データベース・システム 87－5
（１９９２．３．１６）

障害回復に適した先読みスケジューラ

木庭　淳　　加藤直樹
神戸商科大学管理科学科

データベースシステムのスケジューラはデータの首尾一貫性を保証するために，各トランザクションのデータ項目への読出し(read)・書込み(write)要求の発生時に，その要求の遅延か許可，もしくはトランザクションのアボートを判断する．このうち先読みスケジューラは，データベースに様々な障害が起こった場合を除く正常な実行の範囲内において，後退復帰を必要とせず，デッドロックが生じないという優れた特性を持っている．すなわち，時刻印方式や二相ロック方式に見られるような，並行処理制御の方法自体が原因となるアボートは生じないことが保証されているため，デッドロックに対する対策は不要である．しかし実際問題として，ごくまれにトランザクションレベルあるいはシステムレベルでの障害の可能性はあるため，障害回復への対策は先読みスケジューラといえども避けることはできない．本稿では，単版方式においては，アボートが起こったとしてももとのデータ項目の値を回復できるようなアボート影響回避先読みスケジューラを提案し，一方多版方式においては，あるトランザクションのアボートが他の正常なトランザクションのアボートを引き起こすことのない，連鎖的アボート回避先読みスケジューラを提案する．

CAUTIOUS TRANSACTION SCHEDULERS SUITABLE FOR RECOVERY CONTROL

Jun Kiniwa　　Naoki Katoh

Department of Management Science,
Kobe University of Commerce,
Kobe-shi, 655 Japan

Abstract

Recovery control is an important issue in the practical design of transaction schedulers for database concurrency control. In view of this, we propose a single-version cautious scheduler which outputs strict schedules and a multiversion cautious scheduler which does not cause cascading aborts. Our main idea is to always assign each read operation to a committed write operation in order to avoid cascading aborts. Furthermore, it is not to allow executing each write operation until the last write on the same data item is committed in order to output strict schedules. We shall also compare the degree of concurrency attained by our schedulers with that by existing cautious scheduler.

1 まえがき

データベースシステムで複数のトランザクションが共有データにアクセス（読出しと書込み）する環境下で，データの首尾一貫性を損なわないように，かつスループットをいかに高めるかというのが並行処理制御の問題である．並行処理制御を実現する方式（ここではそれをスケジューラという）として従来より時刻印方式[9]，二相ロック方式[2]があるが，近年になって各トランザクションがあらかじめ将来アクセスするデータ集合をスケジューラに知らせるという条件のもとで，デッドロック等の欠点をなくした，先読みスケジューラ[4,5,6,10]が提案された．

一方障害回復の問題は，オフライン環境下での単版方式のスケジュールに対し，信頼可能性を調べる問題がPSPACE完全であること[8]が知られており，また障害に耐えるスケジュールのいくつかの性質[1]が研究されている．またオンライン環境下では，各トランザクションがそのコミット時（既にトランザクションの処理が完了して今後無効とされないことが保証された時点）まで全てのロックを解放しない狭義二相ロック方式により，障害に耐えるスケジュールが実現されることが知られている[1]．

しかし，いまだ先読みスケジューラに関して障害に耐えるスケジュールを如何に構成するかは論じられていない．そこで本稿では，単版方式においては，トランザクションの中途破棄（アボート）が起こったとしても，アボートしたトランザクションが更新する以前のデータベース状態（すべてのデータ項目値）を回復し，かつ正常なトランザクションにまでアボートが波及する（連鎖的アボート）事態を避けるような，アボート影響回避先読みスケジューラを提案する．さらに多版方式においては，連鎖的アボート回避スケジューラを提案する．

