Abstract

Software Transactional Memory (STM) Systems have been proposed in order to make parallel programs easier to develop and verify compared to conventional lock-based programming techniques. However, conventional STMs do not scale in performance to a large number of concurrent threads for several classes of applications. While the atomicity semantics of traditional STMs greatly simplify the correct sharing of data between threads, these same atomicity semantics incur a large penalty in program execution time.

Conventional Software Transactional Memory (STM) systems, provide an “atomic” construct to express atomic transactions. When a transaction is done executing, it is checked for conflicts with other concurrent transactions. A conflict is said to have occurred if this transaction’s read-set overlaps with another concurrent transaction’s write-set. When a conflict occurs a typical policy used in many STMs is to abort the transaction that read the stale values and restart it. We will call this a “strict consistency” policy.

In many applications enforcing the strict consistency policy over the entire read-set of a transaction can lead to a large number of unnecessary aborts. This is because some shared data is not required to be completely consistent for correct execution of the program, that is, some reader transactions could have safely read some stale values and proceed. We will call this a “strict consistency” policy.

In many applications enforcing the strict consistency policy over the entire read-set of a transaction can lead to a large number of unnecessary aborts. This is because some shared data is not required to be completely consistent for correct execution of the program, that is, some reader transactions could have safely read some stale values and proceed. We will call this a “strict consistency” policy.

In many applications enforcing the strict consistency policy over the entire read-set of a transaction can lead to a large number of unnecessary aborts. This is because some shared data is not required to be completely consistent for correct execution of the program, that is, some reader transactions could have safely read some stale values and proceed. We will call this a “strict consistency” policy.

In many applications enforcing the strict consistency policy over the entire read-set of a transaction can lead to a large number of unnecessary aborts. This is because some shared data is not required to be completely consistent for correct execution of the program, that is, some reader transactions could have safely read some stale values and proceed. We will call this a “strict consistency” policy.

In many applications enforcing the strict consistency policy over the entire read-set of a transaction can lead to a large number of unnecessary aborts. This is because some shared data is not required to be completely consistent for correct execution of the program, that is, some reader transactions could have safely read some stale values and proceed. We will call this a “strict consistency” policy.

In many applications enforcing the strict consistency policy over the entire read-set of a transaction can lead to a large number of unnecessary aborts. This is because some shared data is not required to be completely consistent for correct execution of the program, that is, some reader transactions could have safely read some stale values and proceed. We will call this a “strict consistency” policy.

In many applications enforcing the strict consistency policy over the entire read-set of a transaction can lead to a large number of unnecessary aborts. This is because some shared data is not required to be completely consistent for correct execution of the program, that is, some reader transactions could have safely read some stale values and proceed. We will call this a “strict consistency” policy.

In many applications enforcing the strict consistency policy over the entire read-set of a transaction can lead to a large number of unnecessary aborts. This is because some shared data is not required to be completely consistent for correct execution of the program, that is, some reader transactions could have safely read some stale values and proceed. We will call this a “strict consistency” policy.

In many applications enforcing the strict consistency policy over the entire read-set of a transaction can lead to a large number of unnecessary aborts. This is because some shared data is not required to be completely consistent for correct execution of the program, that is, some reader transactions could have safely read some stale values and proceed. We will call this a “strict consistency” policy.

In many applications enforcing the strict consistency policy over the entire read-set of a transaction can lead to a large number of unnecessary aborts. This is because some shared data is not required to be completely consistent for correct execution of the program, that is, some reader transactions could have safely read some stale values and proceed. We will call this a “strict consistency” policy.

In many applications enforcing the strict consistency policy over the entire read-set of a transaction can lead to a large number of unnecessary aborts. This is because some shared data is not required to be completely consistent for correct execution of the program, that is, some reader transactions could have safely read some stale values and proceed. We will call this a “strict consistency” policy.