of taking a logical lock is translated into taking physical locks, in such a way that the logical lock appears to be granted as an atomic action.

Global Transactions, Local Subtransactions, and Serializability

Similarly, global transactions may be composed of many local subtransactions, each executing at a different site, and it is the job of the database system to assure that the global transactions behave in a serializable manner. The notion of "serializability" in a distributed database is a natural generalization of the definition given in Chapter 9. A schedule of transactions on a distributed database is *serializable* if its effect on the logical items is the same as that of the transactions executing serially, that is, one-at-a-time, with each in its turn performing all of its subtransactions at all the sites before the next transaction begins.

Example 10.2: Suppose transaction T transfers \$10 from account A to account B. Suppose also that T is initiated at node N_1, copies of A exist at nodes N_2 and N_3, and copies of B exist at N_4 and N_5. Then T must initiate two subtransactions that deduct 10 from the physical copies of item A at N_2 and N_3, and also must initiate two subtransactions that add 10 to the physical copies of B at N_4 and N_5. Thus, the global transaction T consists of local transactions at each of the five nodes $N_1, \ldots, N_5$, and the effects of these transactions must be coordinated and made serializable. For example, the change to one copy of an item must not be made in the permanent database if the same change to the other copy is not guaranteed to be made eventually. That requirement holds even if, say, N_3 fails after A is updated at N_2. In that case, we must be sure that A will be updated at N_3 when that node recovers. □

10.2 DISTRIBUTED LOCKING

Our first task, as we extend concurrency control concepts from the single-site case to the distributed case, is to consider how locks on logical, or global, items can be built from locks on physical, or local, items. The only thing we can do with physical items is take a lock on a single physical copy A_i of a logical item A, by requesting the lock from the lock manager that is local to the site of A_i.

Whatever we do with physical copies must support the properties we expect from locks on the logical items. For example, if we use read- and write-locks, then we need to know that at no time can two transactions hold write-locks, or a read- and a write-lock, on the same logical item. However, any number of transactions should be able to get read-locks on the same logical item at the same time.

If there is but one copy of an item, then the logical item is identical with its one physical copy. Thus, we can maintain locks on the logical item if and only if we maintain locks on the copy correctly. Transactions wishing to lock

an item A with one copy send lock-request messages to the site at which the copy resides. The lock manager at that site can grant or deny the lock, sending a back a message with its decision in either case.

However, if there are several copies of an item, then the translation from physical locks to logical locks can be accomplished in several ways, each with its advantages. We shall consider some of these approaches and compare the numbers of messages required by each.

Write-Locks-All—Read-Locks-One

A simple way to maintain logical locks is to maintain ordinary locks on copies of items, and require transactions to follow a protocol consisting of the following rules defining locks on logical items.

1. To obtain a read-lock on logical item A, a transaction may obtain a read-lock on any copy of A.
2. To obtain a write-lock on logical item A, a transaction must obtain write-locks on all the copies of A.

This strategy will be referred to as *write-locks-all*.

At each site, the rules for granting and denying locks on copies are exactly the same as in Chapter 9; we can grant a read-lock on the copy as long as no other transaction has a write-lock on the copy, and we can only grant a write-lock on the copy if no other transaction has either a read- or write-lock on the copy.

The effect of these rules is that no two transactions can hold a read- and write-lock on the same logical item A at the same time. For to hold a write-lock on logical item A, one transaction would have to hold write-locks on all the physical copies of A. However, to hold a read-lock on A, the other transaction would have to hold a read-lock on at least one copy, say A_1. But the rules for locks on the physical copy A_1 forbid a transaction from holding a read-lock at the same time another transaction holds a write-lock. Similarly, it is not possible for two transactions to hold write-locks on A at the same time, because then there would have to be conflicting write-locks on all the physical copies of A.

