Batch Transaction Scheduling on Multi Disk Database Computers

7Q-1

OHMORI, Tadashi and TANAKA, Hidehiko
Information Engineering Course, Univ. of Tokyo

1 Introduction

In database applications, batch transactions access as large bulk data as hundreds of mega byte to database-disks.

This paper points out a 'convoy' of batches occurs when they run concurrently. The 'convoy' prevents multiple disks from running in parallel. In order to avoid it, we propose a scheduler using 'Weighted Transaction Precedence Graph (WTPG)'.

2 Environment

Our target environment is a multi disk-module database computer; It is composed of multiple database-disk modules interconnected by a network.

As for data placement on multiple disks, "range partitioning" in [2] is assumed. Each relation is divided into partitions by range of values on a specified attribute. Each disk module (DM) stores one partition per relation.

A batch is modeled as a serial sequence of a read/write step to a partition on a DM.

As a concurrency controller, we use cautious (strict) two phase lock protocol (C2PL) in [1]. C2PL is a cautious scheduler where each transaction obeys strict two phase lock protocol. Thus a read/write step is a shared(S) / exclusive(X) lock request. All locks are held until commitment and released after deferred updates. Lock mode elevation is allowed. Lock granule is a partition. C2PL is a deadlock free protocol by declaring access-dataset of transactions.

Each partition is given a cost in proportion to its size of data. The unit of cost is a unit of bulk data such as cylinder of a disk. e.g. if a partition has 4MB and one cylinder is 1MB, its cost is 4. Read/Write steps to a partition is also given the cost of the partition.

Batch:
T1 : r1(D).
T2 : r2(A) → r2(E) → w2(A).
T3 : r3(C) → w3(A,C).
T4 : w4(C) → w4(F).

data-placement of partitions :
DM1 --- E, D DM2 --- F, A, C.

cost of partition :
A, C : 1, E, F : 3, D : 4.

Figure 1: batch model

3 Convoy of batch

Fig.1 illustrates four batch transactions T1 to T4, data placement on two disk modules DM1, DM2, and its cost.

Suppose that they are ordered in the sequence T1, T2, T3, T4 in the ready queue (RQ) of the concurrency controller (CC). CC runs as follows;

When one of the DMs gets idle, CC selects one of the transactions, T, in RQ using a service discipline such as FIFO. T must have a ready step to run on the idle DM. Then CC runs T's step on the DM if its lock is granted. After the step is completed, T is queued into RQ again. []

Note that, in the above scenario, CC does not process lock-request until one of the DMs gets idle.

e.g. Fig.2-a is a Gantt chart where CC schedules the batches in Fig.1 by FIFO service. FIFO makes a serializable order (SR-order) T1, T2 → T3 → T4 as follows; (T2 → T3 says 'T2 precedes T3'.)

Until clock (clk) 2, CC using FIFO service discipline has started r1(D) on DM1, and has completed r2(A) and r3(C) on DM2. At clk 2, DM2 gets idle. Then all ready steps for DM2 are blocked until T2 commits; w3(A) is blocked by r2(A), elevated to X-lock. w4(C) is blocked by r3(C). After T2 commits at clk 8, T4 is blocked again until T3 commits. []

