データベースオペレーティングシステム OPT-R のタスク管理とトランザクションのスケジューリング技法†

大久保 英嗣†† 津田 孝夫††

本論文では，ホストオペレーティングシステム上のデータベース管理システムにおける性能の問題を解決し，システムの適応性を向上させるための，データベースオペレーティングシステム OPT-R のタスク構成とその管理方式について述べる．さらに，一つのトランザクションにおける並行処理可能なタスクの認識に基づく，データベーストランザクションのスケジューリングアルゴリズムを提案する．本アルゴリズムは，システムの処理多重度を高めることを目的としている．

1. はじめに

データベース管理システム（以下 DBMS と記す）は，ホストオペレーティングシステム（以下 OS と記す）が生成するタスクにより，その処理を行う．すなわち，DBMS に割り当てられたタスクは，OS の制御下で稼動する他のタスクと一括して管理される．したがって，DBMS としてマルチプログラミング環境を意識したスケジューリング（I/O バウンド処理と CPU バウンド処理の可能なかぎりのオーバラップスケジューリング）を行っても，タスクスイッチ等，上位の OS の有するオーバヘッドのためにその意図が反映されなくなるといった問題が生じている[1),2)]．

われわれは，データベースに関する処理を効率よく行うために，データベース処理に目的を限定したデータベース専用のオペレーティングシステム OPT-R を設計開発している[9)]．本論文では，従来の DBMS における上述のタスク管理の問題点を解決するために，OPT-R で採用しているタスク構造とタスクのスケジューリングの方法について述べる．さらに，本タスク管理技法をデータベーストランザクションのスケジューリングに適用して，システムの性能を向上させる方法を提案する．

2. OPT-R のタスク構成

OPT-R のタスクは，ユーザ管理，トランザクション処理，データ操作等の各処理単位に一つのタスクを割り当てるという単純な構成をとっている．各タスクには，タスク制御ブロック（TCB）と呼ばれるシステムテーブルが割り付けられ，後述の大域的スケジューリングの対象となる．

OPT-R のタスクは，システム管理用のタスクとユーザ要求管理用のタスクに大別される．以下，これらについて説明を加える．

2.1 システム管理用タスク

従来の OS におけるシステム管理のためのタスク群と同等の機能を有するもので，システムの監視および DBA の制御を司る．これらは，システム初期化時に生成される．

（1） DBA コンソールタスク

DBA が，コンソールを介してシステムに特別な指示を与える際の入出力を管理する．

（2） DBA ブレークタスク

DBA によるコンソールからのブレーク割込みを監視し，入力されたコマンドを解析して実行する．

（3） 入出力障害回復タスク

入出力障害を検出し，入出力装置対応に標準回復処理を行う．

（4） 非常駐スーパバイザタスク

主メモリに存在しない SVC 処理モジュールを呼び出す SVC 割込みに応じて，システムオーバレイ領域へそのモジュールをローディングする．

（5） デッドロック検出解消タスク

OPT-R では，複数ユーザにデータの共有を許している．その際の同時アクセス制御として共有ロック（shared lock）方式を採用している．この方式によって生じるデッドロックを検出し，バックアウトによりそれを解消するタスクである．

（6） アイドルタスク

I/O パトロール（入出力終了割込みの監視）を行

† The Technique of Task Management and Transaction Scheduling in the Database Operating Systems OPT-R by EIJI OKUBO and TAKAO TSUDA (Department of Information Science, Faculty of Engineering, Kyoto University).