Analysis of Write-Locks-All

Let us see how much message traffic is generated by this locking method. Suppose that n sites have copies of item A. If the site at which the transaction is running does not know how many copies of A exist, or where they are, then we may take n to be the total number of sites.[2] To execute `WLOCK A`, the trans-

[2] It is worth noting that considerable space and effort may be required if each site is to maintain an accurate picture of the entire distributed database, at least to the extent

action must send messages requesting a lock to all n sites. Then, the n sites will reply, telling the requesting site whether or not it can have the lock. If it can have the lock, then the n sites are sent copies of the new value of the item. Eventually, a message `UNLOCK A` will have to be sent, but we may be able to attach this message to messages involved in the commitment of the transaction, as discussed in Sections 10.4 and 10.5.

The messages containing values of items may be considerably longer than the lock messages, since, say, a whole relation may be transmitted. Thus, we might consider sending only the changes to large items, rather than the complete new value. In what follows, we shall distinguish between

1. *Control messages*, which concern locks, transaction commit or abort, and other matters of concurrency control, and
2. *Data messages*, which carry values of items.

Under some assumptions, control and data messages cost about the same, while under other conditions, data messages could be larger and/or more expensive. It is unlikely that control messages will be more costly than data messages. Sometimes, we shall have the opportunity to attach control messages to data messages, in which case we shall count only the data message.

When a transaction write-locks a logical item A, we saw by the analysis above that it needed to send $2n$ control messages and n data messages. If one of A's copies is at the site running the transaction, we can save two control messages and one data message, although we must still request and reserve a lock at the local site. If one or more sites deny the lock request, then the lock on A is not granted.

To obtain a read-lock, we have only to lock one copy, so if we know a site at which a copy of A exists, we can send `RLOCK A` to that site and wait for a reply granting the lock or denying the lock request. If the lock is granted, the value of A will be sent with the message. Thus, in the simplest case, where we know a site at which A can be found and the lock request is granted, only two messages are exchanged, one control (the request), and one data (the reply, including the value read). If the request is denied, it probably does not pay to try to get the read-lock from another site immediately, since most likely, some transaction has write-locked A, and therefore has locks on all the copies.

The Majority Locking Strategy

Now let us look at another, seemingly rather different protocol for defining locks on logical items.

---

of knowing what items exist throughout the database, and where the copies are. For this reason, among others, there is an advantage to using large items in a distributed environment.

1. To obtain a read-lock on logical item A, a transaction must obtain read-locks on a majority of the copies of A.
2. To obtain a write-lock on logical item A, a transaction must obtain write-locks on a majority of the copies of A.

We call this strategy the *majority* approach.

To see why majority locking works, note that two transactions each holding locks on A (whether they are read- or write-locks doesn't matter) would each hold locks on a majority of the copies. It follows that there must be at least one copy locked by both transactions. But if either lock is a write-lock, then there is a lock conflict for that copy, which is not permitted by the lock manager at its site. Thus, we conclude that two transactions cannot hold write-locks on logical item A simultaneously, nor can one hold a read-lock while the other holds a write-lock. They can, of course, hold read-locks on an item simultaneously.

Analysis of Majority Locking

To obtain a write-lock, a transaction must send requests to at least a majority of the n sites having copies of the item A. In practice, the transaction is better off sending requests to more than the minimum number, $(n+1)/2$,[3] since, for example, one site may not answer, or another transaction may be competing for the lock on A and already have locks on some copies. While a transaction receiving a denial or no response at all from one or more sites could then send the request to additional sites, the delay inherent in such a strategy makes it undesirable unless the chances of a failed node or a competing transaction are very small. We shall, however, take as an estimate of the number of request messages the value $(n+1)/2$ and use the same value for the number of response messages. Thus, assuming the lock is granted, $n+1$ control messages are used. Eventually n data messages with a new value of A will be sent, as well.

For a read, we must again send requests to at least $(n+1)/2$ nodes and receive this number of replies, at least one of which will be a data message including the value that is read along with the lock on this copy of A. If the transaction runs at the site of one of the copies, we can omit this message. Thus, we estimate the number of messages for a read operation at n control messages and one data messages (including a control portion).

Comparison of Methods

Before proceeding to some other methods for distributed locking, let us compare the write-locks-all and majority methods. Each uses n data messages for a write and one data message for a read. Write-locks-all uses $2n$ control messages for a write and one for a read, while majority uses $n+1$ for write and n for read. Thus,

if an equal number of read and write-locks are requested by typical transactions, there is no advantage to either method. On the other hand, if most locks are for reading, the write-locks-all method is clearly preferable, and if write-locks dominate, we might prefer the majority method.

