DE88－19

並列ディスクデータベースマシンにおける バッチトランザクションのスケジュール方式

Batch Transaction Scheduling on Multi Disk Database Machines

大森 匡　　田中 英彦
OHMORI, Tadashi　　TANAKA, Hidehiko
東京大学 工学部 情報工学専攻
Information Engineering Course, University of Tokyo

Abstract

This paper describes 'convoy' of batch transactions occurs when database machines (DBM) run them concurrently. The convoy prevents multiple disks in DBM from running in parallel. In order to predict and avoid the convoy, Weighted Transaction Precedence Graph and a cautious scheduler using it are proposed. The scheduler generates a serializable schedule so that all disks run in parallel.

1 Introduction

The emerging 24 hour database services require high throughput of on-line transactions. At the same time, they need to compute more batch jobs in a much shorter time. A batch in database applications is a transaction which reads or writes as large bulk data as hundreds of mega byte to database-disks.

For instance, banking systems run bulk update jobs to compute interests of customers. Statistic jobs such as aggregations issue relational algebra operations and read bulk data which satisfy selection-conditions.

The above situations need to execute batches concurrently and fast on database machines (DBM), dedicated for bulk data processing. Previous works on batch processing are, however, discussed in on-line transaction environment [Bay86,MS87]. Our objective is to develop a concurrency controller for batches on DBMs.

Most of current DBMs are composed of multiple disk modules. When batches run concurrently on such a multi-disk DBM, multiple disks don't always run in parallel.

Suppose the following situation.

Batch T1, T2, T3 are active and no other batches exist now. T1, T2, T3 read bulk data on a disk D1, D2, D3 respectively. T1 and T2 update a common record a at first before issuing bulk data access. T2 and T3 update another common record b before their bulk accesses as well. All transactions defer updates until they commit. Then, if T1 and T2 are granted to update a and b at first respectively, we must obey a serializable order T1 $\rightarrow$ T2 $\rightarrow$ T3. Thus T2 cannot issue bulk access to D2 before T1 commits. T3 cannot do so to D3 before T2 commits, either. Consequently three disks D1-D3 don't execute bulk data processing in parallel.

The above phenomenon arises because batches must keep serializability and they access bulk data to different disks.

We refer to the situation as *convoy of batches*. The convoy degrades batch transaction throughput of multi disk DBM signifi-

cantly.

If a serializable order is T1 → T2 and T3 → T2 in the above example, T1 and T3 execute their bulk data processings in parallel on D1 and D3. Hence a concurrency controller should generate the serializable order of batches so that all DMs can run bulk data processing in parallel. It is the problem to be solved in this paper.

The rest of the paper is organized as follows; Section 2 gives our assumptions and illustrates 'convoy' of batches. In Section 3, we propose "Weighted Transaction Precedence Graph (WTPG)". It is a transaction precedence graph with weighted edges. Its weight represents the execution cost of batches. In WTPG, the degree of 'convoy' is measured by the length of critical path from the initial transaction to the final one.

In order to predict and avoid the convoy, the scheduler decides a serializable order so that the critical path get the least in a given WTPG. This decision problem is NP-complete. Section 4 presents a linear time strategic algorithm for the restricted class of the problem. Finally extensions of our work are discussed in Section 5.

2 Batch processing

2.1 Target environment

Our target environment is a multi disk database machines. It is composed of multiple disk modules, interconnected by a network [DeW86, Dew87b]. Each disk module (DM) is a small computer with a database disk. It executes basic data processing such as filtering.

All relations are stored by "range partitioning" on multiple DMs [DeW86]. It divides a relation into partitions by range of values on a specified attribute. e.g. a relation `emp(Id,Name)` is divided into three partitions by three ranges on `Id`; $Id < 10$, $10 <= Id < 20$, $20 =< Id$. Each DM stores one partition per relation, and makes a clustered index on any one attribute.

Note that sorted files or VSAM-like clustered files are naturally stored by "range partitioning". [DeW87a] shows 'range partitioning' is superior to 'hash partitioning' of Teradata in performance.

By range partitioning, a selection-predicate on the partitioned attribute causes bulk data access to a small number of DMs in all DMs.

2.2 Batch model

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

As a concurrency controller (CC) , we use cautious (strict) two phase lock protocol (C2PL) in [Nis87]. 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. Lock mode elevation is allowed. Lock granule is a partition.

In C2PL each transaction must declare all read/write data set of it at its first step. By the declaration, C2PL predicts the doomed conflicts of transactions and avoids deadlock.

