Checkpointing in Oracle

Ashok Joshi, William Bridge, Juan Loaiza, Tirthankar Lahiri
500 Oracle Parkway, Redwood Shores, CA 94065

{ajoshi, wbridge, jloaiza, tlahiri}@us.oracle.com

Abstract

Checkpointing is an important mechanism for limiting crash recovery times. This paper describes a new checkpointing algorithm that was implemented in Oracle 8.0. This algorithm efficiently finds buffers which need to be written for checkpointing and easily scales to very large buffer cache sizes: it has been tested with buffer caches as large as six million buffers. Based on this algorithm, we have implemented a new checkpointing mechanism which we refer to as the incremental checkpointing mechanism. Incremental checkpoints are continuous, low overhead checkpoints that write buffers as a background activity. Incremental checkpointing is able to continuously advance the database checkpoint, i.e., the starting position in the redo log for crash recovery, resulting in dramatic improvements in recovery time while imposing minimal overhead during normal processing. The rate of buffer writes for incremental checkpointing can be controlled by the user to balance checkpoint writing overhead with recovery time requirements. In this paper, we describe the new data structures and algorithms that have been implemented for checkpointing and for incremental checkpointing in Oracle 8.0.

Proceedings of the 24th VLDB conference, New York, USA, 1998

1 Introduction

Most conventional database systems (including Oracle) follow the *no-force-at-commit* policy for data blocks [Haerder83] because of its significant performance benefits. The use of this policy implies that a page modified in memory may need recovery if there is a system crash. A database checkpoint is critical for ensuring quick crash recovery when the no-force-at-commit policy is employed since it limits the amount of redo log that needs to be scanned and applied during recovery.

As the amount of memory available to a database increases, it is possible to have database buffer caches as large as several million buffers. A large buffer cache imposes two requirements on checkpointing. First, it requires that the algorithms be scalable with the size of the buffer cache. Second, it requires that the database checkpoint advance frequently to limit recovery time, since infrequent checkpoints and large buffer caches can exacerbate crash-recovery times significantly.

In order to address these issues, the checkpointing algorithm has been completely rewritten in Oracle 8.0. Our objective was to make the algorithm scalable to very large buffer caches, and to facilitate frequent checkpointing for fast crash recovery. Scalability is achieved by organizing all the modified buffers on ordered queues; such queues increase the efficiency of identifying the precise set of buffers that need to be written for checkpoints. Frequent advancement of the database checkpoint is made possible by the introduction of incremental checkpointing.

The rest of this paper is organized as follows: We begin with a brief description of the Oracle-specific requirements and terminology relating to checkpointing and the Oracle processes involved in a checkpoint. The next section contains a description of the checkpoint data structures used by the Oracle 8.0 checkpointing algorithm followed by a description of the incremental checkpoint mechanism. We conclude with some observations on the benefits of the new algorithm.

