Reload in a Main Memory Database System: MARS

Le Gruenwald, Margaret Eich

Department of Computer Science and Engineering
Southern Methodist University

Abstract

In a main memory database, the primary copy of the database may be stored in a volatile memory. When a crash occurs, a complete or partial reload of the database from archive memory to main memory must be performed. It is essential that an efficient reload scheme be used to ensure that the expectations of high performance database systems are met. This paper introduces different complete reload algorithms that aim at fast response time of transactions and high throughput of the overall system.

Introduction

Recently, with memory costs decreasing, storage capability increasing, and the need for high performance database systems rising, many researchers have been examining the problem of designing *Main Memory Database (MMDB) systems* [1,2,3,4,5]. *MARS(MAin memory Recoverable database with Stable log)* is a MMDB system designed at Southern Methodist University [6]. It assumes that the primary copy of the database is in main memory (MM) and that an *archive database* existing on secondary storage (AM) is used solely as a backup in the event of main memory media failure or system failure. The log is assumed to exist both on disk and in a stable memory buffer. All updates take place in a stable memory which acts as a shadow memory. At commit time, after image records (AFIM) are copied from the shadow memory to MM. A database processor (DP) is used to handle normal database processing, while a recovery processor (RP) is used to perform recovery processing activities.

In a MMDB system, such as MARS, the primary copy of the database may be stored in a *volatile* main memory. When a crash occurs because of system failure or main memory media failure, a complete or partial reload is needed to load the database from archive memory into main memory. Even though this reload process does not take place frequently, it could become a major bottleneck during processing. Performance impediments are mainly caused by two problems: down time and page faults. While the system is down, no transactions can be processed. According to [7], there are many applications that could use 1000 transactions per second. If our reload process takes one hour to complete, then in this environment there could be 3,600,000 transactions backlogged if the entire database must be reloaded before bringing the system up. This figure is intolerable in a high performance system. If we choose to bring the system online before the reload process completes, we then will encounter another serious problem: page faults caused by referencing data not yet loaded.

Our objective is to develop a reload scheme that aims at fast response time of transactions and high throughput of the overall system. In this paper we introduce two reload algorithms and analyze their advantages and disvantages. These algorithms are developed based on the MARS model. Note that at present, there is no reload scheme that is in use on MARS.

Below is the list of terminologies and their meanings that will be involved in the discussion of the algo-

† This material is based in part upon work supported by the Texas Advanced Research Program (Advanced Technology Program) under Grant No. 2265.

rithms: *Reload Preemption*: reload of some data is suspended and replaced by reload of some other data; *Reload Granularity*: the smallest unit of data to be reloaded. During the reload of this unit, no reload preemption is allowed; *Reload Prioritization*: priority of data to be reloaded. This indicates which data will be reloaded first and which data will be reloaded next; *Reload Threshold*: a variable specifies the amount of the database that must be memory-resident before the system can be brought online.

Ordered Reload with Prioritization Algorithm

This algorithm does not use access frequency; but does consider reload prioritization and preemption. Its goal is to reload data that is needed immediately before other data so that the system can be brought up before the entire database is reloaded and transaction response time can be reduced. The algorithm consists of the following steps:

- Step 1: identify waiting transactions and their needed pages. Group these pages according to cylinders. Waiting transactions include not-yet-committed transactions and backlogged transactions when the system was down.
- Step 2: reload system pages into MM on a cylinder basis. This means that cylinder 1's of all disks are reloaded in parallel (system pages are assumed to be stored on cylinder 1's).
- Step 3: reload the rest of the database based on the following prioritization until the reload threshold is reached: (1) Priority 1 (highest): reload pages needed by waiting transactions on a cylinder basis; and (2) Priority 2: reload the rest of the cylinders on all disks according to the order they are stored on disks, starting from the block pointed by the head of each disk.
- Step 4 (performed by the database processor (DP) in parallel with step 3 which is performed by the recovery processor (RP): copy to the shadow memory all AFIM records of committed transactions which are recorded on the log from the second to last begin checkpoint. All these AFIM records are associated with a special recovery transaction.
- Step 5: bring the system up when the reload threshold is reached.
- Step 6: reload the rest of the database based on the following prioritization until the entire database is in MM: (1) Priority 1 (highest priority): reload pages needed by executing transactions on a demand basis. Executing transactions are those which arrive after the system resumes its execution. A *cylinder* is chosen to be our reload granularity, therefore the reload will not take place until the cylinder which is being reloaded is completely memory resident; (2) Priority 2: same as priority 1 in step 3; and (3) Priority 3: same as priority 3 in step 3. Note that in this step, reload preemption is in effect: the reload of a lower priority is preempted by the reload of a higher priority to ensure that data that is needed immediately is brought into MM before other data.
- Step 7: commit the special recovery transaction to copy all AFIM records mentioned in step 4 to MM.

Following are the tradeoffs of this algorithm: *Advantages*: (1) only a portion of the database needs to be reloaded before the system can resume its execution, (2) reload prioritization and reload preemption are taken into account which allow executing transactions to be given immediate attention. This might lead to an improvement in system throughput and transaction response time, and (3) the reponse time of waiting transactions is reduced since data needed by them are reloaded before data that is not needed by any transactions; *Disadvantages*: (1) more complicated implementation is required to control the reload prioritization and reload preemption, and (2) identification of waiting transactions and their needed pages adds an overhead to the reload process.