C2PL grants a lock-request to a data item d iff it is not conflicted with current locks on d, AND the transaction precedence graph does not get cyclic when the lock-request is granted.

In order to depict a state of batch scheduling, we define a cost of batches and use a gantt chart.

Each partition is given a cost in proportion to its size of data. The unit of size is a unit of bulk data such as cylinder of a disk. e.g. if the partition D on DM1 has 4MB and one cylinder is 1MB, D is given a cost 4. Read/Write steps to a partition are also given the cost of the partition. For simplicity, this paper assumes each read/write step is executed without interruption.

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

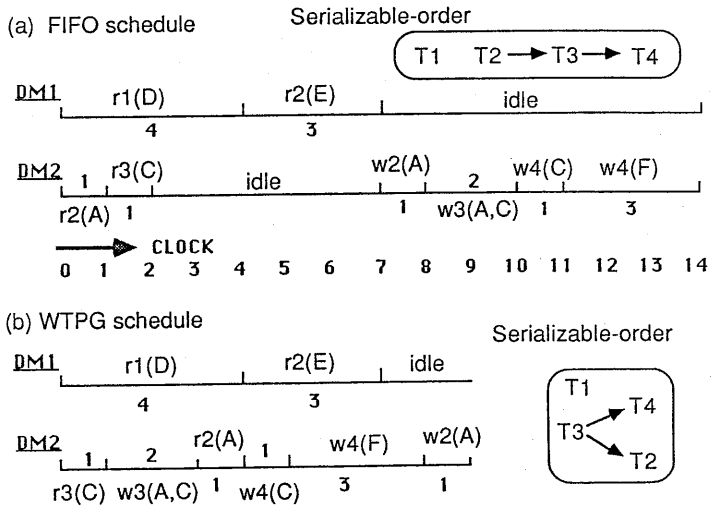

Figure 2: Gantt chart of the batches in Fig.1

2.3 Convoy of batch

This section illustrates '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 in a serializable order'.)

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. []

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. The chain makes DM1 idle after clk 7. Furthermore `r1(D)` makes DM1 busy and defers commitment of T2. This busy disk is named 'hot disk'. DM2 is idle till clk 7 by the delayed commitment on the hot disk DM1.

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 must be small because they require much resource such as large main memory for join or sort.

Another one is to distribute all files randomly to multiple DMs. The method cannot, however, store sorted files or VSAM-clustered files. Since these file-organizations are inherent in high throughput on-line transaction processing, we assume range partition and allow for 'hot disk'.

We take a solution s.t. the scheduler predicts the possible convoy among a few active batches and avoids 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 run bulk data processing in parallel.

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

In the theory of scheduling, our problem is formally a dynamic scheduling of job-shop, extended by serializability constraint. Static job-shop scheduling is known to contain a NP-hard problem [Gra79]. Moreover good approximate algorithms are not found. Hence we must use a strategic scheduling.

It is our scheduling strategy to predict and avoid 'convoy' of batch.

3 Scheduler by WTPG

3.1 Scheduling strategy

The convoy is caused by either a long chain of transactions in SR-order or delay of commitment by 'hot' disk. Thus our scheduling strategy must avoid those phenomena.

The key points are to decide the priorities of lock service and disk service. Lock service priority decides which should be granted its lock request among conflicting transactions. Plural transactions may be ready under the decided SR-order. Disk service priority decides which should run on an idle disk among them.

The inherent problem is to decide lock service priority. The disk service priority is the same as those proposed in dynamic job-shop scheduling.

Suppose that two conflicting transactions Ta and Tb can commit at clock 1 and at clock 3 at earliest respectively. Tb (or Ta) must execute a step of cost 2 (or 4) after Ta (or Tb) commits and releases the conflicting lock. Ta and Tb can complete as soon as they commit. Then which transactions should precede the other in lock service?

We prepare a transaction precedence graph with weighted edges. The weight represents cost of batches as follows in Fig.3.; (T0/Tf is the initial/final transaction. 'Ta - Tb' refers to "a directed edge from Ta to Tb").

The edge T0 - Ta has a weight 1 for Ta's earliest possible commitment time. T0-Tb has a weight 3 as well. Ta-Tb has the weight 2 as the incremental cost from Ta's commitment to that of Tb. Tb-Ta has the weight 4 as well. Ta-Tf and Tb-Tf have a weight from the commitment of Ta/Tb to its completion.