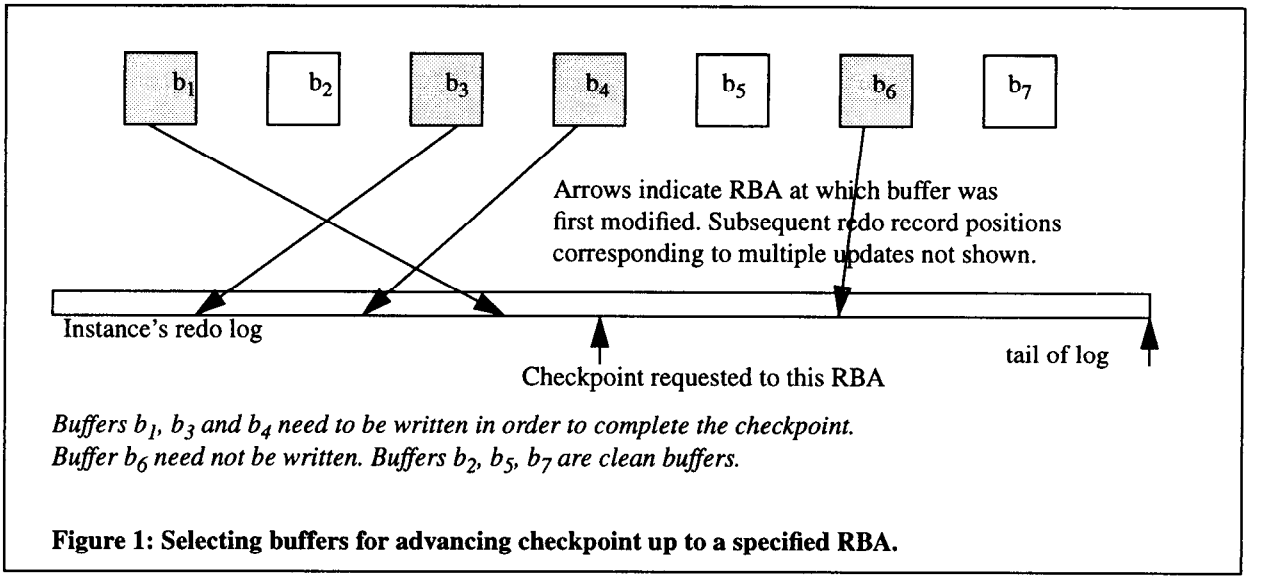


Figure 1: Selecting buffers for advancing checkpoint up to a specified RBA.

2 Oracle Checkpointing Overview

Oracle supports a shared-disk architecture; the shared-memory and group of Oracle processes that run on each node in a multi-node shared disk cluster are collectively known as an *instance* of Oracle. We briefly describe a conventional (as opposed to an incremental) checkpoint in Oracle. For the purpose of this discussion, the log may be thought of as an ever-growing file containing redo records generated by an instance. An *RBA* (redo byte address) indicates a position in the redo log. Oracle uses a set of a dedicated processes (called the *database writers* or *DBWRs*) for writing data blocks to disk. A dedicated process (called the *checkpoint process* or *CKPT*) records checkpoint information to the *Control File* which represents stable storage for maintaining bookkeeping information (such as checkpoint progress) for an Oracle database.

Each instance in Oracle has its own log. An *instance checkpoint* refers to some RBA (called the *checkpoint RBA*) within an instance's log such that all in memory buffers whose redo appears prior to this RBA have been written to disk. Hence, recovery for that instance needs to recover only those data blocks whose redo records occur between the checkpoint RBA and the end of the log.

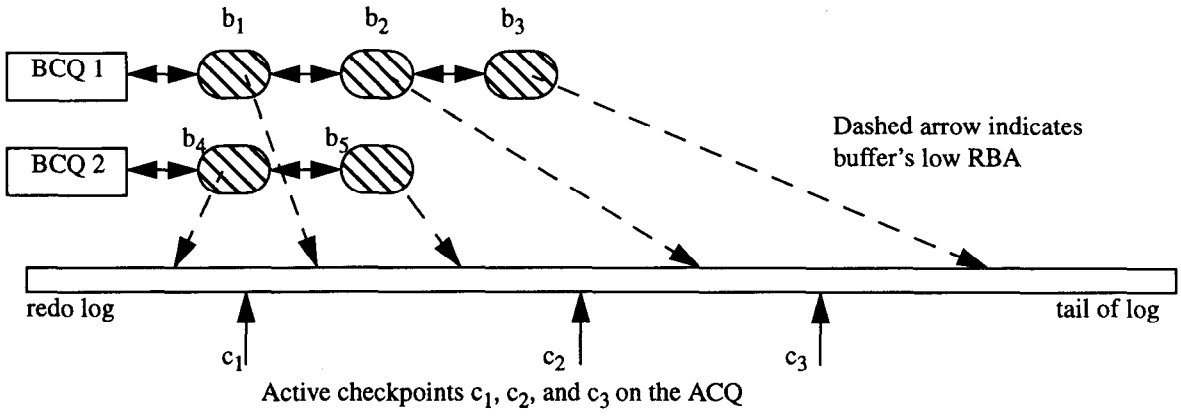
A checkpoint operation consists of three distinct phases. In the first phase, the process initiating the checkpoint "captures" the checkpoint RBA. This RBA is most often the current RBA (the RBA of the last change made to a buffer) at the time the request is initiated. In the second phase, the DBWR process writes out all required buffers, i.e., all buffers that have been modified at RBAs less than or equal to the checkpoint RBA. After all required buffers have been written, in the third phase, the CKPT process records the completion of the checkpoint in the control file. Only when the third phase has completed can we assert that the instance checkpoint has advanced to the new RBA. Normal transaction activity is not affected while a checkpoint request is active. Figure 1 provides an example of how buffers are selected to be written for checkpoint operations. [Oracle97] contains further details on the architecture of the buffer manager and the recovery subsystem in the Oracle8 Universal Server.