2 諸定義および基礎概念

2.1 データベースシステム

データベースシステムはユーザの発行するトランザクションの集合 $\mathbf{T}=\{T_1, \cdots, T_n\}$ とデータベース上のデータ項目の集合 $\mathbf{D}=\{x,y,z, \cdots\}$ からなる．トランザクション T_i はデータ項目 $d \in B$（但し $B \subseteq \mathbf{D}$）を読み出すいくつかの**読出しステップ** $R_i[B]$ と $d \in B$ に書き込む**書込みステップ** $W_i[B]$，および**コミット操作** c_i または**アボート操作** a_i からなり，ステップ数は任意とする．ある読出しステップ $R_i[B]$（または書込みステップ $W_i[B]$）において $d \in B$ に対する $R_i[d]$（または $W_i[d]$）をそれぞれ**読出し操作**，**書込み操作**という．1つのトランザクションにおいてあるデータ項目 d に対し，読出しも書込みも行う場合，読出し操作の後で書込み操作を実行するものとする．また1つのトランザクションが同一のデータ項目に対し，重複した読出しまたは書込み操作をもたないものとする．

時間の流れにおいて，あるデータ項目 d への書込み操作群による過去からの値を**版**という．システムの方式として過去に書込んだ版すべての存在を許すものを**多版方式**，各データ項目につき最新版の存在のみを許すものを**単版方式**という．

各トランザクション T_i は必ずコミット操作 c_i またはアボート操作 a_i により終了する．コミット操作が行なわれたトランザクションはその実行が正しく行なわれたことが保証され，アボート操作が行なわれたトランザクションはそのすべての影響が抹消される．システム内にはすべてのステップを終了した**コミット済み**トランザクションと，処理すべきステップが残っている**コミット予定**トランザクションが存在する．

各ユーザの発行するトランザクションのステップおよびコミット操作をスケジューラに到着順に左から右へ並べた**系列**を考える．系列中

のステップに関する全順序を<で表す．系列の前後に仮想的に $W_0[\mathbf{D}]$ のみからなる**初期トランザクション** T_0（$W_0[\mathbf{D}]$ は実行と同時にコミットも行なうものとする）と，$R_f[\mathbf{D}]$ のみからなる**最終トランザクション** T_f を付加し，$W_i[d] < R_j[d]$(系列中で $W_i[d]$ の方が $R_j[d]$ より前)を満たす読出し操作集合から書込み操作集合への写像である**割り当て** I を考える．すなわち $I(R_j[d]) = W_i[d]$ と表すとき，$W_i[d]$ による値を $R_j[d]$ が読み出すものとする．全ての読出し操作に直前の書込み操作による値を割り当てる場合を特に I_0 と表す．I を考慮した系列 h を**スケジュール**といい $< h, I >$ と表わす．$h = h_a h_b$ のように任意のステップで系列 h を前後に分けたとき，$< h_a, I_a >$ を**部分スケジュール**という．I_a は割り当て I の系列 h_a における読出し操作集合への限定である．ここで $I(R_j[d]) = W_i[d]$ なる関係があるとき，T_i をアボートすれば必然的に T_j もアボートしなければならないことに注意されたい．

〔定義 2.1：障害に耐えるスケジュール〕[1]
スケジュール $< h, I >$ において任意の 2 つのトランザクション T_i, T_j の間に次の関係が成り立つとき，各々 $< h, I >$ を，(a) **回復可能** (recoverable) スケジュール，(b) **連鎖的アボート回避** (avoiding cascading abort) スケジュール，(c) **アボート影響回避** (strict) スケジュールと呼ぶ．

(a) $I(R_j[d]) = W_i[d]$ ならば $c_i < c_j$.

(b) $I(R_j[d]) = W_i[d]$ ならば $c_i < R_j[d]$.