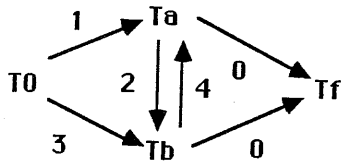

Figure 3: example of WTPG

Then *we determine the lock service priority so that the critical path from T0 to Tf get the least in the WTPG.* The critical path is just the earliest possible completion time of a total schedule of Ta and Tb.

In the example, SR-order Ta → Tb makes the critical path T0-Ta-Tb-Tf with length 3. Tb → Ta makes the path T0-Tb-Ta-Tf with length 7. Thus the scheduler grants Ta's lock request rather than that of Tb.

This lock service strategy is aimed at decreasing the earliest possible completion time of total serializable schedule.

The strategy predicts convoy as follows; A long chain of transactions makes the critical path long in the WTPG. Hot disk makes the earliest possible commitment time of some transactions long.

3.2 WTPG

This section defines WTPG formally.

Definition 1 Weighted Transaction Precedence Graph (WTPG) is a transaction precedence graph $< N, E, C, w >$ as follows; (T0 is the initial transaction. Tf is the final transaction. Ti,Tj are any other transactions.)

1. N is a set of nodes. A node is a transaction.

2. E is a set of directed edges Ti-Tj. It expresses "Ti precedes Tj" in the serializable

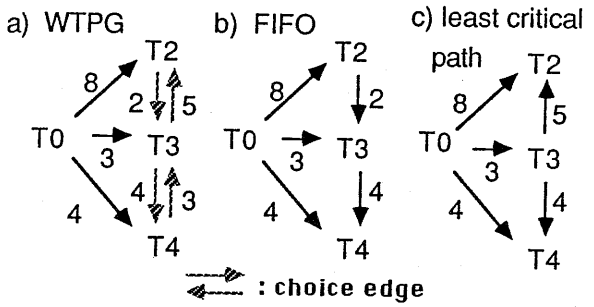

Figure 4: WTPG of Fig.2-a at clock 0

order". For each transaction Ti, T0 precedes Ti and Ti precedes Tf.

3. C is a set of *choice edges*. A *choice edge* (Ti,Tj) is a pair of directed edges Ti-Tj and Tj-Ti. It expresses Ti conflicts with Tj. If Ti is determined to precede Tj, the edge Tj-Ti disappears and the edge Ti-Tj is regarded as a member of E. This operation is named *resolution of choice edge*.

4. w is a weight function of edge Ti-Tj. For each transaction Ti, an edge T0-Ti has a weight as Ti's earliest possible commitment time from the current. An edge Ti-Tf has a weight for a cost from Ti's commitment to Ti's completion time.

Each directed edge Ti-Tj has a weight as an incremental cost from commitment of Ti to that of Tj. []

e.g. Fig.4-a is a WTPG of Fig.2-a, at clk 0 after T1 has started. T1, Tf and their edges are omitted in Fig.4. The weights on Ti-Tf is negligible in C2PL and are regarded as zero.

In the WTPG, the edge T0-T2 has a weight 8 because T2 can commit at clk 8 at earliest.

The edge T2-T3 has a weight 2 for the cost of `w3(A,C)`. T3 must run the step `w3(A,C)` before its commitment after T2 commits and releases X-lock on A.

3.3 Scheduler using WTPG

A SR-order of all transactions in a given WTPG resolutes all choice-edges in a WTPG. The order is named *a full SR-order* in the WTPG. The resoluted WTPG must be acyclic.

As shown in Fig.2-a, FIFO lock service makes a full SR-order { T1, T2 → T3 → T4}. Fig.4-b is the WTPG resoluted by this SR-order. Its critical path is T0-T2-T3-T4 with length 14. A chain of transaction T2-T3-T4 makes a long critical path.

The SR-order 'T3 → T2, T3 → T4' makes the least critical path T0-T3-T2 in Fig.4-c. Its length is 8.

Our lock service priority is to grant a lock request so that the critical path get the least in the current WTPG.

Based on this principle, CC runs as follows;

Step1. Wait until a DM gets idle. If a DM gets idle, goto step2.

Step2. If there are ready steps of transactions which determines no new serializable order, regard them as candidates to run and goto step4. If not, goto step3.

Step3. determine the full SR-order which has the least critical path in the current WTPG. Then, regard the ready transactions under the SR-order as candidates to run. goto step4.

