Distributed Databases 2018 HW 3

Student ID: **2018280070** Name: **Peter Garamvoelgyi**

1. a) Which of the following schedules are conflict equivalent? (Ignore the commit "C" and abort "A" commands)?

```
S1 = W2(x), W1(x), R3(x), R1(x), C1   , W2(y), R3(y), R3(z), C3   , R2(z), C2
S2 = R3(z), R3(y), W2(y), R2(z), W1(x), R3(x), W2(x), R1(x), C1   , C2   , C3
S3 = R3(z), W2(x), W2(y), R1(x), R3(x), R2(z), R3(y), C3   , W1(x), C2   , C1
S4 = R2(z), W2(x), W2(y), C2   , W1(x), R1(x), A1   , R3(x), R3(z), R3(y), C3
```

Two operations are conflicting if they access the same piece of data and one of them is a write. Two schedules are conflict equivalent if the relative order of execution of conflicting operations is the same. To check this, let us do a pairwise comparison of S1-S4:

```
S1 = W2(x), W1(x), R3(x), R1(x), C1   , W2(y), R3(y), R3(z), C3   , R2(z), C2
S2 = R3(z), R3(y), W2(y), R2(z), W1(x), R3(x), W2(x), R1(x), C1   , C2   , C3
```

```
S1 = W2(x), W1(x), R3(x), R1(x), C1   , W2(y), R3(y), R3(z), C3   , R2(z), C2
S3 = R3(z), W2(x), W2(y), R1(x), R3(x), R2(z), R3(y), C3   , W1(x), C2   , C1
```

```
S1 = W2(x), W1(x), R3(x), R1(x), C1   , W2(y), R3(y), R3(z), C3   , R2(z), C2
S4 = R2(z), W2(x), W2(y), C2   , W1(x), R1(x), A1   , R3(x), R3(z), R3(y), C3
```

```
S2 = R3(z), R3(y), W2(y), R2(z), W1(x), R3(x), W2(x), R1(x), C1   , C2   , C3
S3 = R3(z), W2(x), W2(y), R1(x), R3(x), R2(z), R3(y), C3   , W1(x), C2   , C1
```

```
S2 = R3(z), R3(y), W2(y), R2(z), W1(x), R3(x), W2(x), R1(x), C1   , C2   , C3
S4 = R2(z), W2(x), W2(y), C2   , W1(x), R1(x), A1   , R3(x), R3(z), R3(y), C3
```

```
S3 = R3(z), W2(x), W2(y), R1(x), R3(x), R2(z), R3(y), C3   , W1(x), C2   , C1
S4 = R2(z), W2(x), W2(y), C2   , W1(x), R1(x), A1   , R3(x), R3(z), R3(y), C3
```

According to this analysis, **only S1 and S4 are conflict equivalent**. (Note that not all conflicts are highlighted above.)

1. b) Which of the following schedules (S1,S2,S3,S4) are serializable?

A schedule is serializable if its net effect is equivalent to some serial execution of the transactions. This means that Sx is serializable if there is a serial schedule S0 such that Sx and S0 are conflict equivalent.

```
S1 = W2(x), W1(x), R3(x), R1(x), C1   , W2(y), R3(y), R3(z), C3   , R2(z), C2
SA = W2(x), W2(y), R2(z), C2   , W1(x), R1(x), C1   , R3(x), R3(y), R3(z), C3
```

```
S2 = R3(z), R3(y), W2(y), R2(z), W1(x), R3(x), W2(x), R1(x), C1   , C2   , C3
Problem: W1(x), R3(x), W2(x), R1(x)
```

```
S3 = R3(z), W2(x), W2(y), R1(x), R3(x), R2(z), R3(y), C3   , W1(x), C2   , C1
SB = W2(x), W2(y), R2(z), C2   , R3(z), R3(x), R3(y), C3   , R1(x), W1(x), C1
```

```
S4 = R2(z), W2(x), W2(y), C2   , W1(x), R1(x), A1   , R3(x), R3(z), R3(y), C3
```

According to the analysis above, **S1, S3 and S4 are serializable**. (S1 and S3 are conflict equivalent to the serial SA and SB respectively; S4 is already serial.)

