Two Phase Locking Scheduler
Data Management

Simone Scaccia

October 2023

Abstract

This document discusses implementing a 2PL scheduler for the Data Management project. The scheduler aims to check if a schedule is in the 2PL class with exclusive locks or with shared and exclusive locks, and it returns the lock-extended schedule, the data action projection, and the serial conflict-equivalent schedule. The scheduler also implements the lock anticipation to try to avoid blocking transactions. The scheduler accepts in input only read, write, and commit operations. Deadlocks are only recognized by the scheduler, and the scheduler does not implement any deadlock resolution policy. The scheduler is implemented in Java, using Spring Boot, and Docker. The implementation of the scheduler is available on GitHub [1].

1 Introduction

Two-phase locking (2PL) is a method for concurrency control that guarantees serializability by using a lock mechanism to ask and perform an operation on a specific element within the database.

1.1 Background information

Concurrency control in DBMS ensures that multiple transactions execute without compromising data integrity. A transaction schedule is considered serializable if the resulting database state is the same as if its transactions are executed one at a time without interleaving. The scheduler is the part of the transaction manager that is responsible for managing the schedule. A transaction is the execution of a software procedure that can read from and write on a database, forming a single logical unit. The Two-phase locking method introduces new operations to request and release elements in the database. A schedule that includes these operations is referred to as a lock-extended schedule. Every transaction in a lock-extended schedule must be well-formed and the schedule must be legal. The definition of these two rules depends on the version of the Two-phase locking protocol.

There are two different versions of the Two-phase locking protocol, distinguished by their operations:

- Exclusive locks: The exclusive lock $l_i(A)$ is used to request, by the transaction T_i, read or write access to an element A in the database. The unlock $u_i(A)$ is used to release, by the transaction T_i, the element A in the database.
  A transaction is considered well-formed if each $l_i(A)$ or $u_i(A)$ is issued only once, and every read or write action is enclosed by a pair of lock-unlock.
  A lock-extended schedule is legal if no transaction locks an element already locked by another transaction until it has been unlocked.
  A scheduler with exclusive locks follows the Two-phase locking protocol if all lock operations of a transaction precede its unlock operations for every output generated by the scheduler.

- Shared and exclusive locks: By recognizing that two read operations on the same element do not conflict, we can implement a new type of lock called a shared lock.
  The exclusive lock $xl_i(A)$ is used to request write access. The shared lock $sl_i(A)$ is used to request read access. The unlock $u_i(A)$ is used to release the access.
  The transition from the shared lock $sl_i(A)$ to the exclusive lock $xl_i(A)$ is called a lock upgrade.
  A transaction T_i is considered well-formed if:

  - Every $r_i(A)$ must be preceded by either $sl_i(A)$ or $xl_i(A)$, with no $u_i(A)$ in between.
  - Every $w_i(A)$ must be preceded by $xl_i(A)$ with no $u_i(A)$ in between.
  - Every lock $sl_i(A)$ or $xl_i(A)$ is followed by $u_i(A)$.

  A lock-extend schedule is legal if:

  - An $xl_i(A)$ must be followed by an $u_i(A)$ before any $xl_j(A)$ or $sl_j(A)$ with $i \neq j$.
  - An $sl_i(A)$ must be followed by an $u_i(A)$ before any $xl_j(A)$ with $i \neq j$.

  A scheduler with shared and exclusive locks follows the Two-phase locking protocol if all lock operations, shared or exclusive, of a transaction precede its unlock operations for every output generated by the scheduler.
  When a transaction issues an unlock command on A, multiple transactions may be waiting for a lock on A. The scheduler has to decide which transaction should be granted the lock. There are various methods to do this:

  - The FIFO (First In, First Out) method helps prevent starvation.
  - Prioritize transactions that require a shared lock.
  - Prioritize transactions that require an upgrade lock.

