データベース・マシンの同時処理制御

A Concurrency Control in a Database Machine

水摩正行　牧野武則　梅村護　日吉茂樹
Masayuki MIZUMA　Takenori MAKINO　Mamoru UMEMURA　Shigeki HIYOSHI
日本電気（株）中央研究所
Central Research Lab. Nippon Electric Co., Ltd.

1. はじめに

データの共同利用を主目的とするデータベース管理システム（以下DBMSと略す）では、そのシステムが採用している同時処理制御方式が、システム性能、システムの使い易さに大きな影響を与える。

DBMSの制御方式に対して、種々の方式が提案され、実際に実現されてきた。同時処理制御の特徴は、制御の対象（ロック対象）と制御モードに現われる。システム設計において、それらをどのように決定するかは、ロック対象への競合度，システムオーバヘッド，使い易さ、実現の容易性などを十分考慮したうえで行うべきである。

現存するDBMSの中で同時処理制御機能の高いシステムはシステムRである[1]．システムRでは、tuple，データアイテム値など、非常に細いレベルでの排他制御を行なっており、その制御モードもshared lock, exclusive lockなど任意に選べる。しかしながら、実用システムとして、このようなきめ細い制御機能を組み込むのは、それによってもたらされる効果と制御の複雑さ、オーバヘッド等を考慮すると必ずしも最適でない。

一方，DMS1100[2]，IDS[3]などでは、ロックの最小単位をページでおさえている．この方式は、システムRに比較して同時動作性、制御の柔軟性に劣るが制御が単純になる。しかしながら，ページに対するロックをソフトウェアにより実現しているのでオーバヘッドが問題である。

本資料では、後置プロセッサタイプのデータベースマシン（Generalized Database Subsystem；以下GDS[4]と略す）の同時処理制御の設計思想、実現方式について報告する。

GDSの同時処理制御はDMS1100のアプローチに近いが、次のような特徴をもっている。

（1）ユーザインタフェースレベルでは、二段階石（エリア，レコードオカレンス）の排他制御を採用している。しかし、レコードレベルの排他制御はシステム内部でページに置換している。

（2）ページの排他制御は、ロックオーバヘッドを減少させるため、ハードウェアによって実現されたアドレス変換機構を利用して行われる。

（3）デッドロックフリー方式を採用している．デッドロック検出機構はハードウェア化される。

以下、データベースのロック対象の大きさとシステム性能との関係を解析する。そして、その解析結果を参考にして設計されたGDSの同時処理制御方式について紹介する。

2. ロック単位の選択基準

DBMSの設計において、同時処理制御の最小単位をどの程度にするか、が重要な問題である。ロックの対象単位が大きくなればある程トランザクション間の競合が発生し、多重処理の効果が減少する。一方、ロックの最小単位を細かくすると、競合率は減少するが、時間的空間的なオーバヘッドが増大し、インプリメンテーションも困難になってくる。

ここで、ロック単位と競合性との関係を考察する。

はじめに、記号の定義を行う。

t; 平均実行トランザクション数

u_r; トランザクション当りの要求レコード数

d; データベース(DB)の総レコード数

g; DBブロック数(DBの分割数)

s; トランザクション当りの要求ブロック数

このとき、特定ブロックに i 個のトランザクションが競合する確率 C_i は、次の二項分布で表現される。

$$C_i = \binom{t}{i}\left(\frac{s}{g}\right)^i\left(1-\frac{s}{g}\right)^{t-i} \qquad (1)$$

平均専有ブロック数 B_c は、

$$B_c = \sum_{i=1}^{t} g\cdot C_i = g(1-C_0) = g\left(1-\left(1-\frac{s}{g}\right)^t\right) \qquad (2)$$

平均実行可能トランザクション $\tilde{t}$ は、

$$\tilde{t} = (B_c/ts)t = B_c/s \qquad (3)$$

となる。

図1(実線)に g, s, $\tilde{t}$ の関係を示した。ここでは、$t=10$, $u_r=50$ を仮定している。u_r と s の関係は、一般にトランザクションのデータ参照特性(ローカリティ特性)に影響される。ここでは、次の3ケースについて示した。