†† 京都大学工学部情報工学科

う．本タスクは，最も優先順位が低いため，他のいかなる割込みも受け付け，他のタスクに制御が渡る．

2.2 ユーザ要求管理用タスク

OPT-R に固有のタスクで，ユーザの要求を処理するタスクである．とくに，OPT-R の使用目的を考え，データベースへのアクセス処理を中心にタスクを構成して処理効率の向上を図っている．以下に，各タスクの機能を述べる．

（1） 端末制御タスク（TIT）

ユーザからのブレーク割込みの受付けおよび以下に述べる UMT の生成と消滅を管理するタスクであり，システム初期化時に一つ生成される．このタスクは，以下に述べる UMT および TRT の親タスクとなるものであり，OPT-R のユーザ全体を管理するタスクであるといえる．

（2） ユーザ制御タスク（UMT）

OPT-R との間のセッションを開設するユーザ対応に，TIT によって生成される．ユーザに直接関与するタスクで，ユーザと端末との間の入出力制御およびユーザインタフェース，トランザクション管理モジュールもこのタスクで動作する．

（3） トランザクションタスク（TRT）

ユーザの要求するトランザクション対応に，UMT によって生成される．このタスクは，トランザクション管理により生成された内部コード系列からなり，個個のトランザクションを管理する．

（4） データベース操作タスク（DOT）

各データベース操作を実行するタスクで，システム初期化時に生成される．4 章で詳述するが，TRT と同期をとりながら，ユーザの要求したトランザクションを処理する．

3. 大域的スケジューリング

3.1 従来の DBMS における問題点

複数ユーザをサポートするために，DBMS で一般的にとられている方法は，各ユーザ対応に OS のタスクを一つ割り当てることである．この場合，実行中のタスクは，バッファに存在しないデータを参照するたびにタスクスイッチを生じさせる．すなわち，要求したデータが参照可能になるまで当該タスクの処理は中断され，他のタスクが実行されることになる．汎用的な OS は，データベースユーザ用のタスクも含めて，大量のタスクに関する情報を管理し，スケジュールしているので，そのオーバヘッドはたんに入出力にかかる負荷としては大きすぎる．

Gray は，データベースに関する割込みと OS に関する割込みを引き受けるソフトウェアを OS や DBMS の一段上に置く，一種の割込みハンドラを提案している（これを hypervisor と呼んでおり，Virtual Machine がこの概念に近い）[2]．また，Stonebraker は，この問題を解決するためには，OS が DBMS に関して特別のスケジューリングクラスを設定するしかないとしている[1]．

DBMS が，OS の一つのタスクとして，他の非データベースプログラムと同一のレベルにある間は，DBMS のスケジューラと OS のスケジューラの間の矛盾は解決しないように思える．SQL/DS のように，SQL/DS が稼働するパーティション（区画）を独立させ，そのなかにディスパッチャを置くことも考えられるが[8]，これも結局は，ユーザ空間の中で稼働する OS を開発することと同等になり，性能の低下は避けられない．

3.2 OPT-R におけるスケジューリング法

OPT-R では，前節に述べた問題点を解決するために，2 章で述べた各タスクをすべて同等のレベルで管理している．すなわち，OS と DBMS のタスク階層に起因する性能低下の問題は存在しない．したがって，タスクスイッチにかかる負荷は，スーパバイザ割込みの共通処理とタスクディスパッチング処理時間の合計であり，ダイナミックステップ数で数百命令で済んでいる．さらに，タスク間の同期処理に使用する POST および WAIT のシステムプリミティブの処理ルーチン（SVC ルーチンとして構成されている．詳細は 4 章参照）自体のダイナミックステップ数は，おのおの百命令以下である．以上のように，タスクスイッチおよび同期が高速に行われることによって，システムの処理多重度が高められているといえる．

OPT-R のタスクスケジューリング（タスクディスパッチング）のアルゴリズムとしては，複雑なものは使用せずに，以下に述べるような非常に簡単なプライオリティスケジュールを採用している．これは，詳細なレベルでのスケジューリングを動的に行おうとすると，各タスクの文脈にまで立ち入らなければならず，非常に複雑になるということと，OPT-R では，一つのトランザクション内では，その文脈に依存した静的なスケジューリング（5 章参照）が可能であることによる．

各タスクには，それぞれ優先順位が与えられてお

り，実行待ち状態になると，当該タスクの TCB は優先順位に応じて RTQ (Ready Task Queue) へキューイングされる．システムは割込みが発生すると，割込みハンドラへ制御を移し，割込み要因の解析を行う．その後，割込みの種類に対応した処理を行い，制御はタスクディスパッチャへ移される．タスクディスパッチャは，RTQ にキューイングされている TCB のなかから，その時点で最も優先順位の高いものを捜した後，当該タスクへ制御を移す．さらに，同一優先順位のタスク群に関しては，FIFO (First In First Out) スケジューリングを行っている．