The Two-phasing lock protocol's name comes from the locking and unlocking scheme in each transaction since the first phase is named growing transaction in which every required lock is granted, and then there is the second phase named the shrinking phase, in which every object previously granted is unlocked.
To keep track of granted locks, the scheduler utilizes a data structure known as a lock table.

A passive locking scheduler takes an input lock-extended schedule S and produces a corresponding output schedule by using the following rules:

- Processing a read or write operation on x by T_i: if x is locked by T_i then the operation proceeds else T_i is blocked.
- Processing $l_i(x)$: if x is locked by T_j, different from T_i, then T_i is blocked else the lock is executed.
- Processing $u_i(x)$: the scheduler updates the lock table.

An active locking scheduler can insert lock and unlock commands into the input schedule. To prevent transactions from being blocked, it can utilize its capability to insert lock and unlock commands. This is done by taking advantage of the information it has on the transactions. As a result, the scheduler may decide to anticipate the execution of the lock on an object.

DT(S) in a lock-extended schedule is the projection onto read, write, commit, and abort actions.
Gen(2PL) is the set of all schedules generated by all the 2PL schedulers. Each schedule follows the properties:

- is legal
- all its transactions are well-formed
- follows the 2PL protocol

Let S a schedule and S' the lock-extended output schedule of a 2PL scheduler with exclusive locks (with shared and exclusive locks). The class of 2PL schedules with exclusive locks (with shared and exclusive locks) is the set $\{DT(S')\}$. If a schedule is in the class of 2PL schedules with exclusive locks (with shared and exclusive locks), we say that it is accepted by the 2PL scheduler with exclusive locks (with shared and exclusive locks). It follows that if S is without lock operations and is accepted by a 2PL scheduler, then $S = DT(S')$ for some S' produced in output by a 2PL scheduler.
Deciding whether a schedule S without locking operation is in the class of 2PL schedule with exclusive (with shared and exclusive locks) requires checking whether there exists a schedule with exclusive locks (with shared and exclusive locks) S' such that $S' \in Gen(2PL)$ and $S = DT(S')$.

A deadlock happens when two transactions, namely T_1 and T_2, are using two

elements, A and B, and both request exclusive locks on the other's element. This leads to a situation where neither transaction can proceed.
A deadlock can occur when using a 2PL scheduler with exclusive locks or with shared and exclusive locks. Here's an example of deadlock:

$$r_1(A)r_2(B)w_1(B)w2(A)$$

A wait-for graph is maintained by the locking scheduler, with nodes representing transactions and edges showing which transactions are waiting for others to release locks. When a cycle appears in the graph, the deadlock can be resolved by terminating one of the transactions involved.

Theorem 1 *If $S' \in Gen(2PL)$, then $S = DT(S')$ is conflict-serializable.*

The theorem states that S is conflict-equivalent to the serial schedule that orders the transactions of S according to the following rule:

- Take as the first transaction the one that executes the first unlock operation.
- Take as the second transaction the one that executes the first unlock operation among the remaining (N-1) transactions in S.
- ...
- Take the last transaction as the N-th transaction.

1.2 System functionalities

The system presented is the implementation of an active Two-phase locking scheduler. The main functionalities of the scheduler are:

- Given a schedule S without lock operations checks if S is in the 2PL class with exclusive lock (or shared and exclusive locks).
- Use the lock anticipation when possible to try to avoid blocking transactions.
- Returns the output lock-extended schedule also for schedules not in the 2PL class. The output schedule can be obtained by resuming blocked transactions. Only when deadlocks are not present.
- For each schedule S in the 2PL class, returns a serial schedule conflict equivalent to S.

The scheduler accepts in input only read, write, and commit operations.

Table 1: Lock Table

	Exclusive	Shared
o1		T2,T3,T4
o2	T2	
...	...	...

2 Methods

To satisfy the requirements, the schedule will implement the following logic.