$s=5g$; 完全離散

$s=g\left(1-\left(\frac{g-1}{g}\right)^{u_r}\right)$; ランダム

$s=10$; ローカリティがある程度存在

また、Closed Queuing Network Model[5]を用いて、ロックによりシステム性能がどの程度低下するかを調べた。その結果を図1(破線)に示している。なお、システムのモデルおよび値の設定は付録に示す。

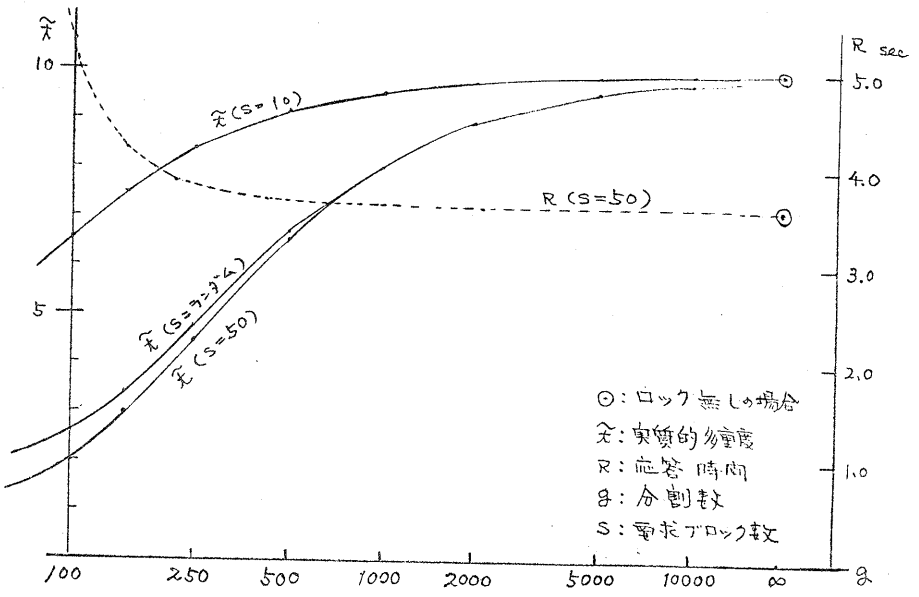


図1. ロック単位と競合性の変化

図から、DBの分割数が　500　以上のオーダにはれば、ロック競合による実質的多重度の低下はそれ程認められないことが判る。

一般に、DBのロックの対象候補として、DB全体、エリア、レコードクラス、レコードオカレンス、データアイテム値，ページなどが考えられるが、前記の解析結果より、大規模DBを取り扱うDBMSでは、ロックの最小単位はページで十分であると云えよう。

上記の背景をもとに、GDSでは、ロックの最小単位をページと定めシステム設計を行なった。以下、GDSの同時処理制御について紹介する。

3. データベースの同時処理制御

GDSの同時処理制御は、ユーザが陽に要求を出すものと、システムが自動的に行うものがある。

前者は、ユーザのプログラムロジックが他のユーザの更新によって乱されるのを防ぐために行われる。後者は、DB上での矛盾を防ぐために行われる。矛盾が生じる主な原因は、複数ユーザが同時に出す更新要求の干渉、および更新処理の途中でのトランザクションの撤回である。

3.1　ユーザインタフェース　レベルでの保護

ユーザインタフェース　レベルでデータの一貫性（Consistency）を保障するため、GDSでは、エリアレベルの保護とレコードレベルでの保護機能を提供している。

(1)　エリアの保護

エリアはCODASYL案[6]のエリアの概念に相当する。一般にエリアに含まれるデータ量は非常に多い。このレベルでは、トランザクション間の競合を緩和させるために柔軟な排他制御モードが必要になる。

GDSでは、CODASYL仕様に従った排他制御モードを全てサポートしている。

エリアに対する排他制御の宣言は、OPEN／CLOSE命令によって行われる。ユーザはこの時、エリアの使用モード（RETRIEVE/UPDATE）と排他制御モード（EXCLUSIVE/PROTECTED/SHARED）の宣言を行う。表1に同時処理の規則を示す。