Smart Reload Algorithm

This algorithm considers prioritization, preemption, and access frequency. Its purpose is not only to reload data that is needed immediately before other data but also to reload data that is accessed more frequently before data that is accessed less frequently. Its motivation is to take the advantages of *hot spots* which have been demonstrated to exist in some database applications ([8,9]). A *hot set (or hot spot)* is a subset of the database that is frequently accessed. Examples of hot sets are a data dictionary, file indices, summary data (company totals), extremely active accounts, and bank balances in debit/credit transactions [10].

Reloading the hot sets first should reduce the number of page faults, and thus reduce response time of subsequent transactions. This leads us to the development of a priority reload algorithm that makes use of frequency access to predict future reference. This algorithm does not attempt to predict what will be referenced next, instead it relies on the hot set concept to guarantee that the data reloaded is that with the highest probability of being referenced among data in the archive memory.

In this algorithm, the access frequency of each page (block) is assumed to be available. At any point in

time, except for the case of demand reload, the most frequently accessed page is searched and reloaded into MM. Blocks are not organized on AM in either increasing or decreasing order of access frequency. Therefore, we choose a block to be the reload granularity. Besides these differences, The rest of the algorithm is exactly the same as the *ordered reload with prioritization* algorithm.

This algorithm has all advantages and disadvantages of the *ordered reload with prioritization* algorithm plus the following: *Advantages*: (1) since only one block instead of an entire cylinder needs to be reloaded in order for a reload preemption to take place, the waiting time of executing transactions might be shorter; and (2) the more frequently accessed pages are reloaded before the less frequently accessed pages except for the case of demand reload. This might reduce the number of page faults, which in turn will lead to a higher system throughput; *Disadvantages*: (1) Extra overhead is incurred due to frequency count calculation; (2) more stable memory is needed to store the access frequency information. (3) searching for the highest frequency page among all the pages to reload imposes an extra overhead on the reload process; and (4) the block reload granularity requires more seek time and latency time than the one incurred in the cylinder reload granularity. This might lead to a longer total reload time.

Summary and Conclusions

In this paper, we introduced two different algorithms to perform a complete reload of data from AM into MM when a system crash occurs: *ordered reload with prioritization*, and *smart reload*. The first algorithm uses a cylinder as its reload granularity and does not take the access frequency into consideration. The second algorithm uses a block as its reload granularity and makes use of access frequency. Both algorithms allow the system to be brought online before the entire database is reloaded and implement the same priority reload scheme: the highest priority is given to data needed by executing transactions, the second highest priority to data needed by waiting transactions, and the last priority to the remaining data. Reload of data of lower priority is preempted by reload of data of higher priority to achieve faster system response time.

In the next phase of this research, we intend to conduct a simulation on MARS using the Winconsin benchmark to find out which algorithm is the best in terms of total reload time, system throughput, and transaction response time. The results of this simulation will be reported in the literature.

References

[1] Ammann, A., Harahan, M., Krishnamurthy, R. "Design of a Memory Resisent DBMS",*Proceedings of the IEEE Spring Computer Conference*, 1985, pp. 54-57.

[2] Hagmann, R. "A Crash Recovery Scheme for a Memory Resident Database System", *IEEE Transactions on Computers*, Vol. C-35, No. 9, Sept 1986, pp. 839-843.

[3] DeWitt, D., Katz, R., Olken, F., Shapiro, L., Stonebraker, M., Wood, D. "Implementation Techniques for Main Memory Database Systems", *Proceedings of the 1984 ACM SIGMOD Conference*, June 1984, pp. 1-8.

[4] Garcia-Monila, H., Lipton, R., Honeyman, P., "A Massive memory database system", Princeton University, Department of Electrical Engineering and Computer Science, Technical Report, Sept 1983.

[5] Lehman, T., Carey, M. "A Recovery Algorithm for a High- Performance Memory Resident Database System", *Proceedings of the 1987 ACM SIGMOD*, 1987.

[6] Eich, M., "MARS: The Design of a Main Memory Database Machine", *Proceedings of the International Workshop on Database Machines*, ICOT, Oct. 1987, pp. 468-481.

[7] Gray, J., Good., B., Gawlick, D., Homan, P., Sammer, H. "One Thousand Transactions Per Second", Tandem Computers, TR 85.1, Nov. 1984.

[8] Chou, H., DeWitt, D., "An Evaluation of Buffer Management Strategies for Relational Database Systems", *Proceedings of the International Conference on Very Large Data Bases*, 1985, pp. 127-141.

[9] Sacco, M., Schkolnick, M. "Buffer Management in Relational Database Systems", *ACM Transactions on Database Systems*, Vol. 11, No. 4, Dec. 1986, pp. 473-498.

[10] Gawlick, D. "Processing Hot Spots in High Performance Systems", *Proceedings of the IEEE Spring Computer Conference*, 1985, pp. 249-251.