汎用ＤＢＭＳの評価方法について

中村史朗、吉田郁三，近藤秀文（日立システム開発研究所）
葛西靖三（日立中央研究所）

1. はじめに

データ・ベース・システム（以後ＤＢシステムと略す）は、特に、その稼動環境がオンライン使用の場合、性能の不足が直ちに、システム全体の有用性の欠如に結びつくという危険を有している。従って、オンラインＤＢシステムを計画する際には、システムの設置に先立つ性能評価が充分になされていることが必要である。従来、実用に供されてきた性能評価の手段は、モニタリング，モデリングに大別されるが、前者は、対象システムが稼動状態にあることを前提とし、その適用範囲としては、むしろシステム性能の tune-up に適していると考えられる。これに対して、後者は、

・事前の性能予測に使用可能なこと（妥当な精度のもとで）
・まだインプリメントされていない、新しい方式に対して実験ができること

の２点で、他の技法にはない特色を持っている。

本報告では、ＤＢシステムの性能評価を目的として開発された、大規模シミュレーション・モデルに関して、

・モデルの概略
・シミュレーション結果の例
・ハードウェア・モニタを使用した、モデルの妥当性検証

について述べる。

モデルの記述言語としては、ＤＢシステムの評価と同様、オペレーティング・システム（ＯＳ）、データ・ベース・マネジメント・システム（ＤＢＭＳ）の方式検討にも、使用できるように、Event－Driven 型のコンピュータ専用シミュレータを用いた。このシュミレータは、ＯＳ、ＤＢＭＳおよびアプリケーション・プログラムの動作を逐次シミュレートする。モデルは、ハードウェア構成およびソフトウェアの方式検討にも使用するため、非常に詳細に作成されている。

本モデルを使用する場合には、ハードウェア構成ならびに、アプリケーション環境（データ・ベース定義など）の定義と、アプリケーション・プログラムのモデル化が必要である。一たん、モデルが完成されれば、多種類のケースのシミュレーションが、単にパラメータを変更するだけで行なえる（メッセージ・トラフィック、データ・ベース・バッファ・プール・サイズ、アプリケーション・プログラム用タスク数 etc.）。

2、シュミレーション・モデル

モデルは、定義部と処理部の２つの部分から成っている。定義部は、シミュレーション対象のシステムの環境を表現する部分であり、処理部は シミュレータで提供された命令によって、ソフトウェア・システムを表現する部分である。シミュレータは、定義部の情報を参照しながら、処理部の命令をインタプリティブに実行する。各部分には、次のような情報が含まれる。

定義部

（1） ハードウェア定義

(2) アプリケーション定義(データ・ベース定義など)