表1．エリアの排他制御規則

	S.R	S.U	P.R	P.U	EX
S.R	○	○	○	○	×
S.U	○	○	×	×	×
P.R	○	×	○	×	×
P.U	○	×	×	×	×
EX	×	×	×	×	×

○：実行可　×：不可

S（Shared）：エリアの共有使用

P（Protected）：検索処理ユーザに対しては許されるが更新処理ユーザは共用不可．

EX（Exclusive）：完全排他使用

R（Retrive）：検索処理

U（Update）：更新処理

なお、GDSでは、エリアレベルでの排他制御によってシステムがデッドロック状態に陥るのを防ぐため、トランザクションはエリアの使用を一括要求するようにしている。

(2)　レコードオカレンスの保護

2章の解析結果から判るように、レコードオカレンス単位ではトランザクション間の競合は余り生じない。そこで、GDSでは、このレベルでの排他制御は共用モード（Shared）と専有モード（Locle）に限定している。

ロック要求は次のようにして行われる。GDSでは、種々のアクセスパス（PRIMARYパス，IMAGEパス，LINKパス）に従って、DB処理を行うことができるが、ユーザはその処理に先立ってアクセスパスの生成を要求する[4]。この時、必要があればロック要求を指定する。ロック要求が指定された場合、アクセスパスのカレントレコードは自動的にロックされる。カレントレコードが切り替わると旧カレントレコードはUNLOCKされ、新カレントレコードがLOCKされる。カレントレコードが切り替わった後に旧レコードをロックしたい場合には

LOCK命令を使って陽に指定できる。

3.2 システムレベルでの保護

GDSでは、1レコードアクセスを基本としているため、DBの更新は複数の命令を使って手続き的に行われる。更新処理が完全に終了するまでの間に、他のトランザクションの更新が発生した場合DBの論理的な矛盾が発生する。また、更新の途中でトランザクションが撤回される可能性もある。このような状況からDBを守るため、DBレコードの更新されたページは更新トランザクションが終了するまで、GDSが自動的にロックをかける。

また、DBの更新に伴なって、インデックスやディスクリプタ等のシステム制御情報の修正が必要になってくる。これらの制御情報に対しては、GDSが必要に応じて該当ページを一時的にロックする。

4. システム構成

GDSの多重処理制御は、1つのトランザクションにつき、1つのGDIコマンド(Generalized Database Interface)のみ実行する方式を採用している。

以下、多重処理制御機能に注目し、GDSのシステム構成を説明する。

GDSは処理の形態に応じて、いくつかの機能モジュール(FM)に分割されている。図2はその構成を示したものである。

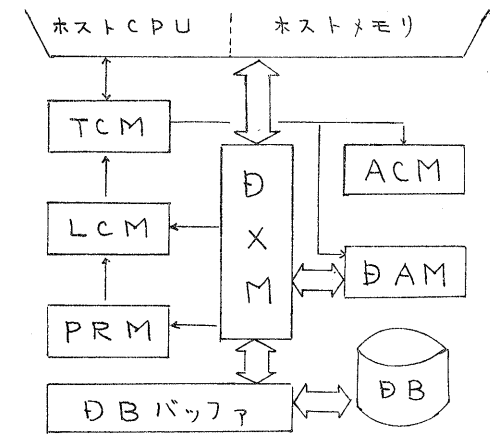
(1) トランザクション コントロール モジュール(TCM)

TCMはホストプロセッサとの通信、およびトランザクション処理のスケジューリングを行う。はじめに、START TRANS 命令を受けとると、トランザクションIDを作成する。トランザクションの動きは、トランザクションコントロールブロック(TCB)によって管理される。TCMは汎用計算機のそれに比べ単純化されており、少いオーバヘッドで制御できる。

多重処理の制御は、3章で示したDBの同時処理制御イベント、IO処理イベント、ホストからのCALL GDI 命令を契機に行われる。図3は、GDS内部でのトランザクションの制御状態の動きを示している。

(2) エリア コントロール モジュール

GDSに認識されたトランザクションは、最初にエリアの使用宣言を行う。ACMはトランザクションが要求した使用モードと排他制御モードを現在のエリアの使用状態と付き合わせて、トランザクションのキュー管理を行う。



ACM: Area Control Module
DAM: Data Access Module
DXM: Data Transfer Module
LCM: Lock Control Module
TCM: Transaction Control Module
PRM: Page Replacement Module