以上のように，OPT-R では，従来の OS と同様のスケジューリング機構をサポートしている．OPT-R では，これを大域的スケジューリング (global scheduling) と呼んでおり，ユーザ管理用のタスクやトランザクション処理用のタスクを同等のレベルでスケジューリングの対象としている．

4. タスク制御方式

前章で述べたように，各タスクは，タスクディスパッチャによりプライオリティ/FIFO スケジュールによって管理されるため，各タスクは並行して処理を行うことになる．したがって，各タスク間の連絡機能等のタスク制御機能が必要となる．本章では，各タスクを制御するために用意しているプリミティブについて説明する．

4.1 タスクの生成と消滅

同時に存在しうるタスク数が増加すると，タスクスイッチの処理による負荷が増大し，システムの性能低下につながる．したがって，OPT-R ではタスクの生成と消滅を動的に行うことを可能としている．

タスクの生成は，システムプリミティブの ATTACH マクロにより SVC 処理を介して行われる．ATTACH の処理は，使用されていない TCB を一つ取り出し，その TCB を RTQ へキューイングすることによって実現される．

タスクの消滅は，同様に，DETACH マクロにより行われる．DETACH 処理は，当該タスクの TCB を RTQ から切り離すことによって実現される．

4.2 タスクの同期と交信

OPT-R のタスク間の同期と交信は，事象制御ブロック (ECB : Event Control Block) を介して，セマフォア (semaphore) を実現するシステムプリミティブ POST および WAIT により行われる．

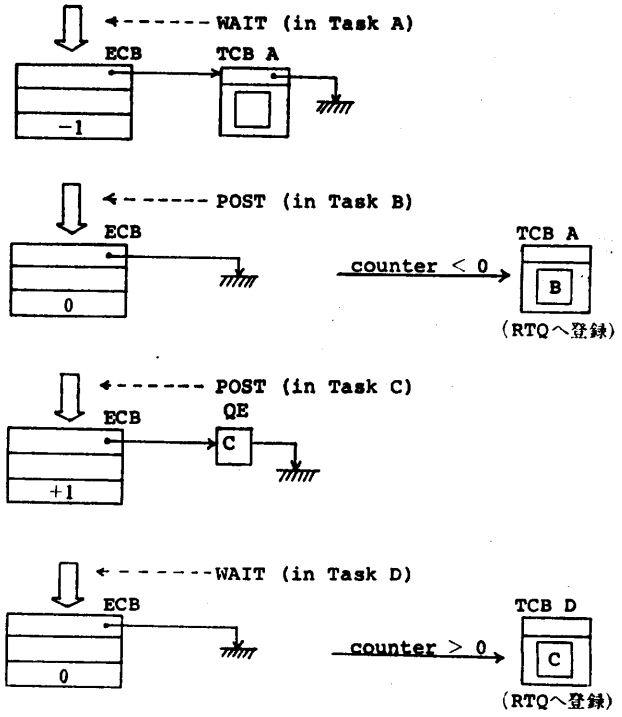


図 1 タスクの同期の例
Fig. 1 Example of task synchronization.

あるタスクから POST が発行されると，ECB のカウンタに 1 が加えられ，QE (Queue Element) が一つ ECB へキューイングされる．また，WAIT が発行されると，当該タスクの TCB が ECB へキューイングされ，カウンタから 1 が減じられる．ただし，ECB のカウンタが正のときに WAIT を発行すると，当該タスクの TCB が RTQ へ登録され，実行可能状態となる．また，ECB のカウンタが負のときに POST を発行すると，ECB にキューイングされていた TCB が一つ取り出され RTQ へ登録される．以上の操作を**図 1** に示す．

