2.5 Concurrency of Operations on B-Trees

We have already seen that the problem of concurrency control is a critical issue in database management systems. In this section, we will explore some of the solutions that have been proposed to address this problem.

1. Introduction

The introduction of B-Trees can be used as a data structure in a multi-user environment. The solution presented here uses simple locking protocols. Thus, the problem of concurrency control can be handled in a straightforward manner. An additional feature is provided which allows the system to detect and handle conflicts automatically. Another feature that has been added is the ability to detect and handle conflicts at the time of insertion or deletion of data.

2. Definitions

In this section, we define the basic terms and concepts that are used in the discussion of concurrency control on B-Trees.

3. The problem of concurrency control on B-Trees

The problem of concurrency control on B-Trees is related to the following aspects:

- The concurrent modification of the database
- The requirement for efficient access to the database
- The need for consistency between concurrent transactions

In this section, we discuss some of the complications that arise when concurrent access is allowed to the database.

4. The solution to the concurrency control problem on B-Trees

The solution to the concurrency control problem on B-Trees is based on the use of simple locking protocols. These protocols are designed to ensure that conflicting operations are not allowed to execute simultaneously.

5. Conclusion

In conclusion, we have presented a simple solution to the concurrency control problem on B-Trees. This solution is effective and efficient, and it can be used in a variety of environments.

B. Bayer and E. F. Codd

IBM Research Laboratory, San Jose, CA 95193, USA

R. Bayer and M. Schkolnick

Concurrent Operations on B-Trees
\[
\begin{align*}
&\text{1. If a process or thread is active in the transaction, it will acquire the lock for the shared objects.}
\end{align*}
\]
A Generalized Solution

A counterexample to the idea that a lock's ownership implies that other locks must be converted into x-locks is provided by the following example. Consider a situation where two transactions, A and B, are running concurrently on a shared resource. Transaction A has acquired an x-lock on the resource, while transaction B attempts to acquire an s-lock. If transaction A releases its lock before transaction B can acquire its lock, transaction B will be blocked, even though it attempted to acquire a lock on a resource that is already locked by transaction A. This example demonstrates that a lock's ownership does not prevent other locks from being acquired on the same resource.

Protocol for a reader is in Solution 1 with respect to the protocol for a writer.

Using this protocol, we can resolve the race in solution 1 by introducing a counterexample to the idea that a lock's ownership implies that other locks must be converted into x-locks. Consider a situation where two transactions, A and B, are running concurrently on a shared resource. Transaction A has acquired an x-lock on the resource, while transaction B attempts to acquire an s-lock. If transaction A releases its lock before transaction B can acquire its lock, transaction B will be blocked, even though it attempted to acquire a lock on a resource that is already locked by transaction A. This example demonstrates that a lock's ownership does not prevent other locks from being acquired on the same resource.

Protocol for a writer is in Solution 2.
Declarative Features of the Generalized Solution

The preceding sections have shown the amount of concurrency provided by the algorithms. To combine c and d, a lock must be a lock on both c and d. However, a lock on one of the objects does not provide any information about the other object. A lock on both c and d is needed. After the lock is acquired, the lock on c is released and an attempt is made to combine c and d.

Figure 6: Example of update

- (a)
- (b)
- (c)

2.5 Concurrency of Operations on B-Trees

- Figure 7: Procedure for additional x-locks

- (a)
- (b)
- (c)
- (d)
Lemma 2. A process holds a 2-lock on a node in Z then either
\( (m) \leq (\bar{m}) \)

Proof of Lemma 2. From Observation 1.
Recall that \( (m) = \{m, 2 \} \) is the local of node \( m \).

Lemma 1. A process holds a 2-lock on a node in Z if and only if
\( (m) \leq (\bar{m}) \)

Proof of Lemma 1. From Observation 2.

**Example of deadlock:**

![Diagram of a network with nodes and edges representing a deadlock scenario.](attachment:image.png)

DEFINITION: A process is called a process if it has a pending lock request for a lock.

DEFINITION: A 2-process is defined as a process that holds or requests a 2-lock on a node in Z.