S2 is not serializable: We cannot rearrange R and W operations in a serial way without introducing violations of conflict equivalence.

2. Compare and analyze the relative merits of centralized and hierarchical deadlock detection approaches in a distributed DBMS.

Centralized deadlock detection: "*One site is designated as the deadlock detector for the system. Each scheduler periodically sends its local WFG to the central site which merges them to a global WFG to determine cycles.*"

Pros:

+ Simple to implement.
+ Single source of truth; easier to manage and monitor.

Cons:

– Single-Point-of-Failure (SPoF): If the site fails or becomes unavailable, replacement policies are needed. Also, a Byzantine-faulty site might make incorrect detection (false positives and false negatives).
– Hard-to-set parameters: How often to transmit? Cost-delay tradeoff.
– High communication costs; potential communication bottleneck that might affect the rest of the system.

Note: A reasonable choice if the concurrency control algorithm is also centralized.

Hierarchical deadlock detection: "*Organize sites into a hierarchy and deadlock detectors (DD) send local graphs to parent in the hierarchy.*"

Pros:

+ Localized detection: If a deadlock can be detected in an intermediate node, there is no need to propagate the partial WFG to the top.
+ Lower communication costs.
+ Better fault tolerance: With faulty nodes, local deadlock detection is still possible.

Cons:

– Fault tolerance is still not perfect: Failure of a single node can make detecting *global* deadlocks impossible. Policies are needed to restore the hierarchy between remaining nodes.

Note: Distributed deadlock detection has the best fault tolerance and robustness. However, it is much harder to implement and has its unique challenges.

3. Consider the following modification to a local wait-for graph: Add a new node T_{ex}, and for every transaction T_i that is waiting for a lock at a certain external site, add the edge $T_i \rightarrow T_{ex}$. Also add an edge $T_{ex} \rightarrow T_i$ if a transaction executing at a certain external site is waiting for T_i to release a lock at this site.

a) If there is a cycle in the modified local waits-for graph that does not involve T_{ex}, what can you conclude?

It is easy to see that cycles in the modified WFG that do not contain T_{ex} must have been present in the original WFG. This means that there is a cycle in the local WFG, which in turn means that there is a deadlock situation between local locks.

Thus, a cycle in the modified WFG that does not contain T_{ex} indicates a local deadlock.

b) If there exists a cycle involving T_{ex}, what can you conclude?

A cycle in the modified WFG containing T_{ex} suggest that there is a circular dependency between external and local locks. However, let us not forget that T_{ex} represents all external sites; it is then possible that local locks are waiting for external locks on site1, while external locks on site2 are waiting for the local locks, which is not a deadlock scenario (unless there is a dependency between site1 and site2 locks as well).

Thus, we can conclude that a cycle in the modified WFG containing T_{ex} is a necessary but not sufficient indicator of a cross-site / global deadlock.

4. Whenever the local waits-for graph at a certain site suggests that there might be a global deadlock, send the local waits-for graph to the next site. At that site, combine the received graph with the local waits-for graph. If this combined graph does not indicate a deadlock, ship it on to the next, next site, and so on, until either a deadlock is detected or we are back at the site that originated this round of deadlock detection. Is this scheme guaranteed to find a global deadlock if one exists?

Answer: **Yes.**

Given n sites, the worst-case scenario (in terms of communication cost) is that we almost go a full round and end up at site n – 1 before we can detect the deadlock.

In this case, site n – 1 will have received all the local WFGs from other sites (and of course it knows its own WFG) so it is able to reconstruct the global WFG.

Given the global WFG, this site is guaranteed to find a global deadlock if one exists.

Caveat: Of course, if any of the WFGs change during this process, newly introduced deadlocks might remain undetected. In this case, these deadlocks will be found during the next round.