処理部

(3) OSモデル

(4) DBMSモデル

(5) アプリケーション・プログラム・モデル

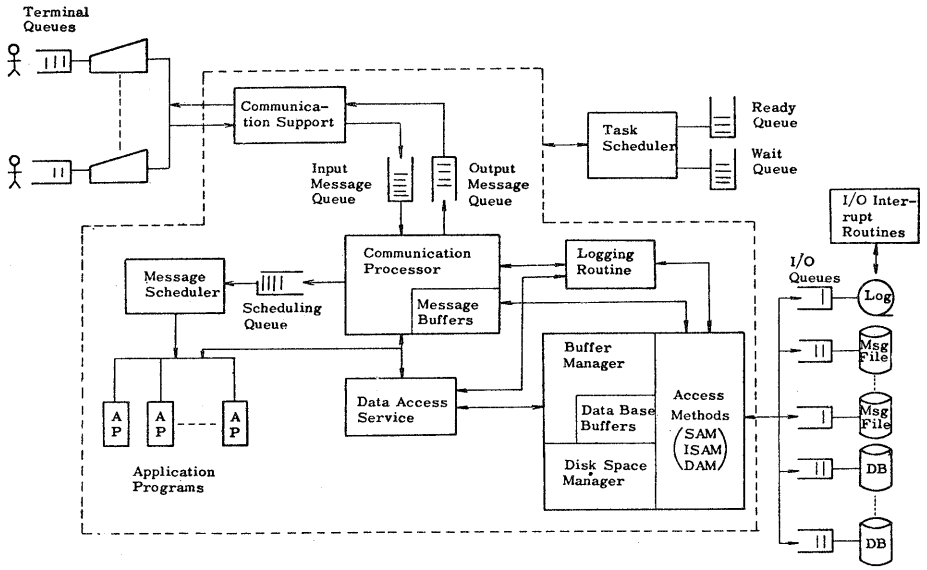

Figure 1—Overall view of simulation model

シミュレーションを実施するには、アプリケーション・システム毎に異なる部分、すなわち、(1)、(2) および (5) を準備しなければならない。Figure 1 にモデルの全体構成図を示す。シミュレーションによって得られる結果としては、次のようなものがある。

- 各種リソースの利用率(CPU, チャネル、ディスク、磁気テープ、通信回線、バッファ・プール)
- ソフトウェア・モジュールの使用統計
- レスポンス・タイム/スループット
- データ・ベースおよびその他のI/O回数
- トランザクション当たりのCPUステップ数
- 各種キューの統計

これらの結果から、システム設計者は、システムの能力や、ボトル・ネックおよびクリティカル・ファクタを明らかにすることができる。また、システムの処理能力の改善手順や、ファクタ間のTrade-off関係をつかむこともできる。

ハードウェア定義

ここでは、ハードウェア構成ならびに次に示すような、ハードウェアの特性を指定する。

- CPU -------- 平均命令実行時間
- 通信回線------送信速度と伝送所要時間
- 端末----------メッセージ発生率
- チャネル-------データ転送率とInterference Rate

・ディスク--------データ転送率、平均回転待時間、シーク時間分布、
およびＲＰＳ（Rotational Position Sensing）Lead / Hold 時間
・磁気テープ------データ転送率およびスタート時間

アプリケーション定義

ここで最も重要な情報は、データ・ベースの定義であり、それは、データ構造、ストレージ構造およびアクセス法により特徴づけられる。Figure 2、Table 1 およびTable 2に、データ・ベース定義情報をどのようにモデル中で取り扱っているかを示す。

(1) データ構造―― モデル化の対象となったＤＢＭＳは、ハイアラキー構造をサポートしており、その構成要素をセグメントと呼ぶ。Table 1 中、項番6以降でデータ構造を規定する。

(2) ストレージ構造―― Table 1 の項番2と5、およびTable 2 の項番2～7がストレージ構造に関係する情報である。データ構造に比べ、ストレージ構造は、モデル中で正確に表現することは容易ではない。それは、何万件あるいは、何十万件というレコードに対して、その個々の情報を持つことは、困難であることによる。したがって、ここではディスクの割当てやブロックサイズといった環境情報を保持し、これに、後で述べるデータ・ベース・アクセス・コールに工夫をこらすことによってこの問題を解決するようにしている。

(3) アクセス法―― ＤＢＭＳは、基本的なアクセス法として、シーケンシャル（ＳＡＭ）インデックスト・シーケンシャル（ＩＳＡＭ）および、ダイレクト（ＤＡＭ）の各アクセス法をサポートしている。

Table 1 – Data Base Definition Table

No.	CONTENTS
1	Data Base Name
2	Disk Type to Store Data Base
3	Pointer for FDT 1
4	Pointer for FDT 2
5	Segment Overflow Ratio
6	Number of Segments in Data Base
7	Segment Name
8	Segment Length
9	Segment Level in Hierarchy
10	Parent Segment Name
11	Occurrence under Parent Segment
⋮	Repeat 7 ~ 11 until Last Segment