The two methods differ in a subtle way that affects the likelihood of a deadlock. Using the write-locks-all approach, two transactions, each trying to write logical item A, that begin at about the same time are likely each to manage to obtain a lock on at least one copy of A. The result is a deadlock, which must be resolved by the system, in one of a number of costly ways. In comparison, under the majority approach, one of two competing transactions will always succeed in getting the lock on the item, and the other can be made to wait or abort.

A Generalization of the Two Previous Methods

The two strategies we have mentioned are actually just the extreme points in a spectrum of strategies that could be used. The "k-of-n" strategy, for any $n/2 < k \leq n$, is defined as follows:

1. To obtain a write-lock on logical item A, a transaction must obtain write-locks on any k copies of A.
2. To obtain a read-lock on logical item A, a transaction must obtain read-locks on any $n - k + 1$ copies of A.

To see that the method defines locks properly, observe that if one transaction held a read-lock on logical item A, it would hold read-locks on $n - k + 1$ copies of A, while if another transaction simultaneously held a write-lock on A, it would hold write-locks on k copies of A. Since there are only n copies of A, some copy is read-locked and write-locked by different transactions at the same time, an impossibility. Similarly, if two transactions simultaneously hold write-locks on logical item A, then each holds locks on k copies of A. Since $k > n/2$, some copy is write-locked by both transactions at the same time, another impossibility.

What we referred to as "write-locks-all" is strategy n-of-n, while the majority strategy is $(n + 1)/2$-of-n. As k increases, the strategy performs better in situations where reading is done more frequently. On the other hand, the probability that two transactions competing for a write-lock on the same item will deadlock, by each obtaining enough locks to block the other, goes up as k increases. It is left as an exercise that we cannot do better. That is, if the sum of the number of copies needed for a read-lock and a write-lock, or for two write-locks is n or less, then physical locks do not imply logical locks.

Primary Copy Protocols

A rather different point of view regarding lock management is to let the responsibility for locking a particular logical item A lie with one particular site, no matter how many copies of the item there are. At the extreme, one node of the network is given the task of managing locks for all items; this approach is the "central node method," which we describe shortly. However, in its most general form, the assignment of lock responsibility for item A can be given to any node, and different nodes can be used for different items.

A sensible strategy, for example, is to identify a *primary site* for each item. For example, if the database belongs to a bank, and the nodes are bank branches, it is natural to consider the primary site for an item that represents an account to be the branch at which the account is held. In that case, since most transactions involving the account would be initiated at its primary site, frequently locks would be obtained with no messages being sent.

If a transaction, not at the primary site for A, wishes to lock A, it sends one message to the primary site for A and that site replies, either granting or withholding the lock. Thus, locking the logical item A is the same as locking the copy of A at the primary site. In fact, there need not even be a copy of A at the primary site, just a lock manager that handles locks on A.

Primary Copy Tokens

There is a more general strategy than the simple establishment of a primary site for each item. We postulate the existence of *read-tokens* and *write-tokens*, which are privileges that nodes of the network may obtain, on behalf of transactions, for the purpose of accessing items. For an item A, there can be in existence only one write-token for A. If there is no write-token, then there can be any number of read tokens for A. If a site has the write-token for A, then it can grant a read or write-lock on A to a transaction running at that site. A site with only a read-token for A can grant a read-lock on A to a transaction at that site, but cannot grant a write-lock. This approach is called the *primary copy token* method.

If a transaction at some site N wishes to write-lock A, it must arrange that the write-token for A be transmitted to its site. If the write-token for A is already at the site, it does nothing. Otherwise, the following sequence of messages is exchanged:

1. N sends a message to all sites requesting the write-token.
2. Each site M receiving the request replies, either:
   a) M either has no (read or write) token for A, or it has, but is willing to relinquish it so N can have a write-token.
   b) M has a read- or write-token for A and will not relinquish it (because some other transaction is either using the token, or M has reserved that token for another site).