3 Oracle Checkpointing Data Structures and Algorithm

3.1 Buffer Checkpoint Queues (BCQ)

The most significant enhancement to the checkpoint algorithm is the introduction of new data structures that increase the efficiency of finding the buffers that need to be written for a checkpoint. The primary data structure is the *Buffer Checkpoint Queue* which contains modified buffers linked in ascending order of their *low RBA*. The low RBA is the RBA corresponding to the first (in-memory) modification of the buffer. Each buffer header contains the value of the low RBA associated with the buffer; this value is set when the buffer is first modified. Obviously, a clean buffer does not have any low RBA in its buffer header and is not linked on the checkpoint queue.

In response to a checkpoint request, buffers are written from the head of the checkpoint queue i.e., in ascending order of low RBA values. After a buffer is written, it is



Notes: DBWR always selects buffers in ascending low RBA order. After b_4 is written, c_1 is completed. After b_1 and b_5 are written, c_2 is completed. After b_2 is written, c_3 is completed.

Figure 2: Illustration of the organization of two BCQs and the ACQ with 3 entries.

unlinked from its checkpoint queue. Given a checkpoint RBA, DBWR writes buffers from the head of the queue until the low RBA of the buffer at the head of the checkpoint queue is greater than the checkpoint RBA. At this point, CKPT can record this checkpoint as having completed by updating the checkpoint progress record in the control file (phase 3).

The operations of linking and unlinking a buffer from the checkpoint queue need to performed in a critical section under a latch (a lightweight mutual exclusion primitive). In order to reduce hotspots on this critical section, it is possible to configure an Oracle instance with multiple latches, each protecting a different checkpoint queue. Each buffer is statically associated with a checkpoint queue. Having multiple checkpoint queues also improves scalability and write-throughput since it is then possible to also configure the instance with multiple DBWR processes which are responsible for writing buffers from different checkpoint queues.

3.2 Active Checkpoint Queue (ACQ)

The ACQ is a single queue that contains active checkpoint requests. Whenever a checkpoint is requested, a new *active checkpoint entry* describing the request is added to the ACQ. Each such entry on the ACQ contains the RBA up to which buffers need to be written in order to complete the checkpoint represented by the entry. The entry also contains other information specific to the checkpoint request. Note that it is possible to have several checkpoint requests active at the same time. Figure 2 illustrates two BCQs and three active checkpoints.

A checkpoint may be requested for various reasons. A user can initiate a checkpoint at any time, for instance, by issuing an `ALTER SYSTEM CHECKPOINT` command. Whenever an instance switches from one log file to another, it starts a checkpoint so that it will be possible to reuse the log file later. In addition, datafiles must be checkpointed before they can be backed up, therefore, backup operations initiate checkpoints. For all these reasons it is possible to have multiple entries on the ACQ.