4.3 ユーザ要求管理用タスク群の同期処理

OPT-R では，システム初期化時に，2 章で述べたシステム管理用タスク群およびユーザ要求管理用タスク群のうちの TIT, DOT を ATTACH し実行待ち状態にする．各タスクは，アイドルタスクを除いて，システムテーブル領域内に設定されている各 ECB に対して WAIT を発行し，何らかの事象が発生するのを待つ．システム管理用タスクに関しては，処理要求が生じた際に，この ECB に対して POST を発行すれば，当該タスクがスケジュールされる．

ユーザ要求管理用タスクの標準的な同期処理は，以

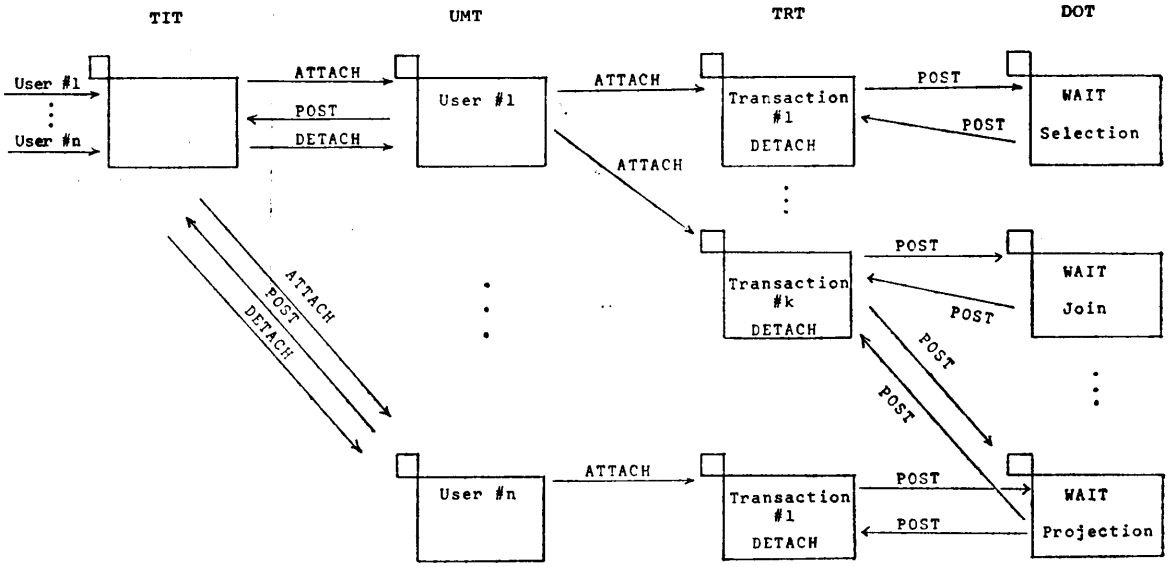


図 2　ユーザ要求管理用タスクの同期処理
Fig. 2　Processings of synchronization for the control tasks of user's requests.

下のように行われる（**図 2** 参照）.

TIT では,ユーザのブレーク割込みがあり LOGON コマンドが投入されると，セッション開始の正当性をチェックした後，UMT を ATTACH する．UMT では，端末からのユーザの要求を処理するが，トランザクションに関係しないコマンドは，このタスクで即時処理される．一方，LOGOFF コマンドが投入されると，UMT は TIT へ POST を発行して当該ユーザのセッション終了を伝える．TIT は，当該ユーザに対応する UMT を DETACH し，セッションを終了させる．

ユーザがトランザクション処理を要求すると，UMT は，トランザクションに対応して TRT を一つ ATTACH する．TRT では，データ操作系列に基づいて，各データ操作に対応した DOT へ POST を発行して処理要求を伝える．DOT では，その際に渡されるパラメータをもとにデータ操作を行った後，当該 TRT へ POST を発行し処理の終了を伝える．TRT と DOT 間で，この同期処理を繰り返して結果が得られると，当該 TRT は自分自身を DETACH する．