(c) $W_i[d] < R_j[d]$ ならば $c_i < R_j[d]$, $W_i[d] < W_j[d]$ ならば $c_i < W_j[d]$. □

障害に耐えるという意味からすると，(c) が最も好ましく，(a) は最も好ましくないとされている[1]．但し (a),(b),(c) のスケジュール集合間には真の包含関係が成り立ち[1]，並行性に関しては (a) が最も高く，(c) が最も低い．

まず単版方式を考えると，アボートすべきトランザクションがあるデータ項目値を更新していたとすると，障害回復後もその更新前の値は容易に回復できない．従ってトランザクションがある値を更新しようとするときにはその値を書き込んだトランザクションが既にコミットされていることが必要であり，すなわち (c) を満たすことが望ましいと考えられる．なお (c) が満たされれば (a),(b) も満たされる．

一方多版方式においては，アボートしたトランザクションの影響を避けることは簡単にできる（つまり (c) を満たす必要はない）．すなわちアボートすべきトランザクションがあるデータ項目の値を更新していたとしても，それ以前の値を用いることにより，アボートトランザクションが最初から無かったスケジュールと同じ実行が可能である[1]．しかし (a) のスケジュールは，並行性は高いが障害が起きた場合に不適切と考えられる．従って多版方式において望ましいと考えられるのは，(b) の連鎖的アボート回避スケジュールのクラスである．

2.2 直列化可能性

オフラインスケジュール（完結したスケジュール）において，スケジュール $< s, I_0 >$ が**直列**であるとは，任意のトランザクションの実行中に他のトランザクションのステップが割り込まないことをいい，スケジュール $< h, I >$ が**直列化可能**であるとは，$< h, I >$ と同じステップからなる直列な $< s, I_0 >$ に対して，$I = I_0$ となることをいう[1,3,8]．

我々の提案する先読みスケジューラ（3 章）においては，将来来ることが予定されているステップが，既に到着したトランザクションについては予めわかっている．つまりオフラインと同じく，常に完結したスケジュールを予想してリクエストの許可・遅延を決定するため，従来の直列化可能性の議論を用いなければならない（付録 参照）．

直列化可能性を容易に判定するために，**TIO グラフ** (transaction input/output graph) が導入された[3]．

〔定義 2.2 : TIO グラフ〕[3] $TIO(<h,I>)$ は節点集合 **T** ∪ **T'** と有向枝集合 **A** からなるラベル付き有向グラフである．$<h,I>$ において $I(R_j[d]) = W_i[d]$ のときラベル d 付き有向枝 $(T_i,T_j) \in$ **A** を付加する（$(T_i,T_j) : d$ と表記し，**読出し枝**と呼ぶ）．また $<h,I>$ において $I(R_j[d]) = W_i[d]$ を満たす $R_j[d]$ が存在しないとき，つまり書込んだ値がどのトランザクションにも読まれなかったとき（$W_i[d]$ を**無効書込み操作**と呼ぶ），実際のトランザクションには対応しないダミー節点 $T_i' \in$**T'** をつくり，ラベル d 付きダミー有向枝 $(T_i,T_i') : d \in$ **A** を付加する． □

このとき以下に定義される DITS(disjoint-interval topological sort) が $TIO(<h,I>)$ に存在することが $<h,I>$ が直列化可能であることと同値であることが示されている[3]．

〔定義 2.3 : DITS〕[3] $TIO(<h,I>)$ の節点に関するある全順序 $\ll$ が次の 3 条件を満たすとき，DITS であるという．

(1)$T_i \ll T_j$ ならば T_j から T_i への有向路は存在しない．(2)$g \neq j$ なる任意の $(T_g,T_i) : d$，$(T_j,T_k) : d$ に対して，$T_g \ll T_k$ ならば $T_i \ll T_j$ が成り立つ．(3)$T_i \neq T_0, T_f$ なる T_i に対して $T_0 \ll T_i \ll T_f$ である．

ここで (2) における (T_i,T_j) を**排除枝**と呼ぶ． □

直列化可能性の判定は一般に NP 完全であるので[3,8]，TIO グラフに wr,rw,ww 等の制約枝を入れてこの判定を多項式オーダーで行なう方法が提案されている[3]．

3 障害回復に適したスケジューラ

3.1 先読みスケジューラのしくみ

各トランザクション T_i が第一ステップ発行時に，T_i の書き込むデータ集合 WS_i と読み出すデータ集合 RS_i を予告する条件下で，先読みスケジューラは直列化可能スケジュールを生成する[4,5,6,7]．このときデッドロック[2]は生じず，後退復帰[9]も不要である．なお各トランザクションは自己のステップが許可されて初めて次のリクエストを出すものとする[4,5,6,7]．