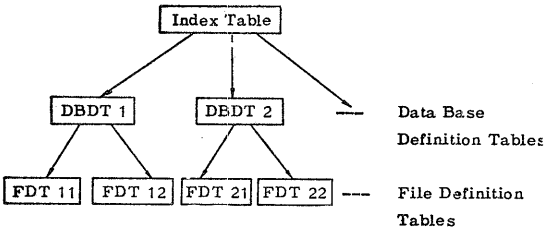

Figure 2 – Table Relationships in Data Base Definition

Table 2 – File Definition Table

No.	CONTENTS
1	Access Method
2	Start Address in Disk Number
3	Start Address in Cylinder Number
4	Number of Blocks Allocated
5	Block Length
6	Number of Blocks per Track
7	Logical Record Length

ＯＳモデル

汎用のＯＳモデルを作るのが、目的ではないため、膨大なＯＳの機能のうち、データ・ベースに必要な部分だけをモデルに取り入れた。その主な対象モジュールについて以下に説明する。

(1) タスク・スケジューラー

これは、マルチ・プログラミングおよびマルチ・タスキングの制御を行なうのが目的である。ＤＢＭＳおよびアプリケーション・プログラムのタスクの起動／停止、レディ・キュー／ウェイト・キューの維持ならびに各タスクの優先度に応じたタスク管理を行なう。

(2) コミュニケーション・サポート

通信回線と端末の管理、ポーリング制御、メッセージの送受信およびメッセージＩ／Ｏ完了割込みのサービスを司どる。端末にメッセージが発生するとProductive Pollが起こり、メッセージの転送が始まる。メッセージの転送が完了すると、そのメッセージは、入力メッセージ・キュー上に置かれる。逆に、出力メッセージ・キューにメッセージがあれば、端末に送信し、送信完了後ポーリングが再開される。

(3) Ｉ／Ｏ割込みサービス・ルーチン

Ｉ／Ｏ装置の型に応じた割込みルーチンが準備されている。モデルで許しているＩ／Ｏ装置としては、磁気テープ（ログ用）、Ｈ8578 ディスク（集団ディスク、Ｈ8589 ディスク（大容量）がある。リード、ライトおよびシークの完了により、これらのルーチンに起動がかかる。

ＤＢＭＳモデル

全体のモデルの中で、最も重要な部分であり、詳細にモデル化されている。

(1) メイン・コントローラ

ＤＢＭＳ全体の管理を行なう部分である。ログ情報の取得、データ・ベース・システムの状態管理、メッセージのスケジュールなどを行なう。入力メッセージが到着すると、コミュニケーション・プロセッサにコントロールを渡す。そこで入力メッセージの処理が終了すると、そのメッセージの優先度に応じて、スケジュールを行ない、アプリケーション・プログラムの起動要求を発行する。また、アプリケーション・プログラムから、データ・アクセス・コールが出されると　その処理のため、データ・アクセス・サービスにコントロールを渡す。

(2) コミュニケーション・プロセッサ

ＯＳのコミュニケーション・サポートと連携して、入出力メッセージのサービスを行なう。次の場合にコントロールを得る。

(a) コミュニケーション・サポートが、メッセージの入出力を完了した時

(b) 送信すべきメッセージが、ＤＢＭＳ内に存在する時

入力メッセージ・キューにメッセージがあると、それをスケジューリング・キューに移し、同時に、ディスク上にそのコピーを作成する。同様出力すべきメッセージがある場合には、メッセージ送信の要求を発行し、同時にディスク上にコピーを作成する。

(3) データ・アクセス・サービス

本モジュールは、アプリケーション・プログラムで発行された、データ・

アクセス・コールにより、起動される。ここには、データ・アクセス・コール――検索、挿入、置換、削除――に対するサービス・ルーチン、バッファ管理モジュール、ディスク・スペース管理モジュールおよび、アクセス法、（ＳＡＭ、ＩＳＡＭ、ＤＡＭ）をサポートするモジュールが含まれる。