以上のような同期処理と，前章に述べたタスクの優先順位によるスケジューリングによって，以下の利点が生まれる．

（1）　ユーザのシステムに対する要求は，ただちに受け付けられる．とくに，トランザクションに関係しないコマンドは即時処理される．

（2）　トランザクション処理は，UMT から離れてバッチ処理的に行われるので，CPU の利用率が上がる．

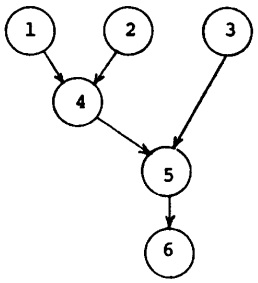
（3）　ユーザによるトランザクションの取消しは，UMT が当該トランザクションに対応する TRT を DETACH することにより，簡単に実現できる．

（4）　各処理単位に一つのタスクを割り当てる方式をとっているので，システムの拡張が容易である．たとえば，ユーザ数，トランザクション数の拡張，新しいデータ操作の追加は，他モジュールに影響を及ぼすことなく簡単に実現できる．

5. 局所的トランザクションスケジューリング

マルチプログラミングの理論は，マルチプロセッサシステムに関するものが多いが，CPU バウンドと I/O バウンドの分離可能な単一プロセッサシステムにおいても，処理のオーバラップは可能である[3),4)]．DBMS においては，マルチプログラミングの概念は，異なるトランザクションの間のオーバラップを導入するときに適用可能となるが（これについては，3 章で述べた），一つのトランザクション処理のなかでの十分なオーバラップを導入するときにも適用可能である．これは，OPT-R を実現する小型計算機においてとくに重要である．小型計算機では，主メモリ，入出力チャネル，周辺装置が制限されているので，複数のトランザクションが同時に実行される際のメモリスワ

ッピング等のオーバヘッドが避けがたい．したがって，性能を向上させるためには，おのおの参照の逐次性[5),6)]の存在する異なったトランザクション間で処理のオーバラップを実現することによって，システム全体として参照の局所性を向上させることはもとより，一つのトランザクション内で演算と入出力処理のオーバラップを最大にして，システムの処理多重度を上げることが考えられる．具体的には，2章で述べた TRT を，DOT への POST-WAIT 系列へ変換することによって実現される．これは，各 DOT が，データベースへのアクセスとアクセスしたデータベースに対する演算の処理を含んでいるので可能となる．すなわち，互いに独立な DOT へ POST を発行することは，暗に，CPU バウンドの処理と I/O バウンドの処理をオーバラップすることになるからである．



(a) 有向グラフ
(a) Directed graph

vertex \ vertex	1	2	3	4	5	6
1	0	0	0	1	0	0
2	0	0	0	1	0	0
3	0	0	0	0	1	0
4	0	0	0	0	1	0
5	0	0	0	0	0	1
6	0	0	0	0	0	0

(b) 隣接行列
(b) Adjacency matrix

図 3 関係操作系列の有向グラフと隣接行列
Fig. 3 Directed graph and adjacency matrix for the sequence of relational operations.

本章では，以上のような観点から，1トランザクション内の並行処理可能な処理単位の識別と，それに基づくトランザクションのスケジューリングアルゴリズムについて述べる．

5.1 並行処理可能なタスクの識別アルゴリズム

ユーザの要求したトランザクションは，構文解析および意味解析によって関係操作の木へ変換される（木の各ノードは DOT に対応する）．本節では，この段階から，以下の関係操作系列を用いて，並行処理可能なタスクの識別アルゴリズムについて説明する．

1 $T_1 \leftarrow R[R_1\ \theta\ C_1]$ selection
2 $T_2 \leftarrow S[S_1\ \theta\ C_2]$ selection
3 $T_3 \leftarrow U[U_1\ \theta\ U_2]$ restriction
4 $T_4 \leftarrow T_1[R_1\ \theta\ S_2]T_2$ join
5 $T_5 \leftarrow T_4[R_1\ \theta\ U_1]T_3$ join
6 $T_6 \leftarrow T_5[R_1, R_2, U_2, U_3]$ projection