スケジューラが各時点で保持する情報を次のように表記する．すなわち $<P,I>$ を，既に出力した部分スケジュール．q を，許可するか否かを現在判定中のステップのリクエスト．$PEND$ を，予告されているが**未実行ステップ**の集合（未実行コミット操作も含む）．DEL を，**実行遅延ステップ**の集合（$DEL \subseteq PEND \subseteq$ {コミット予定ステップ}）[1]とする．

スケジューラは，$PEND$ 中の全ステップからなる系列 Q で，qQR_f[**D**] のある適当な割り当て I' に対し $<PqQR_f$[**D**], $II'>$ が直列化可能となる Q が存在するかどうかを調べ（**完成テスト**），直列化可能ならば q を許可し DEL 中で実行可能なステップを調べる．直列化可能でなければ q を遅延して DEL に入れ，次の到着ステップについて実行可能かどうかを調べる．

すなわち完成テストは，既に到着しているトランザクションの情報のみから，現在のリクエスト q を許可しても将来直列化可能スケジュールとなり得るかどうかを調べる．手続きの詳細については文献 [4,5,6,10] 等を参照されたい．

〔定義 3.1 : 完成テスト〕[5] 完成テストが成功するとは，次の 3 条件が成り立つような $PEND$ の全ステップからなる系列 Q が存在することである．(1)Q のステップの順序は各 T_i のステップの順序に矛盾しない．(2) 今後 qQ の順序でステップを実行したとき，先読みスケジューラは全てのステップを遅延させることなく許可する．(3)(2) において定まる qQR_f[**D**] の割り当てを I' とすると $TIO(<PqQR_f$[**D**], $II'>)$ は DITS をもつ． □

完成テストを実行するため，次の **ATIO** (Active TIO) グラフを利用する．ATIO グラフは

[1] コミット予定ステップは，実行されているがコミットされていないステップと，未実行ステップからなる．

TIO グラフのオンライン版である．

〔定義3.2:ATIO グラフ〕[4,5,6,10] $< P, I >$, q, $PEND$ を上で定義されたものとするとき，$ATIO(< P, I >, q, PEND)$ は $P \cup q$，もしくは $PEND$ に属するステップをもつトランザクション，最終トランザクション T_f およびダミー節点から構成される節点集合と，2つの互いに素な有向枝の部分集合 **A** と **A'** からなるラベル付き有向グラフである．**A** は $< P, I >$ に含まれる読出し操作に対応する読出し枝と，$< P, I >$ における無効書込み操作 $W_i[d]$（$I(R_j[d]) = W_i[d]$ なる $R_j[d] \in P$ が存在しない）に対応するダミー有向枝 $(T_i, T_i') : d$（T_i' はダミー節点）とから成り，**A'** は $\{q\} \cup PEND$ に属する書込み操作 $W_i[d]$ に対応するダミー有向枝 $(T_i, T_i') : d$（T_i' はダミー節点）から成る． □

3.2 障害に耐えるスケジュールの実現

我々のスケジューラは，障害に耐えるスケジュールを実現するために cc 制約という制約を導入する．従来の単版スケジューラ $CS(\overline{WRW})$ [4] に対応するアボート影響回避スケジューラを $CS_{ST}(\overline{WRWC})$，多版スケジューラ MCS $(MWRW)$ [5] に対応する連鎖的アボート回避スケジューラを $MCS_{CA}(MWRWC)$ と表すことにする．なお $CS_{ST}(\overline{WRWC})$ では $ATIO_{\overline{wrwc}}$ グラフ，$MCS_{CA}(MWRWC)$ では $ATIO_{mwrwc}$ グラフを用いる．

3.2.1 単版アボート影響回避スケジューラ $CS_{ST}(\overline{WRWC})$

$CS_{ST}(\overline{WRWC})$ で用いる $ATIO_{\overline{wrwc}}$ グラフの制約枝を定義する．なお以下において q が遅延されれば対応する (B) の有向枝を除くが，(A) の節点・有向枝は除かない．