⇒ データフロー
⟶ 制御フロー

図2. システム構成

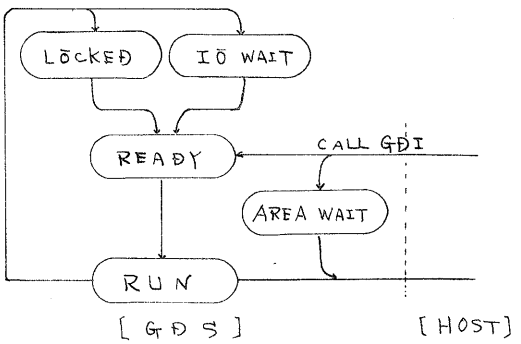


図3 制御状態の変動

（3） データ転送モジュール（ＤＸＭ）

ＤＸＭは、ＧＤＳの中心的な存在で、ＤＢバッファと、他のＦＭのワークメモリ、ホストプロセッサの主メモリとの間のデータ転送を行う。データ転送はレコードイメージで直接行われる。データ転送の指定パラメータは、論理レコードアドレス（エリア番号，ページ番号，ライン番号），レコード内オフセットと長さ，で表現される。このデータ転送中にロックのチェックを行い、他のトランザクションによってロックされている場合にはＬＣＭに競合指令を発する。

（4） ロック コントロール
モジュール（ＬＣＭ）

ＬＣＭは、ＤＸＭが確認したデータ競合に従って、トランザクションのキュー管理を行う。また、キュー管理と同時にデッドロック状態の検出を行う。

（5） ページ リプレースメント
コントロール モジュール（ＰＲＭ）

ＤＢバッファ内のページ管理を行う。即ち、ＤＸＭがデータ参照時にミッシングフォルトを確認した場合、要求ページを二次記憶より転送するため、ページの置き換え制御を行う。置き換え制御では、ロックされたページは出来る限り追い出さないようにしている。ロックされたページが追い出される場合には、ロック情報をセーブするためＬＣＭに制御を渡す。

（6） データ アクセス
モジュール（ＤＡＭ）

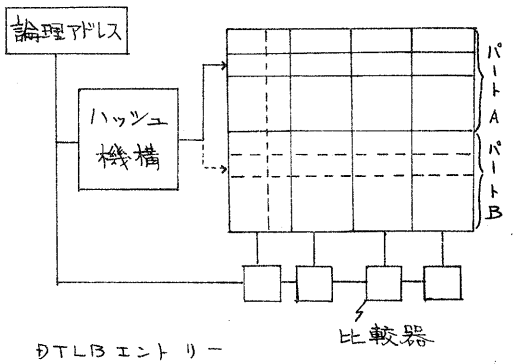
ＧＤＳがサポートしている各種のアクセスパスに従って、レコードの検索、更新処理を実行する。

5. アドレス変換機構とロック

レコード，ページレベルの排他制御は従来ソフトウェアによって実現されていた。このため、ロック制御のためのオーバヘッドが多大であった．また、そのオーバヘッドを無視することができないため、参照データに関係する制御情報を更新する場合（たとえば、カレンシーインジケータの修正時など）のみ、アクセス权のチェックをする、というような方式で予め設計時に一定の規則を定めその規則に従ってチェックを行なっていた。しかしながら、このような方式では、正規の手続きを経ないデータ参照に対してアクセス違反を摘発することができない。

ＧＤＳでは、このレベルの排他制御を行なうために、ハードウェアによって実現されるアドレス変換機構を有効に利用している。ＤＢデータを参照する場合には、常にこのアドレス変換機構を介するので、ロック機能を組み込めば、常時アクセス权のチェックを行うことができ、より強力なロック制御機能を実現できる。また、ロックオーバヘッドも無視できる。なお、レコードに対する排他制御は、競合が少いため、すべてページで代用している。

アドレス変換機構はＤＸＭの１コンポーネントであり、論理アドレスをＩＯバッファ上の物理アドレスに変換する。図４にアドレス変換機構の構成を示す。



Ｔｉｄ；トランザクション ＩＤ
Ｌ.ｂ ； ロックビット
Ｌ.ｃ ； ロック カウント

図4. アドレス変換機構（ＤＴＬＢ）

高性能なアドレス変換機能を実現する場合には、連想メモリ素子の使用が望まれるが、低価格で実現するためハッシュ機構と比較器を組み合わせた疑似連想記憶方式で実現される（ＤＴＬＢと呼ぶ）。

ＤＴＬＢの各エントリーには、アドレス変換情報の他に、ロック制御情報やバッファ管理情報が含まれている。

他のＦＭからデータベースデータの参照要求が発生した場合、このアドレス変換機構を通して、目的データページの物理アドレスを決定すると同時に、データに対するアクセス権のチェックを行う。