モデルでは、データ・アクセス・コールのファンクション・コードとして、より正確なシミュレーションを行なうため、実際とは少し異なった形のものを採用している。すなわち、Get Random, Get Random Next, Get Immediate Next, Get Link, Insert Random, Insert Immediate Next, Replace および、Delete といったファンクション・コードを準備している。たとえば、Get Random は、そのデータ・ベース中から、１つのレコードをランダムに選定し、そのフィジカル・アドレスを決定して、その検索を行なう（インデックスを持っていれば、そのインデックスを経由して）。Get Random Next は、前に得られたセグメントに続くセグメントを、１つのレコード内でランダムに選び出し、その両者のハイアラーキカル構造上の距離を、各セグメントの、オカーレンスに基づいて計算し、アクセスを行なう。Get Immediate Next は、前に検索したセグメントのすぐ次のセグメントを持ってくる。Get Link は、当該データ・ベースに対して、１回のフィジカル・リードを行なう。

すべてのデータ・ベースＥ／Ｏ要求は、すべてバッファ管理モジュール（Buffer Manager）を介して行なわれる。したがって、Buffer Manager は、モデル中でも、特に重要な存在である。Figure３に、実際の Buffer Manager の位置付けと、その機能をモデル化するために、モデル中に設けた情報とを示す。Buffer Manager は、すべてのデータ・ベースに共通のバッファ・プールを設け、その中に、ダイナミックにバッファを割当てる方式をとっている。バッファ管理方式としては、ＬＲＵ、（Least Recently Used）法を採用している。バッファ・プール中の各バッファは、参照された順番にチェインされている。データ・ベース・アクセスの要求が出されると、Buffer Manager は、まずバッファ・プール中を、このチェインをたどってサーチし、目的のデータがすでにバッファ・プール中にあるかどうかをチェックする。もし、目的のデータが存在しない場合には

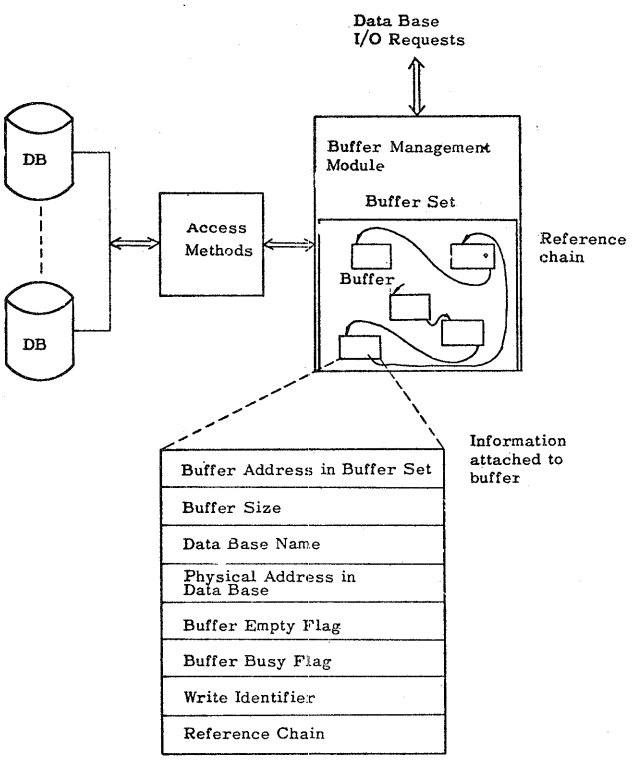

Figure 3 - Buffer management position and buffer information

ディスクから、データを読み込むためのバッファ・スペースを得るために、チェインを逆方向にサーチする。Buffer Managerはまた、同時データ・ベース・アクセスの管理も行なう。

アプリケーション・プログラム・モデル

アプリケーション・プログラムの記述は、一般に非常に簡単である。ここで記述するのは、主に、アプリケーション・プログラムで実行されるCPUステップ数と、処理の流れの制御と、データ・アクセス・コールである。データ・アクセス・コールのフォーマットは、次のようになっている。