〔定義 3.3：$ATIO_{\overline{wrwc}}$ の節点・制約枝〕

(A) q が T_q の最初のリクエストのとき節点 T_q を加え，

- **(A-1)** P において最新の書込み操作 $W_{last}[d]$（$d \in WS_q$）に対して ww 枝 (T_{last}, T_q) を加える．
- **(A-2)** このとき読込み枝 $(T_{last}, T_g) : d$（$d \in WS_q$）をもつ T_g があれば rw 枝 (T_g, T_q) を加える．
- **(A-3)** $W_i[d'] \in P$ $(d' \in RS_q)$ に対して wr 枝 (T_i, T_q) を加える．

(B) q が T_q の最初のリクエストか否かにかかわらず

$q = R_q[B]$ のとき，

- **(B-1)** $W_{pend}[d] \in PEND$（$d \in B$）に対して rw 枝 (T_q, T_{pend}) を加える．
- **(B-2)** また P における最新の $W_{last}[d]$（$d \in B$）がコミットしていなければ，rwc 枝 (T_q, T_{last}) を加える．

$q = W_q[B]$ のとき，

- **(B-3)** P における最新の $W_{last}[d]$ $(d \in B)$ に対して加えられていた，以下の2種類の枝を変更する．
  - $old \neq last$ なる $W_{old}[d] \in P$ からの ww 枝 (T_{old}, T_{last}) を消し，(T_{old}, T_q) に付け変える．
  - $W_{pend}[d] \in PEND$ への ww 枝 (T_{last}, T_{pend}) を消し，(T_q, T_{pend}) に付け変える．
- **(B-4)** $R_{pend}[d] \in PEND$ に対して wr 枝 (T_q, T_{pend}) を加える．
- **(B-5)** また P における最新の $W_{last}[d]$（$d \in B$）がコミットしていなければ，wwc 枝 (T_q, T_{last}) を加える．

(C) $q = c_q$ のとき，
グラフに変化はない．

(D) $q = a_q$ のとき，
節点 T_q と T_q に付随する全有向枝を除去する． □

$ATIO_{\overline{wrwc}}$ において，(A) は T_q の宣言する読出し・書込みデータ集合に関して最新の書込み操作をもつ節点 T_{last} から新節点 T_q への枝（実行済み操作をもつ T_{last} と，未実行操作をもつ T_q

に対する順序付け)，(B) は現在のリクエスト q をもつ節点 T_q からコミット予定操作をもつ節点 T_{pend} への枝，および q と $PEND$ の操作に対する順序付けの枝である．それぞれ同じデータ項目 d に対する読出し・書込み操作に対して付加される．

この制約枝は従来の先読みスケジューラでの制約枝に障害回復を考慮した枝 wwc および rwc が加えられていることに注意されたい．なぜ $ATIO_{\overline{wrwc}}$ グラフによりアボート影響回避スケジュールが実現できるのかを直観的に述べると，$\overline{wrwc}$ 制約枝により現在の読出し・書込みリクエスト q の直前には常にコミット済み書込み操作が直列化されるため，読み出し操作に対する割り当てがコミット済み書込み操作より行なわれ，書込み操作は現在値を更新したトランザクションがコミットされた後に実行されるからである．

3.2.2 多版連鎖的アボート回避スケジューラ $MCS_{CA}(MWRWC)$

$MCS_{CA}(MWRWC)$ で用いる $ATIO_{mwrwc}$ グラフの制約枝を定義する．

〔定義 3.4：$ATIO_{mwrwc}$ の節点・制約枝〕

(A) q が T_q の最初のリクエストのとき節点 T_q を加え，

- **(A-1)** P のコミット済み $W_c[d]$ ($d \in RS_q$) に対して wr 枝 (T_c, T_q) を加える．
- **(A-2)** またもし読込み枝 $(T_c, T_g) : d$ ($d \in WS_q$) をもつ T_g があれば rw 枝 (T_g, T_q) を加える．