ロック制御情報はロックビット，トランザクションＩＤ，ロックカウントから成っている。

ロックビットは対応するページのロック状態を示す。もしロックビットがＯＮであれば、他のトランザクションが対象ページをロックしていることを示す。そのトランザクション名はトランザクションＩＤフィールドに示されている。この場合データ参照はトランザクションＩＤフィールドに一致するトランザクションのみ許可される。ロックビットがＯＦＦの場合には、任意のトランザクションからのデータ参照が許可される。

ロックカウントはロック中のトランザクションが対象ページに要求を出したロックの回数とアンロックの回数の差を示す。対象ページをロックしている間はその値は正である。ロックカウントが０になった場合ロックビットがＯＦＦになりページが解放される。このロックカウントは次の２つの目的で設けられた。

（１）ＧＤＳでは、レコードに対するロックをページで代用している。ページには一般に複数のレコードが存在し、ユーザはそのいくつかのレコードに対しロック要求を出す可能性がある。この時、ユーザが対象ページに含まれているロック中のすべてのレコードをアンロックするまでそのページを解放してはならない。

（２）一般にシステムを構成する場合 E.W. Dijkstra 等が提唱しているように階層設計がなされる。これは他の階層に対する余分な知識を不要にするためである。このような設計においては、各階層で同一オブジェクトに対し異なった目的でロックをかける場合が発生する。このように、目的に応じて自由にロック，アンロックをかけ得るようにすれば設計が楽になる。

上記の状況において、システムがページに対するロックの解放のタイミングをつかむためにロックカウントが設けられた。

6. デッドロック検出機構

デッドロックを防ぐ１つの方法として、資源を一括要求させる方式がある。エリア単位の排他制御を行うため、ＧＤＳでは、この方式を採用している。しかしながら、レコードオカレンスレベルの排他制御でこの方式を採用した場合には、ユーザの負担が大きくなり問題である。

このため、ＧＤＳでは、レコードレベルでの排他制御はデッドロックフリー方式を採用している。

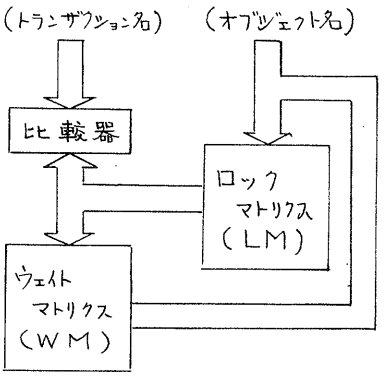


図５　デッドロック検出機構

システムのデッドロック状態を検出するためハードウェアにより実現されるデッドロック検出機構を備える。図５はその構成を示したものである。

デッドロック検出機構は、ロックマトリックス（ＬＭ）とウェイトマトリックス（ＷＭ）から成っている。ＬＭの各要素は、ロック中のトランザクションとオブジェクトとの関係を表わしている。ＷＭの各要素はオブジェクトを待っ

ているトランザクションの状態を示している。

5章で記述したアドレス変換機構により、あるトランザクションが目的ページのロック待ちイベントを確認すると本装置に通知される。

デッドロック検出機構は、ロック待ちトランザクション名とロックオブジェクト名を入力パラメータとして受け取る。最初に、LMより対応するオブジェクトをロックしているトランザクション名を検出する。そして、そのトランザクション名をWMに入力することによって待ち状態にあるオブジェクト名が抽出される。そのオブジェクト名を再度LMに入力させ、その出力が要求トランザクション名（入力パラメータのトランザクション名）と一致した場合にデッドロック状態が発生したとみなす。一致しなければ、同じ動作を続ける。そして、最後まで一致するトランザクションが検出できなければ、デッドロック状態が発生していないこととなる。

7. おわりに

DBMSの設計において、DBの保護の最小単位をどの程度に設定すればよいか、は興味深い問題であろう。

本資料では、この問題を解析的に求め、ページを最小単位とすれば十分であることを示した。

また、データベースマシンの同時処理制御方式について論じた。

本データベースマシンはメモリ共用型の後置プロセッサである。データベースの保護はページを基本単位としている。更に、核機能をハードウェア／ファームウェアにより実現することによって、オーバヘッドの減少をはかっている。

