but instead should request the database system to force-output the buffer blocks. The database system in turn would force-output the buffer blocks to the database, after writing relevant log records to stable storage.

Unfortunately, almost all current-generation operating systems retain complete control of virtual memory. The operating system reserves space on disk for storing virtual-memory pages that are not currently in main memory; this space is called **swap space**. If the operating system decides to output a block B_x, that block is output to the swap space on disk, and there is no way for the database system to get control of the output of buffer blocks.

Therefore, if the database buffer is in virtual memory, transfers between database files and the buffer in virtual memory must be managed by the database system, which enforces the write-ahead logging requirements that we discussed.

This approach may result in extra output of data to disk. If a block B_x is output by the operating system, that block is not output to the database. Instead, it is output to the swap space for the operating system's virtual memory. When the database system needs to output B_x, the operating system may need first to input B_x from its swap space. Thus, instead of a single output of B_x, there may be two outputs of B_x (one by the operating system, and one by the database system) and one extra input of B_x.

Although both approaches suffer from some drawbacks, one or the other must be chosen unless the operating system is designed to support the requirements of database logging.

16.5.4 Fuzzy Checkpointing

The checkpointing technique described in Section 16.3.6 requires that all updates to the database be temporarily suspended while the checkpoint is in progress. If the number of pages in the buffer is large, a checkpoint may take a long time to finish, which can result in an unacceptable interruption in processing of transactions.

To avoid such interruptions, the checkpointing technique can be modified to permit updates to start once the checkpoint record has been written, but before the modified buffer blocks are written to disk. The checkpoint thus generated is a **fuzzy checkpoint**.

Since pages are output to disk only after the checkpoint record has been written, it is possible that the system could crash before all pages are written. Thus, a checkpoint on disk may be incomplete. One way to deal with incomplete checkpoints is this: The location in the log of the checkpoint record of the last completed checkpoint is stored in a fixed position, last-checkpoint, on disk. The system does not update this information when it writes the checkpoint record. Instead, before it writes the checkpoint record, it creates a list of all modified buffer blocks. The last-checkpoint information is updated only after all buffer blocks in the list of modified buffer blocks have been output to disk.

Even with fuzzy checkpointing, a buffer block must not be updated while it is being output to disk, although other buffer blocks may be updated concurrently. The write-ahead log protocol must be followed so that (undo) log records pertaining to a block are on stable storage before the block is output.

16.6 Failure with Loss of Nonvolatile Storage

Until now, we have considered only the case where a failure results in the loss of information residing in volatile storage while the content of the nonvolatile storage remains intact. Although failures in which the content of nonvolatile storage is lost are rare, we nevertheless need to be prepared to deal with this type of failure. In this section, we discuss only disk storage. Our discussions apply as well to other nonvolatile storage types.

The basic scheme is to **dump** the entire contents of the database to stable storage periodically—say, once per day. For example, we may dump the database to one or more magnetic tapes. If a failure occurs that results in the loss of physical database blocks, the system uses the most recent dump in restoring the database to a previous consistent state. Once this restoration has been accomplished, the system uses the log to bring the database system to the most recent consistent state.

One approach to database dumping requires that no transaction may be active during the dump procedure, and uses a procedure similar to checkpointing:

1. Output all log records currently residing in main memory onto stable storage.
2. Output all buffer blocks onto the disk.
3. Copy the contents of the database to stable storage.
4. Output a log record <dump> onto the stable storage.

Steps 1, 2, and 4 correspond to the three steps used for checkpoints in Section 16.3.6.

To recover from the loss of nonvolatile storage, the system restores the database to disk by using the most recent dump. Then, it consults the log and redoes all the actions since the most recent dump occurred. Notice that no undo operations need to be executed.

In case of a partial failure of nonvolatile storage, such as the failure of a single block or a few blocks, only those blocks need to be restored, and redo actions performed only for those blocks.

A dump of the database contents is also referred to as an **archival dump**, since we can archive the dumps and use them later to examine old states of the database. Dumps of a database and checkpointing of buffers are similar.

Most database systems also support an **SQL dump**, which writes out SQL DDL statements and SQL insert statements to a file, which can then be reexecuted to

re-create the database. Such dumps are useful when migrating data to a different instance of the database, or to a different version of the database software, since the physical locations and layout may be different in the other database instance or database software version.

The simple dump procedure described here is costly for the following two reasons. First, the entire database must be copied to stable storage, resulting in considerable data transfer. Second, since transaction processing is halted during the dump procedure, CPU cycles are wasted. **Fuzzy dump** schemes have been developed that allow transactions to be active while the dump is in progress. They are similar to fuzzy-checkpointing schemes; see the bibliographical notes for more details.

16.7 Early Lock Release and Logical Undo Operations

Any index used in processing a transaction, such as a B^+-tree, can be treated as normal data, but to increase concurrency, we can use the B^+-tree concurrency-control algorithm described in Section 15.10 to allow locks to be released early, in a non-two-phase manner. As a result of early lock release, it is possible that a value in a B^+-tree node is updated by one transaction T_1, which inserts an entry $(V1, R1)$, and subsequently by another transaction T_2, which inserts an entry $(V2, R2)$ in the same node, moving the entry $(V1, R1)$ even before T_1 completes execution.[4] At this point, we cannot undo transaction T_1 by replacing the contents of the node with the old value prior to T_1 performing its insert, since that would also undo the insert performed by T_2; transaction T_2 may still commit (or may have already committed). In this example, the only way to undo the effect of insertion of $(V1, R1)$ is to execute a corresponding delete operation.

In the rest of this section, we see how to extend the recovery algorithm of Section 16.4 to support early lock release.

16.7.1 Logical Operations

The insertion and deletion operations are examples of a class of operations that require logical undo operations since they release locks early; we call such operations **logical operations**. Such early lock release is important not only for indices, but also for operations on other system data structures that are accessed and updated very frequently; examples include data structures that track the blocks containing records of a relation, the free space in a block, and the free blocks in a database. If locks were not released early after performing operations on such data structures, transactions would tend to run serially, affecting system performance.

The theory of conflict serializability has been extended to operations, based on what operations conflict with what other operations. For example, two insert

[4]Recall that an entry consists of a key value and a record identifier, or a key value and a record in the case of the leaf level of a B^+-tree file organization.