(B) q が T_q の最初のリクエストか否かにかかわらず

$q = R_q[B]$ のとき，

- **(B-1)** $W_{pend}[d] \in PEND$ ($d \in B$) に対して rw 枝 (T_q, T_{pend}) を加える．
- **(B-2)** また P の $W_k[d]$ ($d \in B$) がコミットしていなければ，rwc 枝 (T_q, T_k) を加える．

$q = W_q[B]$ のとき，

- **(B-3)** $R_{pend}[d] \in PEND$ ($d \in B$) に対して wr 枝 (T_q, T_{pend}) を加える．

(C) $q = c_q$ のとき，

グラフに変化はない．

(D) $q = a_q$ のとき，

節点 T_q と T_q に付随する全有向枝を除去する． □

$ATIO_{mwrwc}$ グラフにおいても，制約枝の付け方の発想は単版方式の場合と同様であり，読出し操作の直前にコミット済み書込み操作を直列化することを目的とした枝を付加する．

スケジューラの完成テストは，$ATIO^*_{cc}$ グラフ[II]の閉路テストと等価である．すなわちグラフに閉路がないときのみ完成テストは成功する．この正当性の証明の詳細についてはここでは省略し，方針のみを記す．

1) 各スケジューラで使用する $ATIO^*_{cc}$ グラフに閉路がないことと，$ATIO_{cc}$ グラフに DITS が存在することが同値であること．

2) $ATIO^*_{cc}$ グラフに閉路がないとき現在のリクエスト q が実行可能であること．

1),2) を示した上で，

3) q を許可しても，後々のスケジュールで DITS が存在する可能性があること，すなわち完成テストに成功することを示す．

4) さらに完成テストに成功すれば出力スケジュールの直列化可能性，および現時点でのコミット直列化可能性が保証されることを示す．

1)–4) より $ATIO^*_{cc}$ グラフに閉路がないとき完成テストに成功し，このとき各読出し・書込み操作は，障害発生時にアボートトランザクションさえ除去すれば，直前のデータベース状態を回復できるようにスケジューリングされる．

[II] $ATIO_{cc}$ グラフに排除枝を可能な限り加えた排除閉包を表わす．

4 性能評価

4.1 無遅延クラス

スケジューラがどのステップも遅延させないで出力するような系列の集合を，**無遅延クラス** (fixed point set) と呼び，$CS_{ST}(\overline{WRWC})$ の無遅延クラスを $fp(\overline{WRWC})$ と表す．また $MCS_{CA}(MWRWC)$ に対して $fp(MWRWC)$ と表す．これら無遅延クラスの大きさを比較することにより，先読みスケジューラの並行処理性能を理論的に比較することができる[4,5,6,10]．なお以下ではすべて証明を省略するが，定理 4.1 では多版方式と単版方式のスケジューラについて比較するので，条件を同じにするために，$fp(MWRWC)$ は割当て I_0 を行なったときに得られる無遅延クラスとする．

〔定理 4.1〕 $fp(\overline{WRWC}) \subset fp(MWRWC)$．□

4.2 取消し異常と読出し追加異常

先読みスケジューラはトランザクション発行時にステップ集合を宣言するが，トランザクション実行中のステップ変更のうち，宣言したステップを未然に取り消したとき以後のスケジュールに支障を生じる（**取消し異常**）かどうかにより性能評価を行う[4,5,10]．また宣言していない読出し操作を新たに $PEND$ に追加するとき，スケジュールに支障を生じる（**読出し追加異常**）かどうかについても調べる．

〔定理 4.3〕 (1)$CS_{ST}(\overline{WRWC})$ は取消し異常を生じない．
(2)$MCS_{CA}(MWRWC)$ は取消し異常を生じない．□

〔定理 4.4〕 (1)$CS_{ST}(\overline{WRWC})$ は読出し追加異常を生じる．
(2)$MCS_{CA}(MWRWC)$ は読出し追加異常を生じる．□