DEFINITION: Given two processes \( X \) and \( Y \), we say that \( X \) waits on \( Y \) if and only if
\( Y \) has a pending lock request for a lock on node in Z. If \( Y \) has a pending lock request on a node in Z then \( Y \) is said to be the critical node of \( X \). If \( Y \) is not the critical node of \( X \), then \( X \) is not waiting on \( Y \).

**Theorem:** A conflict in node in Z is a conflict if and only if
\( (m) \leq (\bar{m}) \)

Proof: By contradiction. Suppose \( (m) \leq (\bar{m}) \) and \( (m) \not\leq (\bar{m}) \).

Let \( X \) be the process that holds a 2-lock on node in Z while \( Y \) is waiting on \( X \). Since each one of Solutions 1 and 2 can be shown to be deadlock-free, we now proceed to show that a deadlock is possible.

Suppose that \( X \) and \( Y \) are in the deadlock process. By Lemma 1, if \( (m) \leq (\bar{m}) \) then \( (m) \not\leq (\bar{m}) \).

Since \( (m) \not\leq (\bar{m}) \), \( X \) cannot hold a 2-lock on node in Z. Therefore, \( X \) is not deadlock-free.

Hence, a deadlock is possible.
2.5 Concurrency of Operations on B-Trees

For these operations, we know that $A$ cannot be a process.

If we have a node which is not possible (we get that $A$ is not possible), we get that $A$ would hold to hold to hold $A$.

Copyright © 2018 Elsevier Inc. All rights reserved.

Theorem: The guaranteed solution is deadlock free.

Lemma 4 allows us to show:

Proof of the lemma.

Lemmas 4 and 5 hold $A$ to hold $A$ to hold $A$.

Copyright © 2018 Elsevier Inc. All rights reserved.

Proof of the theorem.

Lemmas 4 and 5 hold $A$ to hold $A$ to hold $A$.
suitable values of $P$ and $\Xi$ can be chosen. We will assume that all readers and updaters access the structure using the same $P$ and $\Xi$ (as will be explained below, this may not be the case, but helps to evaluate a strategy).

There are two main components in the cost of a given solution. One is the time spent by processes waiting for locks to be removed before they can proceed. The second one is the time overhead due to placing locks on the nodes, converting locks, or repeating part of an analysis.

Assume a tree as in Figure 9, and a given number $n_r$ of readers and $n_u$ of updaters. The updaters will place $\rho_k$-locks from level $\alpha$ down to level $\alpha-P+1$. From level $\alpha-P$ down to $\Xi+1$ they will place $\alpha$-locks and finally, for the last $\Xi$ levels, they will place $\zeta$-locks.

Now, in the upper $P$ levels there are no conflicts, since updaters and readers use compatible $\rho_k$ and $\rho_k$ locks. But in level $\alpha-P$, the updaters set up $\alpha$-locks which are incompatible among themselves. Thus, at this level, some updaters will have to wait for other updaters.

Let $v_i$ denote the number of nodes of the tree at level $i$. We will assume that each updater has equal probability $\frac{1}{v_i}$ of scanning each node at level $i$ when traversing the tree from the root to a leaf node. Then, since there are $n_u$ updaters, the expected number of nodes which are visited by the $n_u$ updaters at level $\alpha-P$ is given by

$$\Phi(\alpha-P, n_u) = \frac{1}{v_{\alpha-P}} \left(1 - \left(1 - \frac{1}{v_{\alpha-P}}\right)^{n_u}\right).$$

To obtain this expression, note that $\left(1 - \frac{1}{v_{\alpha-P}}\right)^{n_u}$ is the probability that a given node will not be scanned by any of the $n_u$ updaters. Thus, $1 - \left(1 - \frac{1}{v_{\alpha-P}}\right)^{n_u}$ gives the probability that a given node will be scanned by at least one updater. The expected number of nodes which will be scanned by at least one updater is then $\Phi(\alpha-P, n_u)$.

This expression also gives the expected number of updaters that will proceed down the tree (from level $\alpha-P$) without waiting for an $\alpha$-lock to be granted.

