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 exists 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 disadvantages. 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.
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.

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 response 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 Wisconsin 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 simu-

lation will be reported in the literature.

References