4.3 計算機実験

4.3.1 実験方法

ここでは主として調べたのは

実験 1 トランザクションが集中したとき各スケジューラのスループットにはどの程度差が出るか．
実験 2 データベースの大きさがスケジューラに及ぼす影響はどうか．
実験 3 トランザクションの性質が変わるとスケジューラの効率はどうなるか．

という点である．

具体的には表 1 において，変化させるパラメータ毎に各評価尺度を測定する実験を行なう．なお表 1 中の基準値は，他の変数を変化させているときにとる値を示す．

モデルを簡単にするために次の仮定をおく．

- 完成テストの実行時間は 0．
- ステップの処理時間は 0．

次にスケジューラの特性を調べるための評価基準を述べる．

a) ステップ平均待ち時間
（ステップの許可時刻）－（ステップの到着時刻）を各ステップ毎に計算し，その平均をとる．

b) ステップ遅延率
（少なくとも 1 回は遅延されたステップ数）／（全体のステップ数）．

c) 平均トランザクション数
システム内に存在する実行中トランザクションの数．

1 回のシミュレーションの実行におけるトランザクション発生数を 3000 個とし，20 回試行を行なってその平均をとるものとする．

4.3.2 実験結果

4.3.2.1 実験 1：トランザクション到着頻度の影響 トランザクションの到着頻度が高まるほど，全体的に処理効率は悪くなる．ステップの遅

れ時間に関しては，トランザクションの到着頻度が相当高まったとき，特にステップの到着頻度より高く（$t\text{-}to\text{-}t \leq 5$）なれば，$MCS_{CA}(MWRWC)$は悪くなるが，そうでないときは$CS_{ST}(\overline{WRWC})$よりも良好である．ステップが遅延される割合については，$MCS_{CA}(MWRWC)$は障害回復を無視した単版スケジューラ$CS(\overline{WRW})$とほとんど性能は変わらない．

4.3.2.2 実験2：データ項目数の影響 一般的にデータ項目数が減ると各トランザクションの競合が起こりやすくなるので，効率が悪くなるのは明かである．ステップの遅れ時間についてはデータ項目数が一つのトランザクションの最大アクセス項目数（$MAXRW$=10）に比べて極端に少ない場合（$d\text{-}size$=10）を除いて$MCS_{CA}(MWRWC)$は良好で$CS(\overline{WRW})$に近い．またステップが遅延される割合も同様なことが言える．

4.3.2.3 実験3：読出し・書込み集合の重複度の影響 一つのトランザクションがアクセスするデータ集合について，読出しデータと書込みデータの重複の度合の影響を調べる．重複度が高い，すなわち読出したデータについてはほとんど書込むトランザクションが多いとき，多版方式の利点は理論的にも失われるので，すべての評価基準で$CS_{ST}(\overline{WRWC})$と$MCS_{CA}(MWRWC)$は一致する値を出している．また重複度が高くなればなるほどステップ遅延時間，遅延率共に徐々に増加する傾向が見られるが，これは同じデータ項目に対する書込み命令は必ず読出し命令後に発行する，というトランザクションの規則により，先に到着したトランザクションの書込み命令が予定ステップとして残っており，これが他のトランザクションとの競合を起こし易くすると考えられる．

4.3.2.4 結論 トランザクション数が非常に多く，ステップの到着頻度よりも高いような高頻度で到着するとか，データベースのアクセス可能範囲が極端に限られている場合を除いて，$MCS_{CA}(MWRWC)$は$CS_{ST}(\overline{WRWC})$よりも優れた性能を持つと思われる．また同時に$CS(\overline{WRW})$も実験しているが，これは障害回復を全く考えていないスケジューラなので，一律に比較すべきではない．ただ同種の短時間トランザクションのみが存在するシステム環境ならば，遅延等は図で表わされる程度であり，ハードウェア等の進歩と障害回復の重要性を考えれば，本稿で提案する$MCS_{CA}(MWRWC)$および$CS_{ST}(\overline{WRWC})$は充分実現可能なシステムであると考えられる．