2.1 Computing the lock extended schedule

The scheduler computes one operation at a time to generate the lock-extended schedule, performing the following actions for each operation:

- Locking the element: To try to lock the element, the scheduler checks the lock table. The lock table is implemented as a hash table, given the object name it returns a pair containing the exclusive lock and the shared locks set. See Table 1.
  Depending on the lock's table state and the required lock (exclusive or shared), the schedule implements different behaviors:
  - The object is not locked by any transaction: the scheduler can update the locking table and insert the lock into the lock-extended schedule.
  - The object is already locked by the same transaction: the operation can be executed and inserted into the lock-extended schedule.
  - The object is locked with an exclusive lock not by the same transaction and an exclusive lock is required: the scheduler tries to unlock the object, if the unlock is successful, the scheduler can update the locking table and insert the lock into the lock-extended schedule.
    There are two possibilities in which the unlock operation can not be done: the object must be accessed again by the same transaction, in this case, the scheduler has only one choice, it needs to block the transaction that has required the lock on the object, and the second possibilities is that the shrinking phase can not be started since the transaction who has the lock on the object needs to lock another object in the future schedule. In the latter case, the scheduler can try to anticipate all the remaining locks to be able to start the shrinking phase and then unlock the object, else the transaction should be blocked as in the first case. The scheduler can anticipate the lock on an object only if any other transactions don't require an exclusive lock on the same object before the operation that requires the lock anticipation.
    To apply the unlock of the object, the scheduler removes the lock

Table 2: Wait-For-Graph

Blocked Transaction	Transaction	Object
T1	T2	o1
T3	T2	o2
...	...	...

from the lock table and writes into the lock-extended schedule the unlock operation.

- The object is locked with a shared lock by the same transaction and an exclusive lock is required: we are in the case of lock upgrading. The scheduler first tries unlocking all the shared locks except the one we need to upgrade, removes it from the lock table, and finally executes the exclusive lock on the object.
- The object is locked with a shared lock not by the same transaction and an exclusive lock is required: as above the scheduler needs to unlock all the shared locks and then compute the exclusive lock.

The outcomes of the lock operation are:

- The object is locked: the scheduler can continue the computation of the schedule.
- The transaction is blocked: the scheduler adds the blocked transaction to a Wait-For-Graph to recognize deadlocks.
  The Wait-For-Graph is represented by an adjacency list, and it is implemented using a hash table. For each transaction blocked, the scheduler stores the transaction that is waiting for and the conflicting object. Since every transaction can wait at most one transaction, the adjacency list contains only one entry per transaction. See Table 2. By using the adjacency list we can store the data structure in $\mathcal{O}(|V|)$ space since there is only one edge for each node, and the operation required is inserting a new edge with a time complexity of $\mathcal{O}(1)$, and removing an edge with a time complexity of $\mathcal{O}(1)$.
  The scheduler also adds the operation blocked to some data structures to resume it as soon as possible.
- Deadlock: when the scheduler tries to add an edge in the graph, it checks if a cycle occurs. If a cycle occurs the scheduler has recognized a deadlock caused by the last transaction blocked.

- Executing the operation: if the lock operation is successful, the scheduler can execute the operation and insert it into the lock-extended schedule.
- Unlock the element: after the execution of the operation, the scheduler implements the policy of unlocking the objects as soon as possible to avoid the unlocking of all the objects at the end of the schedule. Unlocking the

objects at the end of the schedule, in fact, requires keeping in memory all the objects locked by the transaction, or at least scanning the schedule from the start to the end to find the objects locked by the transaction.

- Resume transactions: after unlocking the objects, the scheduler tries to unlock each blocked transaction by checking if the object that has blocked the transaction is been unlocked by the previous scheduler action. If the object is unlocked, the scheduler can remove the transaction from the Wait-For-Graph and resume the blocked operation of the transaction. The scheduler is able

2.2 Computing the data action projection

After computing the lock-extended schedule, the scheduler computes the data action projection. The scheduler scans the lock-extended schedule and for each operation, it checks if the operation is a read, write, or commit. If the operation is a read, write, or commit, the scheduler adds the operation to the data action projection. At this point, the scheduler can check if the input schedule is in the 2PL class with exclusive locks or with shared and exclusive locks by comparing the input schedule with the data action projection. If the input schedule is equivalent to the data action projection of the lock-extended schedule, the input schedule is in the 2PL class with exclusive locks or with shared and exclusive locks. We can observe that every lock-extended schedule obtained by blocking transactions is not in the 2PL class, since the scheduler can not execute the operations in the same order as the input schedule, so to execute all the operations the scheduler needs to stop some transactions, and resume them later.

