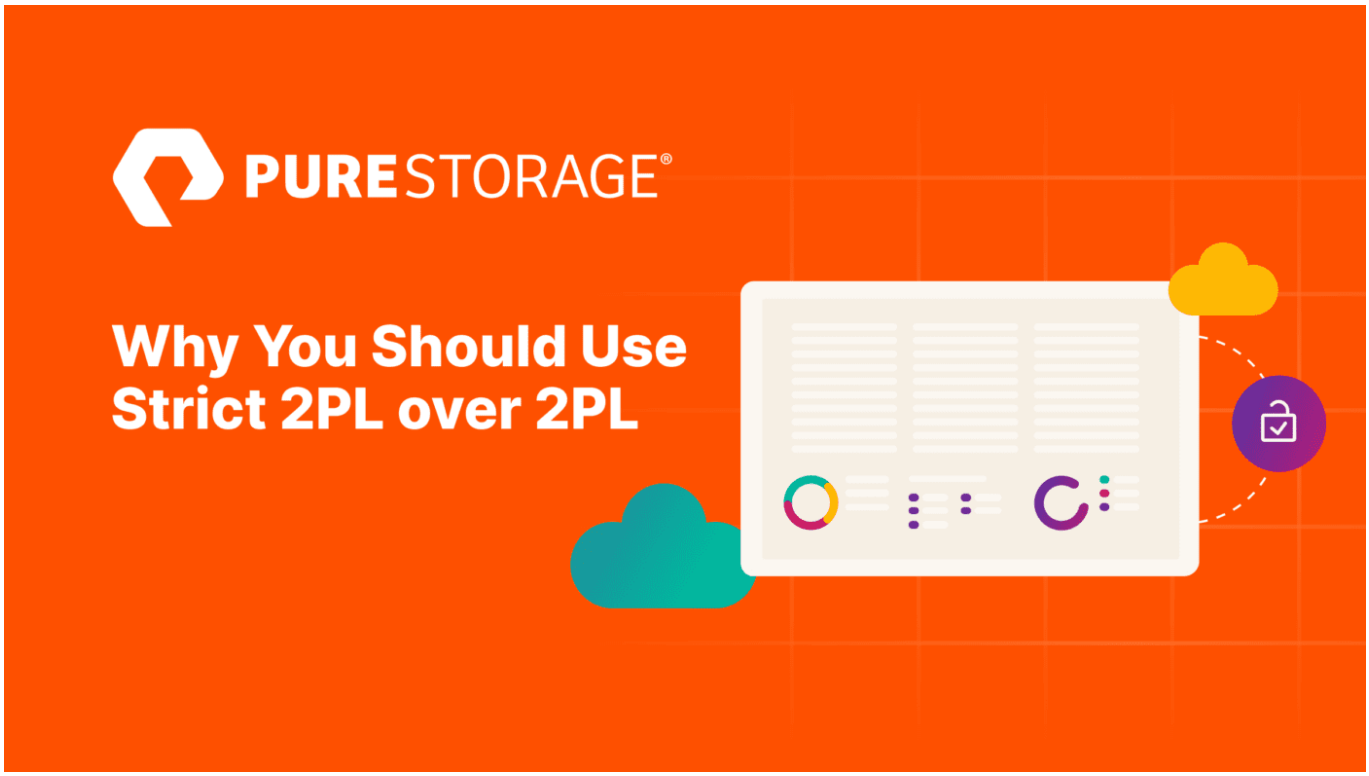
Why You Should Use Strict 2PL over 2PL



Database management systems (DBMSs) are used for storing and updating data, and it needs to happen without corrupting queries. This can be difficult to achieve when you have large enterprise applications that are constantly running queries against data to add, update, and delete it. Locking systems in place can help stop "dirty reads," which can happen if the database doesn't properly handle simultaneous reads and writes on a specific record.

What Is Two-phase Locking (2PL) Protocol?

The two-phase locking (2PL) protocol (also known as basic 2PL) is a method used in database management systems to lock data from concurrent transactions. For example, one user may need to read data from a record while another user simultaneously attempts to change or update that data. This leads to "dirty reads," where some of the record is updated and some isn't during the read

transaction.

The 2PL protocol is a standard to lock a record, perform the query, and then release the lock for the next transaction. When the first query starts, it acquires a lock, locks the record, and then releases the lock. When the transaction obtains a lock, it can't release it until the transaction is finished.

What Is Strict Two-phase Locking (2PL) Protocol?

The 2PL protocol gradually obtains locks and then gradually releases them when they're no longer needed. The difference between the basic 2PL protocol and strict 2PL is that strict 2PL releases the lock immediately after the commit command executes. Instead of gradually releasing locks one by one, the strict 2PL protocol releases them at once.

Strict 2PL works similarly to the basic 2PL protocol at first. It gradually obtains locks as needed, but the locks are only released after a commit. A commit in database terms is the command used to tell the database to execute the queries now instead of rolling back. Only after the commit phase do records change so that the developer has the option to roll back actions prior to committing them.

How Strict 2PL Works with an Example

Busy databases deal with numerous concurrent queries, but let's consider two transactions with strict locking. The first transaction obtains its locks, and during release of the locks, the second transaction contains its own locks. This cascading effect ensures that any queries set to read changed data won't read data until the initial transaction either successfully completes (commits) or rolls back.

If the first transaction is successful, it commits its actions and then immediately releases the locks. Then, the second transaction can read the updated data. However, if the first transaction fails and rolls back, the second transaction must also roll back to avoid reading data that hasn't been updated yet.

Is It Called Strict or Rigorous 2PL?

In some scenarios, the strict 2PL protocol might release read locks earlier instead of releasing all the locks at once. This benefits read queries where data isn't changed and locks can be released for the next transaction to obtain data. Rigorous 2PL adds more restrictive lock standards to transactions than strict 2PL.

In rigorous 2PL databases, all locks can't be released until a transaction commits or rolls back actions. According to Wikipedia, rigorous 2PL has been the choice for database management systems since the 1970s and continues to be the most commonly used locking protocol. Rigorous 2PL will guarantee a cascadeless recoverability to allow other transactions to execute without relying on previous ones to commit.

Why Strict 2PL Is Better than 2PL

If you can't use rigorous 2PL, then the next best option is strict 2PL. The strict 2PL mechanism has the advantage of guaranteeing recoverable transactions. For example, if you have transactions that rely on previous ones for accuracy, you don't want to run a second transaction if the first one fails. If the first transaction fails to update, then the second one would also abort. The second transaction would only continue when the first transaction successfully commits.

When to Use 2PL Rather than Strict 2PL

In some scenarios, you want the next transaction to execute even if the first one fails and you don't want to cascade failures across all transactions. The 2PL protocol would be useful if transactions don't rely on previous transaction success, so you would use this method instead of the cascading failures that are used in strict 2PL.

With 2PL, all assets must be locked or none of them will be locked until they all become available. It leaves less flexibility for locks to be available when needed, but it could lead to dirty reads on high-volume databases.

Conclusion

Most database configurations are done by administrators, but understanding the difference between the locking mechanisms can help you choose the best database engine for your application environment. As you develop your queries, you can use locking mechanisms to avoid crashes, deadlocks, and dirty reads.

Written By:

Pure Storage