ただし，$R=R(R_1, R_2, R_3)$，$S=S(S_1, S_2)$，$U=U(U_1, U_2, U_3)$ は関係を表す．さらに，θ は比較演算子，C_1 および C_2 は定数を，$T_i(1 \leqslant i \leqslant 6)$ は一時的関係を表す．

さて，上記の系列を各関係操作を一つの頂点とし，その実行順序の依存関係を辺として有向グラフに表現すると**図3**(a)のようになる．これは，さらに，図3(b)のような隣接行列として表現される．並行処理可能なタスクの識別は，この隣接行列をもとに，当該集合内では互いに実行順序の依存関係がないタスク（すなわち DOT）の集合族を求めることと同等である．この過程は，図3(b)の隣接行列を，すべての要素が零である行列と，それ以外の行列に分割することよりなる．分割は，以下の2種類が考えられる（**図4**参照）．

(1) 列 分 割

隣接行列より，列要素がすべて零の列に対応する頂点の集合を並行処理可能なタスクの集合とする．次に，これらの頂点に対応する行と列を除去し，残りの頂点に関する行列で同様の操作を繰り返す．

(2) 行 分 割

列分割処理と同様の操作を行を単位として行う．列分割は，有向グラフの有向道をたどることに対応し，時間的に早く実行されるタスクの集合から決定される．行分割は，逆有向グラフの有向道をたどることに対応し，時間的に遅く実行されるタスクの集合から決定される．図4に示すように，例として使用した系列の分解（すなわちスケジュール）は，

列分割：{1, 2, 3}→{4}→{5}→{6}
行分割：{1, 2}→{3, 4}→{5}→{6}

となり，列分割と行分割は一般に一致しない．

5.2 局所的スケジューリング

OPT-R では，各演算単位にタスクを割り当て（すなわち，DOT として構成し），前節に述べたアルゴリズムを用いて，1トランザクション内での局所的なスケジューリングを行っている．すなわち，前節のアルゴリズムによって得られる集合族の時間的順序関係に着目して，当該集合内では実行順序の依存関係がないことにより，同一集合に属する DOT へ一度に POST

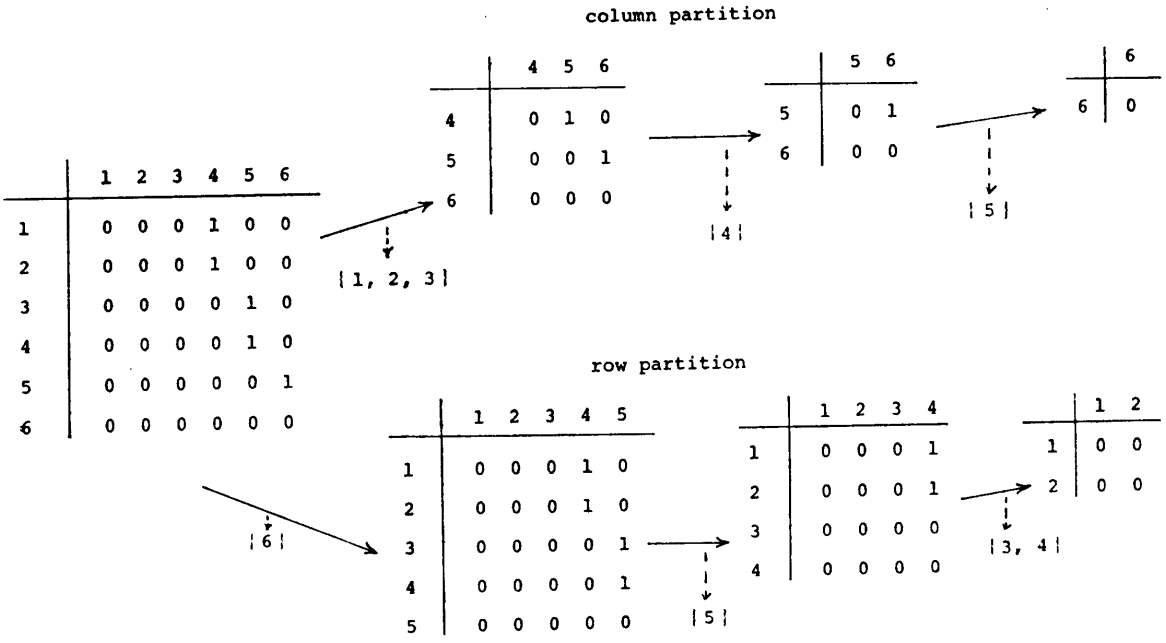


図 4　列分割と行分割
Fig. 4　Column partition and row partition.