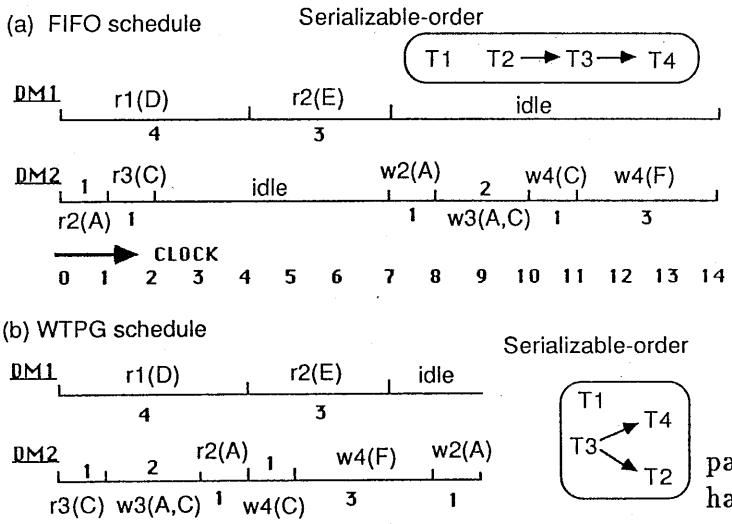

Figure 2: Gantt chart of the batches in Fig.1

In Fig.2-a, DM2 is idle until T2 commits. After clk 7, DM1 is idle in turn. Consequently two DMs don't run in parallel. It is because FIFO makes a long chain of transactions T2 → T3 → T4. Furthermore `r1(D)` makes DM1 busy and defers commitment of T2.

We refer to the situation as *convoy of batches*.

A solution to the convoy is to make many batches active and run no-conflicting ones on idle disks. However the number of active batches is small because they require much resource such as large main memory.

Another solution is to predict the possible convoy among a few active batches and avoid it.

Fig.2-b is the case where CC selects T3 at clk 0 on DM2. The SR-order is T1, T3 → T2, T3 → T4. Both DMs can run in parallel.

Thus CC must determine SR-order of batches so that all DMs can run in parallel.

4 Scheduler using WTPG

We propose a cautious scheduler which predicts and avoids 'convoy' of batches. For the prediction, we use Weighted Transaction Precedence Graph (WTPG); transaction precedence graph with weighted edges, defined in [3].

Fig.3-a is a WTPG of Fig.2-a, at clk0 after T1 has started. T0 is the initial transaction. The final transaction Tf and its edges are omitted in Fig.3.

In the WTPG, the edge T0-Ti has a weight as Ti's earliest possible commitment time. e.g. T2 can commit at clk 8 at earliest.

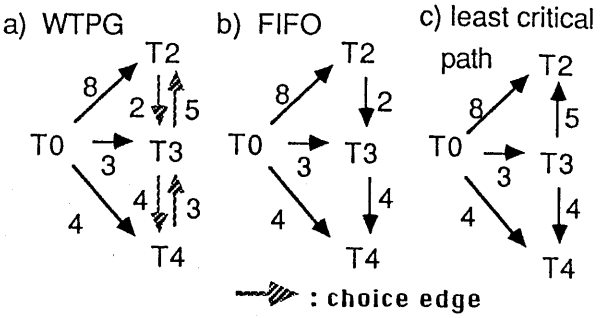

Figure 3: WTPG of Fig.2-a at clock 0

A pair of conflicting transaction Ti and Tj has a pair of edges named *choice edge*. The edge Ti-Tj has a weight as an incremental cost from commitment of Ti to that of Tj. So does the reverse edge Tj-Ti. When Ti is determined to precede Tj, the edge Tj-Ti disappears. The operation is named *resolution of choice edge*. e.g. it takes the cost of `w3(A,C)` = 2 until T3 commits after T2 commits.

The edge Ti-Tf has a weight as a cost from Ti's commitment to its completion. The weight is negligible in C2PL and omitted in Fig.3.

A full schedule gives a SR-order which resolutes all choice-edges in a WTPG. The resoluted WTPG should be acyclic. In the WTPG, the earliest possible completion time of the schedule is the length of the critical path from T0 to Tf.

Fig.3-b is the WTPG resoluted by FIFO. Its critical path is T0-T2-T3-T4 with the length 14. The SR-order 'T3 → T2, T3 → T4' makes the least critical path T0-T3-T2 with the length 8 in Fig.3-c.

Using WTPG, CC predicts 'convoy' and runs as follows;

when a DM gets idle, CC determines the SR-order with the least critical path in the current WTPG. Then, among ready transactions under the order, it selects one to run whose ready step has the smallest processing time. []

The latter part obeys Small Processing Time priority in job-shop scheduling. Fig.2-b is a schedule generated by the above strategy.

It is NP-complete to decide the SR-order with the least critical path in a given WTPG. Its restricted problems are, however, solved in linear time of the number of nodes [3].

[references]

[1]. Nishio, et al, IWDM87, pp.212-225. [2] Dewitt, et al, VLDB86,pp.228-237. [3] Ohmori, et al, to appear in Data Eng., Tech. Rep., IEICE, July,88.