Homework 9

Erik Brakke

April 12, 2016

Answer 1

a. Assign $R1 = 9$, $R2 = 5$, $R3 = 6$ then processes can finish in the following order $P4, P5, P2, P3, P1$
This is the optimal allocation for the resources becuase it limits $R1, R2$ to the bare minimum that it will need, and it only adds 1 more resource to $R3$

b. Assiming $A, B, C = R1, R2, R3$ we can show that all of the processes can finish after this request
After this request, all processes can finish in the following way:
$P2 => (0, 2, 0)$ – Now P2 is finished
$P4 => (0, 1, 1)$ – Now P1 is finished
$P5 => (4, 3, 1)$ – Now P5 is finsihed
$P3 => (6, 0, 0)$ – Now P3 is finished
$P1 => (7, 4, 3)$ – Now P1 is fnished

c. No this should not be granted. $P2$ had to finished first for its request to be allowed, but if $P1$ now makes this request, then $P2$ can no longer finish, thus there will be a deadlock

Answer 2

a. All of these succeed because any read happens after a write had already been commited by a previous transaction

b. (a) V1 will succeed because it has read and written data before any other transaction has written data
   (b) V2 will fail because it has created a local copy before T1 had committed its changes, therefore it is reading stale data
   (c) V3 will succeed because is is reading after T1 has finished

c. (a) V1 will succeed because no changes were made since creating the local copy
   (b) V2 will fail because its local data copy is now out of date after T1 committed its changes
   (c) V3 would have failed if T2 went through, but because T2 failed, it has the most up to date local copy of the data, and it will happen after T1, so it will succeed

Answer 3

Assumption is I am finding a way to cause blocking with just 2PL, not 2PL with the proposed prevention for deadlocks

a. A deadlock can occur If transaction A gets a read lock for x, then transaction B gets a read lock for z and y. B is now waiting for a read lock on x, and A will either wait for a read lock on y or z. When this happens, a dead lock will occur because both transactions are waiting on resources that the other has locked.

b. Since the order in which transaction B does its instruction does not matter, We can just move the write(x) command to be first. Now if either A or B get the lock on x first, then that transaction will complete because the other is just waiting on x. Once the transaction completes it will release all of its locks, and the other can finish

c. The downsides to the proposed deadlock prevention approach is that if two transactions require the same resource, they will never be able to execute in parallel. Whichever transaction gets there first will immediately lock all the resources it needs, and any other transaction that comes along will have to wait until the first one has completed.

Answer 4

1. This is done for consistency. If a customer could write to the data that was currently being copied, then it would be the case that the new machine would have data that is not consistant with the current state of the system.

2. The resource that was consumed was the API. The processes that were starved were whatever was making the API requests to create new EBS instances. Because the re-mirroring used all of the capacity of the EBS control plane, these requests to create a new instance were starved as there was no available CPU for them. And because the requests were not timing out, they were all forming into a long queue.

3. This is crucial for consistency. If there was no single point of accesses, then certain EBS instances would have different states and it would be impossible to remirror as there is no clear node to choose

4.

5. Searching for a new node with free space to re-mirror to has no bound. This caused the all free space to drain quickly which caused nodes to become stuck searching for free space, this also caused all CPU resources to be consumed. A limit on the number of nodes attempting to re-mirror would have alieviated this because the free space would not have depleted so fast, and CPU resources would be free to handle other jobs (like incoming API requests)