Step4. Among candidates, select the transaction to run whose ready step has the smallest processing time. Run the selected step on the idle DM, and update parameters in the WTPG. goto step1. []

Step2 gives disk service priority to transactions which are determined to precede others. e.g. Suppose the following. T1 precedes T2. T2 is blocked. T3 has ready step which conflicts with T2. If T3 runs, a new SR-order T3 → T2 is decided. Then Step2 selects T1 rather than T3.

Step4 uses Small Processing Time (SPT) priority for disk service [Gra79]. Step4 updates the parameters on the edges T0-Ti. Other parameters are not changed as a schedule proceeds.

In Step3, ready transactions may not exist. It is the case where the determined SR-order

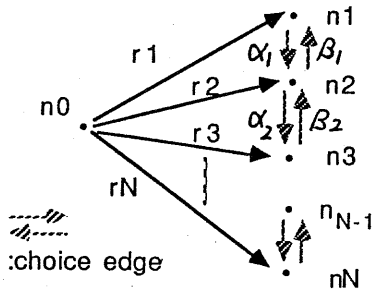

Figure 5: chain WTPG G_1

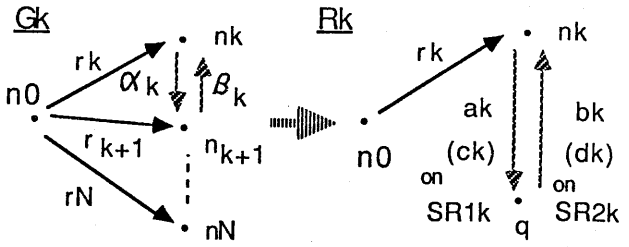

Figure 6: G_k and R_k

has given a lock-priority to transactions which don't get ready yet. This case inserts idle time to DM.

Strictly speaking, CC runs those steps when a new event occurs. The event is 'a DM gets idle' or 'a new transaction arrives'.

Fig.2-b is a schedule generated by the above strategy.

In Fig.2-b, at clk 0 after T1 starts, `r3(C)` of T3 is selected to run by the least critical path in Fig.4-c. At clk 3, `r2(A)` or `w4(C)` may run because they don't conflict with each other. If `w4(C)` is selected, `r2(A)` run at the next clock 4 rather than `w4(F)` by SPT rule.

It is NP-complete to decide the SR-order with the least critical path in a given WTPG. We use a strategic algorithm.

4 Chain WTPG

The problem is to decide the full SR-order in a given WTPG that makes the critical path from T0 to Tf the least. (In this section, "path" refers to a path from T0 to Tf. WTPG is depicted with omitting Tf.) The problem is referred to as Least Critical Path problem (LCP). LCP for a given WTPG is NP-complete, because it is reduced polynomially from Hamilton circuit problem.

This section restricts the topology of WTPG to a chain form and presents a strategic algorithm for LCP in a "chain WTPG". 'Chain' means that all transactions except T0, Tf conflict in a chain topology.

Fig.5 is a chain WTPG with node n0, n1,.., n_N. n0 corresponds to the initial transaction T0. Other nodes do to general transactions. Tf is omitted and the weights on Ti-Tf is regard as zero. r_k, α_k, β_k (k=1,..,N) are weights.

Let this graph be G1. Let G_k be a subgraph of G1 with nodes n_k,...,n_N in Fig.6. A reduced graph R_k of G_k is defined. It represents a critical path structure in G_k as follows;

Definition 2 R_k has nodes n_k and q, and the *reduced choice edge* (n_k, q) as shown in Fig.6.

Let $S1_k$ (or $S2_k$) be a set of full SR-orders on G_k such that they resolutes the choice edge (n_k, n_{k+1}) to n_k−n_{k+1} (or n_{k+1}−n_k).

The parameters $SR1_k, SR2_k$, c_k,a_k, and d_k,b_k are defined below.

- $SR1_k$ (or $SR2_k$) is a full SR-order that makes the critical path of G_k the least among those in $S1_k$ (or $S2_k$).
- In the graph G_k resoluted by $SR1_k$, c_k is the length of the least critical path. a_k is the length of a path which includes the edge n_k−n_{k+1} in the resoluted graph.
- In the graph G_k resoluted by $SR2_k$, d_k is the length of the least critical path. b_k is the length of a path which includes the edge n_{k+1}−n_k in the resoluted graph. □

By reducing G1 to R1, the least critical path in G1 is the minimum of c1 and d1. The choice edge (n1,n2) is resoluted to the edge of the selected minimum value. The SR-order with the least critical path is $SR1_k$ (or $SR2_k$) such that k=1 if the selected edge is n1-n2 (or n2-n1).

```
reduce( Gk, Rk )
Begin
   if k=N then make R_N such that
        a_N = b_N = c_N = d_N := r_N
   else
        reduce( G_{k+1} , R_{k+1} );
        R_k  := generate( R_{k+1} );
   endif
End
```

Figure 7: procedure reduce

A procedure reduce(G_k, R_k) in Fig.7 reduces G_k to R_k strategically. It uses reduce(G_{k+1}, R_{k+1}) and the procedure generate.