を発行し，各 DOT に起動をかける．その後，DOT の処理終了を待つために WAIT を発行する．こうして，各 DOT は実行可能状態となる．DOT は，処理すべきカタログやデータベース等へのアクセスを行い，演算を実行する．したがって，一つの DOT の入出力処理中には，他の DOT の演算が可能になり，逐次的に実行するよりもシステムの多重度が高まり CPU の利用率が上がることになる．本節では，以上の考えをもとに，OPT-R における実際のスケジューリングアルゴリズムについて述べる．

まず，スケジューリング自体のコストについて考察する．前節の例では，列分割から 3!=6 通り，行分割から 2!×2!=4 通りのスケジュールが得られる．したがって，このなかから最適なスケジュールを選択する必要がある．各頂点に対応するタスクは，I/O バウンド処理と CPU バウンド処理を含んでいるから，各集合内の要素に関して，この二つの処理のオーバラップが可能である．したがって，各頂点に，(CPU 時間，I/O 時間，使用メモリ容量) の情報を与え，空間-時間積 (space-time product) によって各スケジュールを評価し，最適スケジュールを得ることが可能である．しかし，一般的には，タスクの各集合内のオーバラップ系列の組合せは，多項式のオーダを越えてしまい，スケジューリング自体の処理コストが無視できない．したがって，OPT-R では，基本的には頂点の番号の小さい順にスケジュールを生成することとした．種々のシミュレーションの結果から，逐次的に処理する場合に比較して処理時間が短縮されることがわかっている．

さて，OPT-R の局所的スケジューリングアルゴリズムは，以下のステップよりなる（ただし，i の初期値は 1 とする）．

STEP 1：前節のアルゴリズムによって（列分割を使用して），並行処理可能なタスクの集合系列を求める．

STEP 2：列分割された第 i 番目の集合の操作に対応する DOT へ POST を発行する．

STEP 3：第 $i+1$ 番目の集合に関して，頂点の番号順にその操作を実行するために，第 1 番目から第 i 番目までの集合のうちのどの操作の終了が必要かを隣接行列より求める（これは，その頂点の番号に対応した列要素の中で '1' になっている行番号に対応した頂点の集合である）．これらの操作に対応する DOT に対して WAIT を発行した後，第 $i+1$ 番目の集合の操作に対応する DOT へ POST を発行する．

以上のステップを集合系列の最後の集合まで繰り返すことによって，POST-WAIT 系列が得られる．

5.3　スケジューリングの評価

トランザクション内の処理多重度を最大にするためには，それぞれの DOT に対する POST と WAIT

の間の距離（同一タスクに関する POST と WAIT の間にある他タスクの POST および WAIT の総数）を最大にすることが考えられる．これは，CPU 時間等の workload に関するパラメータが各 DOT とも同程度であることを仮定した場合のことである．しかし，前節でも述べたように，スケジューリング自体の計算量が問題であるから，これで十分であるといえよう．前節まで使用した例では，以下のようになる（ただし，POST を P，WAIT を W で表す）．