5 むすび

本稿では，直列化可能性を判断するグラフにおいて適当な制約枝を付加することにより，たとえ将来あるトランザクションのアボートが起こったとしても，その影響を他のトランザクションに及ぼさず，かつデータベース状態もただちに回復できるようなスケジューリングを行なう先読みスケジューラについて考察した．この場合，単版・多版それぞれの方式の違いによって，推奨さるべき方法が異なることも指摘した．さらにシミュレーションにより，具体的な性能評価の手がかりも提供した．

付録

一般的にはオンラインスケジュール（実行過程のスケジュール）における議論は，厳密にはコミット済みトランザクションに限定して考えなければならない[1]．すなわち$C(h)$を，系列hよりコミット予定トランザクションのステップを除いた系列とすると，**コミット射影スケジュール**$< C(h), I^c >$は部分スケジュール$< h, I >$に対

して定義される．但しI^cはIの$C(h)$における読出し操作集合への限定である．またスケジュール$< h, I >$がコミット直列化可能であるとは，$< h, I >$の任意の部分スケジュール$< h_a, I_a >$のコミット射影スケジュール$< C(h_a), I_a^c >$が直列化可能であることである[1]．

謝辞

日頃ご指導頂く京都大学長谷川利治教授に深謝致します．また理論で京都大学茨木俊秀教授，Simon Fraser大学亀田恒彦教授，大阪大学西尾章治郎助教授にご助言頂き，実験では神戸商科大学力宗幸男助教授にご支援頂きました．衷心より感謝の意を表します．

参考文献

[1] Bernstein,P.A.,Hadzilacos,V. and Goodman,N.: "Concurrency control and Recovery in Database Systems", Addison-Wesley (1987).

[2] Eswaran,K.P.,Gray,J.N.,Lorie,R.A.and Traiger, I.L. : "The Notions of Consistency and Predicate Locks in a Database System", Comm. ACM,19, 11, pp .624-633 (1976).

[3] Ibaraki, T. ,Kameda, T. and Minoura, T.: "Serializability with Constraints", ACM Trans. Database Syst., 12, 3, pp.429-452 (1987).

[4] Ibaraki, T., Kameda, T. and Katoh, N.: "Cautious Transaction Schedulers for Database Concurrency Control", IEEE Trans. Softw. Eng., 14, 7, pp.997-1009 (1988).

[5] Ibaraki, T., Kameda, T. and Katoh, N.: "Multiversion Cautious Transaction Schedulers for Database Concurrency Control", IEEE Trans. Softw. Eng., 16, 3, pp.302-315 (1990).

[6] Katoh, N., Ibaraki, T. and Kameda, T.: "Cautious Transaction Schedulers with Admission Control", ACM Trans. Database Syst., 10, 2, pp.205-229 (1985).

[7] Katoh, N., Kameda, T. and Ibaraki, T.: "Efficient Implementation of Cautious Schedulers", LCCR TR90- 13, Dept. of Comput. Sci., Simon Fraser Univ., Canada (1990).

[8] Papadimitriou, C.H.: "The Theory of Database Concurrency Control", Computer Science Press (1986).

[9] Reed, D.P.:"Naming and Synchronization in a Decentralized Computer System", MIT/LCS TR-205 Dept. of Electrical Engineering and Comput. Sci. M.I.T., (1978).

[10] 武田真人, 増山繁, 茨木俊秀：”版数制限をもつ先読みスケジューラ”, 信学論 (D), J70-D, 8, pp.1478-1486 (1986).

実験 No.	変数名	値の範囲	基準値	意味
実験 1	*t-to-t*	1–15	10	トランザクション平均到着時間間隔
実験 2	*d-size*	10–100	30	データ項目数
実験 3	*ov*	0–100%	30%	読出し・書込みデータ集合の重複度

表 1: 変化させるパラメータ

定数名	値	意味
NUM-T	3000	トランザクション発生数
S-TO-S	5	ステップ平均到着時間間隔
MAXRW	10	トランザクション当り最大アクセスデータ項目

表 2: 定数