For the algorithm, the following lemma is used.

lemma 1 For a given chain WTPG, $r_k+\alpha_k \geq r_{k+1}$ and $r_{k+1}+\beta_k \geq r_k$.

generate makes R_k from R_{k+1} and the choice edge (n_k, n_{k+1}) as follows:

By adding a new choice edge (n_k, n_{k+1}) to R_{k+1}, generate compares the longest path in R_k including n_k with the previous critical path in R_{k+1}, and decides a new critical path in R_k.

When deciding a_k and c_k, two cases are examined where the reduced choice edge (n_{k+1},q) is resoluted to n_{k+1} - q or q - n_{k+1}.

By lemma 1, a_k and c_k are computed as shown in Fig.8- case1, case2. c_k is determined to be the minimum of ck1 and ck2 in Fig.8-case1 and case2. a_k and $SR1_k$ are those in the case where c_k is selected.

For instance, if ck1 is smaller than ck2, c_k and a_k are selected to be ck1 and ak1 respectively. $SR1_k$ is the SR-order that appends $n_k \rightarrow n_{k+1}$ to $SR1_{k+1}$.

b_k and d_k are computed from those in Fig.8-case3 and case4 in the same way.

In this way, generate examines only four possible pairs of resolution on choice edges (n_k, n_{k+1}) and (n_{k+1}, q) in R_{k+1}. It is O(1). □

Note that the procedure generate don't compute a exact solution in the problem. It is a strategic algorithm.

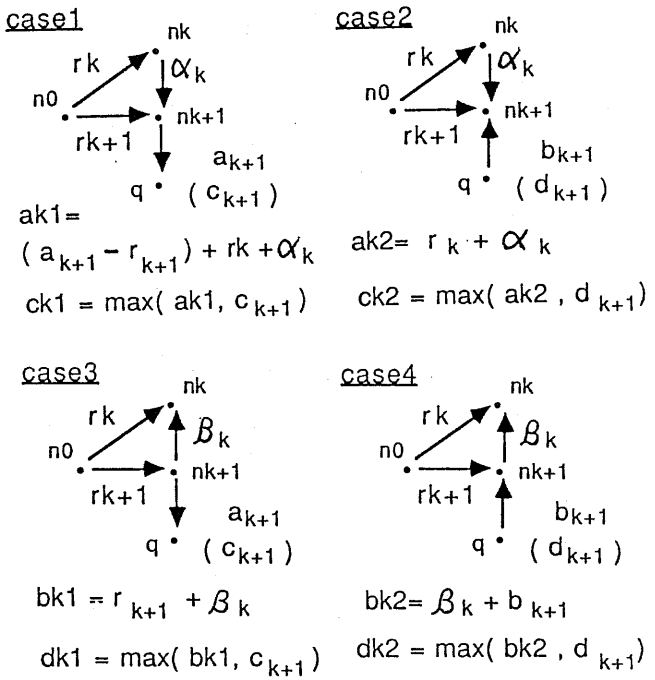

Figure 8: procedure generate

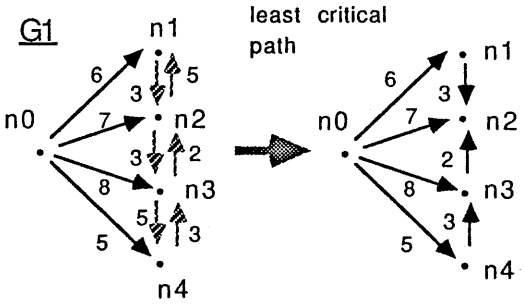

Figure 9: example of reduce

Theorem 1 The procedure reduce(G1,R1) is O(N), s.t. N is the number of nodes in G1.

