The ARIES Recovery Algorithm

We now describe the ARIES algorithm as an example of a recovery algorithm used in database systems. It is used in many relational database-related products of IBM. ARIES uses a steal/no-force approach for writing, and it is based on three concepts: write-ahead logging, repeating history during recovery, and logging changes during undo.

The second concept, repeating history, means that ARIES will retrace all actions of the database system prior to the crash to reconstruct the database state when the crash occurred. Transactions that were uncommitted at the time of the crash (active transactions) are undone. The third concept, logging during undo, will prevent ARIES from repeating the completed undo operations if a failure occurs during recovery, which causes a restart of the recovery process. The ARIES recovery procedure consists of three main steps: analysis, REDO, and UNDO. The analysis step identifies the dirty (updated) pages in the buffers and the set of transactions active at the time of the crash. The appropriate point in the log where the REDO operation should start is also determined. The REDO phase actually reapplies updates from the log to the database. Generally, the REDO operation is applied only to committed transactions. However, this is not the case in ARIES. Certain information in the ARIES log will provide the start point for REDO, from which REDO operations are applied until the end of the log is reached. Additionally, information stored by ARIES and in the data pages will allow ARIES to determine whether the operation to be redone has actually been applied to the database and therefore does not need to be reapplied. Thus, only the necessary REDO operations are applied during recovery. Finally, during the UNDO phase, the log is scanned backward and the operations of transactions that were active at the time of the crash are undone in reverse order. The information needed for ARIES to accomplish its recovery procedure includes the log, the Transaction Table, and the Dirty Page Table. Additionally, checkpointing is used. These tables are maintained by the transaction manager and written to the log during checkpointing.

In ARIES, every log record has an associated log sequence number (LSN) that is monotonically increasing and indicates the address of the log record on disk. Each LSN corresponds to a specific change (action) of some transaction. Also, each data page will store the LSN of the latest log record corresponding to a change for that page. A log record is written for any of the following actions: updating a page (write), committing a transaction (commit), aborting a transaction (abort), undoing an update (undo), and ending a transaction (end). The need for including the first three actions in the log has been discussed, but the last two need some explanation. When an update is undone, a compensation log record is written in the log. When a transaction ends, whether by committing or aborting, an end log record is written. Common fields in all log records include the previous LSN for that transaction, the transaction ID, and the type of log record. The previous LSN is important because it links the log records (in reverse order) for each transaction. For an update (write) action, additional fields in the log record include the page ID for the page that contains the item, the length of the updated item, its offset from the beginning of the page, the before image of the item, and its after image. Besides the log, two tables are needed for efficient recovery: the Transaction Table and the Dirty Page Table, which are maintained by the transaction manager. When a crash occurs, these tables are rebuilt in the analysis phase of recovery. The Transaction Table contains an entry for each active transaction, with information such as the transaction ID, transaction status, and the LSN of the most recent log record for the transaction. The Dirty Page Table contains an entry for each dirty page in the buffer, which includes the page ID and the LSN corresponding to the earliest update to that page.

Checkpointing in ARIES consists of the following: writing a begin_checkpoint record to the log, writing an end_checkpoint record to the log, and writing the LSN of the begin_checkpoint record to a special file. This special file is accessed during recovery to locate the last checkpoint information. With the end_checkpoint record, the contents of both the Transaction Table and Dirty Page Table are appended to the end of the log. To reduce the cost, fuzzy checkpointing is used so that the DBMS can continue to execute transactions during checkpointing.
Additionally, the contents of the DBMS cache do not have to be flushed to disk during checkpoint, since the Transaction Table and Dirty Page Table—which are appended to the log on disk—contain the information needed for recovery. Note that if a crash occurs during checkpointing, the special file will refer to the previous checkpoint, which is used for recovery.

After a crash, the ARIES recovery manager takes over. Information from the last checkpoint is first accessed through the special file. The analysis phase starts at the begin_checkpoint record and proceeds to the end of the log. When the end_checkpoint record is encountered, the Transaction Table and Dirty Page Table are accessed (recall that these tables were written in the log during checkpointing). During analysis, the log records being analyzed may cause modifications to these two tables. For instance, if an end log record was encountered for a transaction $T$ in the Transaction Table, then the entry for $T$ is deleted from that table. If some other type of log record is encountered for a transaction $T'$, then an entry for $T'$ is inserted into the Transaction Table, if not already present, and the last LSN field is modified. If the log record corresponds to a change for page $P$, then an entry would be made for page $P$ (if not present in the table) and the associated LSN field would be modified. When the analysis phase is complete, the necessary information for REDO and UNDO has been compiled in the tables.

The REDO phase follows next. To reduce the amount of unnecessary work, ARIES starts redoing at a point in the log where it knows (for sure) that previous changes to dirty pages have already been applied to the database on disk. It can determine this by finding the smallest LSN, $M$, of all the dirty pages in the Dirty Page Table, which indicates the log position where ARIES needs to start the REDO phase. Any changes corresponding to an LSN $< M$, for redoable transactions, must have already been propagated to disk or already been overwritten in the buffer; otherwise, those dirty pages with that LSN would be in the buffer (and the Dirty Page Table). So, REDO starts at the log record with LSN $= M$ and scans forward to the end of the log. For each change recorded in the log, the REDO algorithm would verify whether or not the change has to be reapplied. For example, if a change recorded in the log pertains to page $P$ that is not in the Dirty Page Table, then this change is already on disk and does not need to be reapplied. Or, if a change recorded in the log (with LSN $= N$, say) pertains to page $P$ and the Dirty Page Table contains an entry for $P$ with LSN greater than $N$, then the change is already present. If neither of these two conditions hold, page $P$ is read from disk and the LSN stored on that page, LSN ($P$), is compared with $N$. If $N <$ LSN ($P$), then the change has been applied and the page does not need to be rewritten to disk. Once the REDO phase is finished, the database is in the exact state that it was in when the crash occurred. The set of active transactions—called the undo_set—has been identified in the Transaction Table during the analysis phase. Now, the UNDO phase proceeds by scanning backward from the end of the log and undoing the appropriate actions. A compensating log record is written for each action that is undone. The UNDO reads backward in the log until every action of the set of transactions in the undo_set has been undone. When this is completed, the recovery process is finished and normal processing can begin again. Consider the recovery example shown in Figure 23.5. There are three transactions: $T_1$, $T_2$, and $T_3$. $T_1$ updates page $C$, $T_2$ updates pages $B$ and $C$, and $T_3$ updates page $A$. 
For the REDO phase, the smallest LSN in the Dirty Page Table is 1. Hence the REDO will start at log record 1 and proceed with the REDO of updates. The LSNs {1, 2, 6, 7} corresponding to the updates for pages C, B, A, and C, respectively, are not less than the LSNs of those pages (as shown in the Dirty Page Table). So those data pages will be read again and the updates reapplied from the log (assuming the actual LSNs stored on those data pages are less than the corresponding log entry). At this point, the REDO phase is finished and the UNDO phase starts. From the Transaction Table (Figure 23.5(c)), UNDO is applied only to the active transaction T3. The UNDO phase starts at log entry 6 (the last update for T3) and proceeds backward in the log. The backward chain of updates for transaction T3 (only log record 6 in this example) is followed and undone.