2.3 Computing the serial conflict-equivalent schedule

By using the theorem 1, for each schedule in the 2PL class, the scheduler can compute the serial conflict-equivalent schedule. The scheduler scans the lock-extended schedule and for each operation, it checks if the operation is an unlock. If the operation is the first unlock of the transaction, the scheduler adds the transaction to the serial conflict-equivalent schedule.

3 Comments

3.1 Output logging

The scheduler, in addition to the output schedule, returns a log of the operations performed by the scheduler. The log contains the following information:

- Successful/unsuccessful lock anticipations.
- Blocked transactions.
- Resumed transactions.
- Deadlocks detected, and the transactions involved in the deadlock.

3.2 Examples

3.2.1 Example 1

Input schedule S taken from an exam of the Data Management course:

$$w1(Z)r2(X)w3(X)r3(Y)w4(Y)w4(X)r2(Y)r1(Y)w2(Z)$$

Result: S is not in the 2PL class with shared and exclusive locks.
Log:

- During locking w3(X), unable to unlock the object X locked by transaction 2. Unable to anticipate lock for r2(Y) due to w4(Y)
- During locking w3(X), transaction 3 blocked, waiting for transaction 2 on object X
- During locking w4(X) and unlocking X, anticipate lock for r2(Y)
- During locking w4(X) and unlocking X, anticipate lock for w2(Z)
- During locking r2(Y), unable to unlock the object Y locked by transaction 4. Unable to anticipate lock for w4(X) due to w3(X)
- During locking r2(Y), transaction 2 blocked, waiting for transaction 4 on object Y
- Deadlock detected, the Wait-For-Graph contains the following cycle T2 T4 T2

In this example, the scheduler computes the input S and checks if S is in the 2PL class with shared and exclusive locks. To return the negative results, it is sufficient to stop at the first transaction blocked, after the impossibility of anticipating the lock on the object Y, but the scheduler continues the computation to try to return in the output the lock-extended schedule, in which its data action projection is in the 2PL class with shared and exclusive locks since it is generated by a 2PL scheduler. Unfortunately, the scheduler is not able to compute the lock-extended schedule since it recognizes a deadlock.

3.2.2 Example 1

Input schedule S' taken from the previous exercise by removing the operation w4(Y):

$$w1(Z)r2(X)w3(X)r3(Y)w4(X)r2(Y)r1(Y)w2(Z)$$

Result: S' is in the 2PL class with shared and exclusive locks.
Lock-extended schedule:

$$xl1(Z)w1(Z)sl2(X)r2(X)sl2(Y)sl1(Y)u1(Z)xl2(Z)u2(X)xl3(X)$$
$$w3(X)sl3(Y)r3(Y)u3(Y)u3(X)xl4(X)w4(X)u4(X)r2(Y)u2(Y)$$
$$r1(Y)u1(Y)w2(Z)u2(Z)$$

DT(S'):

$$w1(Z)r2(X)w3(X)r3(Y)w4(X)r2(Y)r1(Y)w2(Z)$$

Serial conflict-equivalent schedule:

$$T_1T_2T_3T_4$$

Log:

- During locking w3(X) and unlocking X, anticipate lock for r2(Y)
- During locking w3(X) and unlocking X, anticipate lock for w2(Z)
- During locking w2(Z) and unlocking Z, anticipate lock for r1(Y)

In this example, we can observe that there are no blocked transactions, so the input schedule is equivalent to the data action projection of the lock-extended schedule.

