow e_1 \rightarrow ... \rightarrow e_k \rightarrow b$ such that any neighboring pair are neighbor events. Then by simple induction on the event sequence we have $ts(a) < ts(b)$.

9. (Solution and Grading by: <Zhuolun Xiang>)

   For any two quorum of size Q, their intersection size is at least $2Q-N$. And the intersection size between this intersection and the third quorum set is at least $2Q-N+Q-N=3Q-2N$. Intersection between three quorums is no less than K implies $Q_{min}=\lceil (2N+k)/3 \rceil$.

   a. 34
   b. $\lceil (2N+2)/3 \rceil$
   c. $\lceil 5N/6 \rceil$
   d. $\lceil 3N/4 \rceil$

10. (Solution and Grading by: <Rahul>, Grading by <Federico>)



The proposed scheme for timestamps will maintain causality as the timestamp sequence is strictly increasing.

Suppose proving causality means E1 → E2 => timestamp(E2) > timestamp(E1)
Since there is always a message or instruction from E1 → E2 the timestamp will always be increasing.

When we can draw a path of events that follow some order, their timestamp values will always be increasing if the follow the below rules.

1. A process increments its counter when a send event or instruction happens.
2. The send message carries the event timestamp.
3. For recv event the clock is incremented by max(local_clock, msg_timestamp) + (i+1).