Example: Fig.9 depicts a chain WTPG G1 and its least critical path resolution. All reduced graph R3 to R1 are in Fig.10. In Fig.10, R2 is generated from R3 as follows; In the case s.t. (n2,n3) is resoluted to n2-n3, a path n0-n2-n3-q with length 15 is critical path when (n3,q) is resoluted to n3-q. A path n0-n2-n3 with length 10 is a critical path when (n3,q) is resoluted to q-n3. Thus c2 in R2 is minimum(15,10) = 10. $SR1_k$ s.t. k=2 is { $n4 \rightarrow n3$, $n2 \rightarrow n3$ }. b2,d2 are computed in the same way. In R_1, (n1,q) is resoluted to n1-q with the least critical path 10. The SR-order with the least critical path is { n1

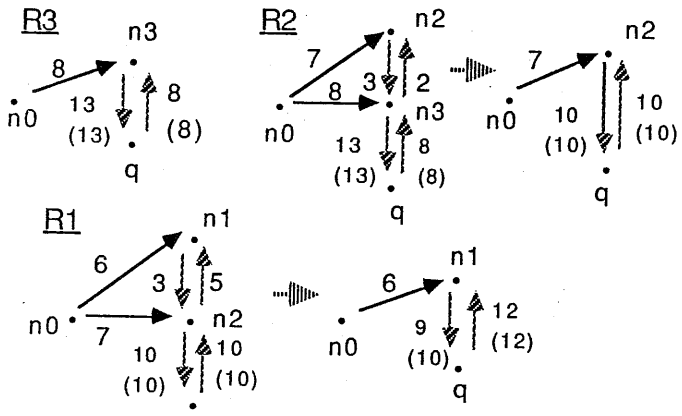

Figure 10: reduced graphs of Fig.9

$\rightarrow$ n2, n4 $\rightarrow$ n3 $\rightarrow$ n2 } []

Binary topological WTPG (Binary Tree LCP) is solved in linear order by reduce. Binary-Tree means "WTPG except T0 and Tf forms a binary tree topology". The name of the game is to generate the reduced graph R_k from R_{k+1}, such that the node n_k has two son-nodes and one parent node. This case has to examine only $2^3 = 8$ possible pairs of resolution on choice edges.

5 Concluding Remarks

In this paper, we have pointed out 'convoy' of batch transactions occurs when they run concurrently on multi disk database machines (DBM). The convoy prevents multiple disks of DBM from running in parallel.

In order to predict the convoy, we have proposed Weighted Transaction Precedence Graph (WTPG) and a cautious scheduler using it. The scheduler generates a serializable schedule with the least critical path in a given WTPG. Strategic solutions are presented in the restricted problems "Chain WTPG" and "Binary tree WTPG". They are computed in linear time order of the number of nodes in a WTPG; i.e. multi programming level of batches.

The following points must be extended in our study.

1. Polynomially solvable class of WTPG.
2. Addition of control edges in WTPG. — *control-edge* defines only a serializable order of some transactions, without weight. We must decide the least critical path such that the generated SR-order obeys the order specified by control edges. control-edges are used in mix-scheduling of on-line transactions and batches.
3. Recovery of batches in WTPG.

Appendix

Proof of lemma1: By the definitions of WTPG, all parameters in WTPG are non-negative. When the transaction n_{k+1} is preceded by the transaction n_k, n_{k+1} can commit at $r_k + \alpha_k$ at earliest. By the definition of r_{k+1}, apparently the lemma holds. []

References

[Bay86] Bayer,R. 'consistency of transactions and random batches', *ACM Trans. on Database Systems*, 11(4):397-404, 1986.

[DeW86] DeWitt,D.J. et al. 'Gamma - High Performance Dataflow Database Machine', In *Proc. 12th Int. Conf. Very Large Data Bases*, 228-237, 1986.

[DeW87a] DeWitt,D.J. et al, 'A Single User Evaluation of the Gamma Database Machine', In *Proc. 5th Int. Workshop on Database Machines*, 43-59, 1987.

[DeW87b] DeWitt,D.J. et al, 'A Single User Performance Evaluation of the Teradata Database Machine', Tech. Rep., DB-081-87, MCC, 1987.

[Gra79] Graham,R.L. et al, 'Optimization and approximation in deterministic sequencing and scheduling: a survey', In *Annals of Discrete Mathematics 5*, 287-326, 1979.

[MS87] Molina,H.G. et al, 'SAGAS', In *Proc. ACM-SIGMOD annual Conf. '87*, 249-259, 1987.

[Nis87] Nishio,S. et al. 'Performance Evaluation on Several Cautious Schedulers for Database Concurrency Control', In *Proc. 5th Int. Workshop on Database Machines*, 212-225, 1987.