Thus, the expected number of updaters that will wait is given by:

$$W_u = n_u - \Phi(\alpha-P, n_u).$$

The $\Phi(\alpha-P, n_u)$ updaters that can proceed downwards from level $\alpha-P$ will do so without interfering with each other. When they get to level $\Xi$ they may interact with readers. Assuming the updaters acquire the $\zeta$-locks before the readers request $\rho_k$-locks (this will give a worst case value for the quantity being computed), the expected number of readers that wait is:

$$W_r = \begin{cases} n_r - \frac{\Phi(\alpha-P, n_u)}{v_{\Xi}} & \text{(if } \Xi \neq 0) \\ 0 & \text{(if } \Xi = 0). \end{cases}$$

$W_u$ and $W_r$ together give a measure of the number of processes that will have to wait when accessing the tree. Besides this component of the cost of a $(P, \Xi)$ solution there is the overhead cost involved. This cost has three subcomponents. One is given by the fact that, after scanning the tree from the root to the leaf, a process may find that all his processing has to be repeated (this happens if in Step 5 of the protocol an updater finds that it still holds a $\rho_k$-lock). We will measure this component by computing $Q$, the expected number of nodes per updater that are scanned again to repeat an analysis. The second subcomponent is $C_{\zeta}$, the expected number of $\zeta$-locks that an updater will convert into $\alpha$-locks. Finally, the third subcomponent is $C_{\Xi}$, the expected number of $\alpha$-locks that an updater will convert into $\zeta$-locks.

To compute $Q$, we will assume that all updaters are performing insertions. In this case, one out of $k$ updaters will, on the average, cause a split of a node at the leaf level which propagates up the tree; one out of $k^2$ updaters will, on the average, cause a split of a node at level 2 which will propagate up the tree, and so on. Thus, we consider the probability that an updater will modify a node at level $i$ or above to be $\left(\frac{1}{k}\right)^{i-1}$.

An updater will repeat his analysis if it causes a node at level $\alpha-P+1$ or above to be modified. Since when this happens, $\alpha$ nodes will be scanned again, we have that

$$Q = \begin{cases} \alpha \cdot \left(\frac{1}{k}\right)^{\alpha-P} & \text{if } P \neq 0 \\ 0 & \text{if } P = 0. \end{cases}$$

(Note that if $P=0$, there is no retry involved.) To compute $C_{\zeta}$ we note that an updater will convert $\zeta$-locks into $\alpha$-locks whenever it reaches Step 5 of the protocol and discovers it holds an $\alpha$-lock, but not a $\rho_k$-lock. This, in turn happens only if the update will modify nodes at a level $\Xi+1$ or higher, but lower than level $\alpha-P+1$. The number of locks to be modified in this case is always $\Xi$. Thus,

$$C_{\zeta} = \begin{cases} \Xi \cdot \left[\left(\frac{1}{k}\right)^{\zeta} - \left(\frac{1}{k}\right)^{\alpha-P}\right] & \text{if } \alpha > P + \Xi \\ 0 & \text{otherwise}. \end{cases}$$
### 7. Extensions to Sequential Readers

| C | f | N | n | m | z | y | x | η | θ | φ | χ | l | z | μ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 2.5 Concurrency on B-Trees

The trick here is to use a custom read lock for the affected nodes. This avoids blocking other parts of the system while ensuring that the update is applied atomically.

Table 1

| C | f | N | n | m | z | y | x | η | θ | φ | χ | l | z | μ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
In the Operating Systems Project BSN at the Technical University in Munich...

8 Conditions and Information Constraints

<table>
<thead>
<tr>
<th>Condition</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. If can be shown that with the modification, the new protocol is self-decoupling.</td>
<td></td>
</tr>
<tr>
<td>b. If the pattern is not broken, and if a change is needed in the pattern protocol.</td>
<td></td>
</tr>
<tr>
<td>c. If the pattern is not broken, and if a change is needed in the pattern protocol.</td>
<td></td>
</tr>
<tr>
<td>d. If the pattern is not broken, and if a change is needed in the pattern protocol.</td>
<td></td>
</tr>
</tbody>
</table>

Consider the following scenarios:

### Table 1 (Continued)
References

- Acknowledgments: The authors thank...