3.3 Managing blocked transactions

Most of the effort in this implementation of the scheduler is to compute the lock-extended schedule managing and resuming the blocked transactions. By doing this, the schedule should maintain information on operations executed, operations blocked who needs to be resumed as soon as possible, and operations to be executed in the future. The scheduler implements the following data structures to manage the blocked operations:

- A counter on the computed operations, either blocked or executed.
- A list of blocked operations. When a transaction is blocked, the operation is added to this list. Also when the scheduler computes an operation in which the transaction is already blocked, the operation is added to the blocked operations list. When we can resume a transaction, all the operations in the blocked operations list of the same transaction are removed from this list, and the scheduler tries to execute them in the same order as in the input schedule. Of course, if the transaction should be blocked again, the remaining operations are inserted in the blocked operations list.

3.4 Testing

To ensure the accuracy of the scheduler during development and upon completion, testing is essential. In the following, there is a list of input schedules that resume the most important functionalities that the scheduler should implement to check if a schedule is in the 2PL class.

3.4.1 Lock upgrade

Input schedule S:

$$r1(x)r2(x)w1(x)$$

Result: S is in the 2PL class with shared and exclusive locks.
Lock-extended schedule:

$$sl1(x)r1(x)sl2(x)r2(x)u2(x)xl1(x)w1(x)u1(x)$$

3.4.2 Blocking and resuming transactions

Input schedule S:

$$r1(x)w2(x)w1(x)$$

Result: S is not in the 2PL class with shared and exclusive locks.
Lock-extended schedule:

$$sl1(x)r1(x)xl1(x)w1(x)u1(x)xl2(x)w2(x)u2(x)$$

Log:

- During locking w2(x), transaction 2 blocked, waiting for transaction 1 on object x
- Transaction 2 resumed, object x unlocked by transaction 1

3.4.3 Lock anticipation

Input schedule S:

$$r2(x)r1(x)w2(x)w1(y)$$

Result: S is in the 2PL class with shared and exclusive locks.
Lock-extended schedule:

$$sl2(x)r2(x)sl1(x)r1(x)xl1(y)u1(x)xl2(x)w2(x)u2(x)w1(y)u1(y)$$

Log:

- During locking w2(x) and unlocking x, anticipate lock for w1(y)

3.4.4 Unable to lock anticipate

Input schedule S:

$$w1(x)w2(x)r3(y)w1(y)$$

Result: S is not in the 2PL class with shared and exclusive locks.
Lock-extended schedule:

$$xl1(x)w1(x)sl3(y)r3(y)u3(y)xl1(y)w1(y)u1(y)u1(x)xl2(x)w2(x)u2(x)$$

Log:

- During locking w2(x), unable to unlock the object x locked by transaction 1. Unable to anticipate lock for w1(y) due to r3(y)
- During locking w2(x), transaction 2 blocked, waiting for transaction 1 on object x
- Transaction 2 resumed, object x unlocked by transaction 1

3.4.5 Deadlock

Input schedule S:

$$w1(x)w2(y)w2(x)w1(y)$$

Result: S is not in the 2PL class with shared and exclusive locks.
Log:

- During locking w2(x) and unlocking x, anticipate lock for w1(y)
- During locking w1(y) and unlocking y, anticipate lock for w2(x)
- Deadlock detected, caused by transaction 1 on transaction 2

4 Conclusions

The scheduler can check if an input schedule with read, write, and commit operations is in the 2PL class with exclusive locks or with shared and exclusive locks. It also provides in output the lock-extended schedule, the data action projection, and the serial conflict-equivalent schedule. The scheduler also implements the lock anticipation to try to avoid blocking transactions. Deadlocks are only recognized by the scheduler, and the scheduler does not implement any deadlock resolution policy.
We assume that we know all future operations to compose our input schedule. A complete scheduler implementation should resolve deadlocks by rolling back the transaction causing the deadlock and returning the lock-extended schedule.

References

[1] https://github.com/simonescaccia/2PL-scheduler