本方式の有効性を確かめるために、現在実験システムを開発中である。

最後に、本研究の機会を与えて下さった日本電気中央研究所コンピュータシステム研究部藤野部長，祢津課長、ならびに、日頃直接御指導頂いている箱崎主任，久保主任に感謝致します。

《参考文献》

1) M.M. Astrahan etal : System R: A relational approach to database management, ACM TODS, Vol.1, No.2 (1976).
2) UNIVAC 1100 Series, Data Management System (DMS 1100) Data Manipulation Programmer Reference, UP-7908.
3) ACOS-6 データ管理 統合データベース（IDS）説明書.
4) K. Hakogaki etal : A Conceptual Design of A Generalized Database Subsystem, 3rd International Conf. on VLDB, oct.'77.
5) W.J. Gorden and G.F. Newell : Closed Queuing Systems with Exponential Servers, Operations Research, Vol 15, No.1-3, 1967.
6) CODASYL Data Description Language Journal of Development, Jun.'73, NBS Handbook (1974).

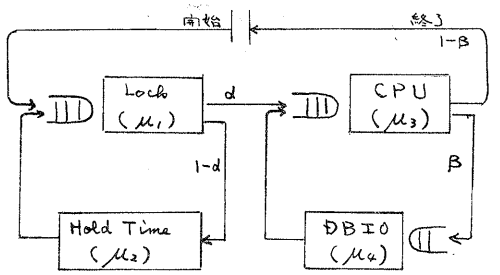
[付録] システム評価モデル

Closed Queuing Network モデルでは、次のことを前提にしている。

- 各サーバのサービス時間：指数分布に従う。
- サーバ間の遷移：遷移確率に従う random walk
- システム内のトランザクションの数は一定である

上記の前提をもとに、システム構成，各サーバの平均サービス時間，サーバ間の遷移確率，システム内のトランザクションの数 を入力パラメータとして、各サーバの平均使用率，平均待ち行列の長さ，システムレーテンタイム（応答時間）を計算できる。

Closed Queuing Network モデルを使って、DBの分割数の変化によってシステム性能がどのように影響されるかを近似的に求めるため、図A-1に示すようなシステム構成を仮定した。



図A-1. システム構成モデル

図において、

μ_1：ロックオーバヘッド時間．本来CPU処理であるが別サーバとした。

μ_2：ロック競合によって待たされる時間

μ_3：CPUによる1回当りのサービス時間

μ_4：DB入出力時間．DBIOは複数デバイスの指定可能

α, β：サーバ間の遷移確率

を表わす。

入力パラメータは、次のようにして決めた。μ_3, βはトランザクションのDB処理特性を反映させるもので、ある保険会社のオンライン・データベースシステムのデータを参考にして設定した。本システムでは、トランザクション当りのCPU処理量の平均は320 msecであり、平均IO回数は25.5回であった（IO時間：40 msec）。この例では、CPU処理/2 msecに1回の割合でIOが発生することになる。

これから

$\mu_3 = 12$ msec

$\beta = 25.5/(25.5+1) = 0.96$

がおまる。

システム内の全トランザクションの数は10に固定した。

αは2章の（3）式とシステム内全トランザクションの関係から求めた。（$\alpha = \bar{n}/10$）

μ_1はロックオーバヘッドを表わすもので、固定的に2 msecとした。

Hold Time（μ_2）は、トランザクションの競合による待ち時間を近似的に表わすため、マルチサーバ（待ちが生じないように20（>10）台）を仮定した。

μ_3の値を相続くトランザクションの処理の終了間隔と対応させるため、CPUとDBIOのみからなる別のQueuing Network Modelを作り多重度$\bar{n}$のときの応答時間を計算した。そして、その応答時間を$\bar{n}$で割った値をμ_3として与えた。

表A-1. 入力パラメータの設定値

入力パラメータ	値
多重度	10
μ_2	（多重度$\bar{n}$のときのCPU, DBIO系のみでの応答時間/$\bar{n}$
μ_1	2 msec （サーバ20）
μ_3	12 msec
μ_4	40 msec （サーバ5）
α	$\bar{n}/10$
β	0.96
S	50

なお、本モデルでは、各トランザクションは処理に関係するすべてのリソース（ブロック）をトランザクション処理の開始時に確保し、終了時に解放することを仮定している。

結果は、図1の破線で示している。