4 Incremental Checkpoints

The incremental checkpoint technique uses the same data structures that are used by conventional checkpoints. It exploits the fact that the dirty buffers in the cache are linked in low RBA order. If DBWR continually writes buffers *from the head of the checkpoint queue*, the instance checkpoint (lowest low-RBA of the modified buffers) will keep advancing. Periodically, CKPT can record this lowest low-RBA to the control file (using a very lightweight control-file update protocol). This periodically recorded lowest low RBA is the current position of the incremental checkpoint for the instance. Since the incremental checkpoint is performed continuously, the value of the incremental checkpoint RBA will be much closer to the tail of the log than the RBA of a "conventional" checkpoint, thus

limiting the amount of recovery needed. When incremental checkpointing is enabled, DBWR keeps writing buffers from the checkpoint queues in ascending low RBA order in addition to performing other writing activity. In addition, the CKPT process periodically records the progress of the incremental checkpoint in the control file. By controlling the rate at which buffers are written, we can reduce the overhead for incremental checkpoint. Quite often, writing buffers in ascending low RBA order also performs LRU replacement writes and vice versa. Hence, aging writes and checkpoint writes can complement each other.

Oracle provides tuning parameters which influence the rate at which incremental checkpoint buffers are written. Note that a smaller rate will advance the incremental checkpoint slowly; a higher rate will advance the checkpoint rapidly. It should also be noted that there is not necessarily any correlation between the number of buffers written for checkpoint reasons, and the progress of the checkpoint, since the sizes of the redo records for different changes to blocks can vary considerably.

5 Performance benefits

The new checkpoint algorithm improves the performance of checkpoint operations in two ways:

First, the cost of performing a checkpoint is determined only by the number of dirty buffers in the buffer cache with RBAs below the checkpoint RBA, and not by the total number of buffers in the cache. If the instance checkpoint advances quickly (as it does when incremental checkpointing is enabled), the number of such buffers is relatively small.

Second, multiple checkpoints can be serviced by servicing one single checkpoint that *subsumes* all other checkpoints: A checkpoint is said to subsume one another checkpoint if the former checkpoint has a higher RBA than the latter. For instance, in figure 2, c_3 subsumes c_1 and c_2 in that if DBWR writes buffers in response to checkpoint c_3, the c_1 and c_2 checkpoints are automatically performed along the way, as a result. This implies that the cost of servicing a set of concurrent checkpoint requests collapses down to the cost of servicing the checkpoint with the highest RBA, instead of being equal to the sum of the costs of performing each checkpoint individually.

Since checkpoints frequently occur during normal processing, both these properties of the new algorithm improve normal runtime performance.

Incremental checkpointing significantly improves recovery performance. The performance of crash recovery is a critical (and often overlooked) component of overall system availability. We have experimented with various workloads, buffer cache sizes ranging from a few hundred to six million buffers, and various settings of the rate at which incremental checkpoint buffers are written. When incremental checkpointing is enabled, we have found dramatic reductions in recovery times. In addition, our measurements indicate that in all but the most demanding of workloads, it is possible to advance the incremental checkpoint at a very high rate without noticeable impact on run-time performance. These preliminary measurements are very encouraging; it is possible to have very fast crash recovery without compromising performance at run-time.

6 Conclusions

Checkpointing modified buffers is a critical aspect of buffer management because it reduces crash recovery times. In this paper, we have briefly described a novel algorithm for checkpointing in version 8.0 of the Oracle Universal Server. This algorithm uses queues of buffers ordered by low RBA to facilitate rapid identification of buffers to be written for checkpoint operations. We have also implemented a new type of checkpointing which we refer to as incremental checkpointing, which continuously advances the database checkpoint RBA as a lightweight background activity. The new algorithm significantly improves the performance of checkpoint operations, and incremental checkpointing greatly improves crash recovery times with negligible impact on normal activity.

7 References

[Haerder83]: Haerder, T., and Reuter, A., *Principles of Transaction-Oriented Database Recovery,* ACM Computing Surveys, 15(4) 287-317

[Oracle97]: *Oracle8 Server Concepts Volume I and II*: Oracle Corporation, Part Number A54646-01 and A54644-01,June 1997