（1） 列分割によるスケジュール

P 1→P 2→P 3→W 1→W 2→W 3→P 4→W 4→P 5→W 5→P 6→W 6，PW 距離＝6

（2） 行分割によるスケジュール

P 1→P 2→W 1→W 2→P 3→P 4→W 3→W 4→P 5→W 5→P 6→W 6，PW 距離＝4

（3） OPT-R のスケジュール

P 1→P 2→P 3→W 1→W 2→P 4→W 3→W 4→P 5→W 5→P 6→W 6，PW 距離＝8

これら三つのスケジュールのなかで，前節に示した OPT-R のスケジュールの POST-WAIT 距離が最大となる．

さらに，関係操作に関するシミュレーションによると，選択，制約操作に比較して射影操作は時間がかかり，後で行うほうがよいことがわかっている．すなわち，ユーザの要求の最適化によって，時間のかかる操作は後で行うように置き換えられるから，並行して処理するタスク数は時間的に早い段階に多く，遅い段階で少ないほうが望ましい．この意味からも，上記アルゴリズムがトランザクションのスケジューリングに適合するといえよう．

5.4 実 現 法

トランザクション管理モジュールは，前節のスケジューリングで得られた POST-WAIT 系列を機械語コードとして生成し，それを一つの TRT として ATTACH する．しかし，POST-WAIT 系列はその都度スケジュールに対応して生成されるのではなく，テンプレートとよばれる共通コードを呼び出す形で実現される．このテンプレートは，POST 発行用コード，WAIT 発行用コード，ワークエリア解放用コード，TRT 消滅用コード等からなる．したがって，TRT は DOT に渡すパラメータのアドレスの設定，DOT との同期に必要な ECB のアドレスの設定，テンプレートを呼び出す命令の3命令の繰り返し現れる機械語に翻訳され，UMT がこの翻訳された TRT を ATTACH してトランザクションの処理が開始される．

6. おわりに

本論文では，OPT-R のタスク管理機構について述べてきた．従来の DBMS で問題となっているタスクスケジューリングは，データベース専用の一貫したスケジューリングを行うことにより解決されると考えている．さらに，従来考えられていなかったトランザクションのスケジューリングについても新しいアルゴリズムを提案し，その実現法を示した．これらアルゴリズムにおける課題としては，以下のものが考えられる．

（1） 各関係演算単位で，CPU 時間，I/O 時間，メモリ使用量を実行の前に算定し（文献7）に見られる），各資源に関する最適なスケジュールを生成する方法．

（2） 各関係演算内で，CPU バウンドと I/O バウンドの処理をオーバラップさせる方法．

（3） 本論文で考えた方法は，入出力処理が各タスクで一つの場合のスケジューリングに関するものであったが，実際の場合は，独立に複数存在すると考えられる．この場合，入出力処理相互でオーバラップ系列を生成することが可能であり，したがって，データベースの2次記憶への配置も問題となる．

参 考 文 献

1) Stonebraker, M.: Operating System Support for Database Management, *Comm. ACM*, Vol. 24, No. 7, pp. 412-418 (1981).
2) Gray, J.: Note on Data Base Operating Systems, in Goods, G. and Hartmanis, J. (eds.), *Lecture Notes in Computer Science*, pp. 393-481, Springer-Verlag, New York (1978).
3) Ramamoorthy, C. V. et al.: Scheduling Parallel Processable Tasks for a Uniprocessor, *IEEE Trans. Comput.*, Vol. C-25, No. 5, pp. 485-495 (1976).
4) Fernandez, E. B. and Lang, T.: Scheduling as a Graph Transformation, *IBM J. Res. Dev.*, Vol. 20, No. 6, pp. 551-559 (1976).
5) Smith, A. J.: Sequentiality and Prefetching in Database System, *ACM TODS*, Vol. 3, No. 3, pp. 223-247 (1978).
6) Rosell, J. R.: Empirical Data Reference Behavior in Data Base Systems, *Computer*, Vol. 9, No. 9, pp. 9-13 (1975).
7) Selinger, P. G. et al.: Access Path Selection

in a Relational Database Management System, Proc. ACM SIGMOD, pp. 23-34 (1979).
8) Chamberlin, D. D. et al.: A History of System R and SQL/Data System, Proc. Very Large Data Bases, pp. 456-464 (1981).
9) 大久保英嗣，津田孝夫：データベースオペレーティングシステム OPT-R の設計目標とアーキテクチャ，情報処理学会論文誌，Vol. 25, No. 4 pp. 535-543 (1984).

(昭和58年5月20日受付)
(昭和59年1月17日採録)