DA　コール・ファンクション・コード、データ・ベース名、セグメント名

3. シミュレーション例

これまでに行なってきた、あるいは行ないつつあるシミュレーション例としては、次のようなものがある。

- 製造会社のオンライン設計情報システムの性能予測
- 保険会社におけるオンライン・データ・ベース・システムの性能予測
- バッファ管理特性のシミュレーション
- Multi－Programmingに関するシミュレーション
- データ・ベースのメモリ・ハイアラーキの特性シミュレーション

ここでは、その一例として、バッファ管理特性のシミュレーション結果について述べる。Figure 4は、メッセージ・トラフィックをパラメータとした時の、バッファ・プール・サイズとレスポンス・タイムとの関係を示している。同時に走るアプリケーション・プログラム・タスクの数は、5個で、いずれも問い合わせ処理のみを行なうものである。

Figure 4から次のことがわかる。

- メッセージ・トラフィックが低い範囲においては、バッファ・プール・サイズのレスポンス・タイムへの影響は、ほとんどない。
- メッセージ・トラフィックが上昇するにつれて、必要なバッファ・プール・サイズは、大きくなる。しかし、それもある点で飽和し、それ以上は、バッファ・プール・サイズを大き

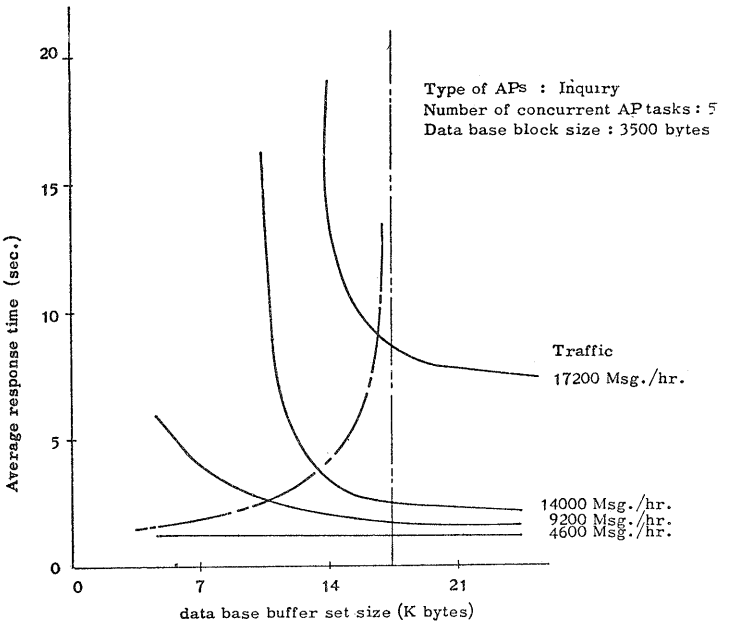

Figure 4 – DB buffer set size vs. response time

くしても、レスポンス・タイムの向上は望めない。
図中、一点鎖線は各メッセージ・トラフィックにおいて、最小必要なバッファ・プール・サイズを結んだものである。図から、それが17.5KBすなわち、5バッファを含むバッファ・プール・サイズに近づいていっているのがわかるであろう。すなわち、同合せのみのシステムにおいては、同時に走るアプリケーション・プログラム・タスク数に等しいだけのバッファ数があれば十分であることを示している。

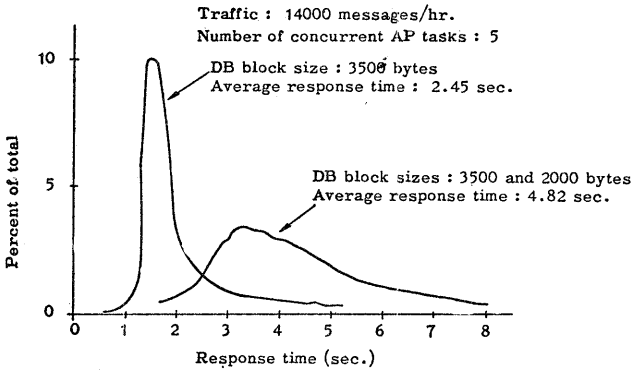

Figure 5 - Response time distribution (2)

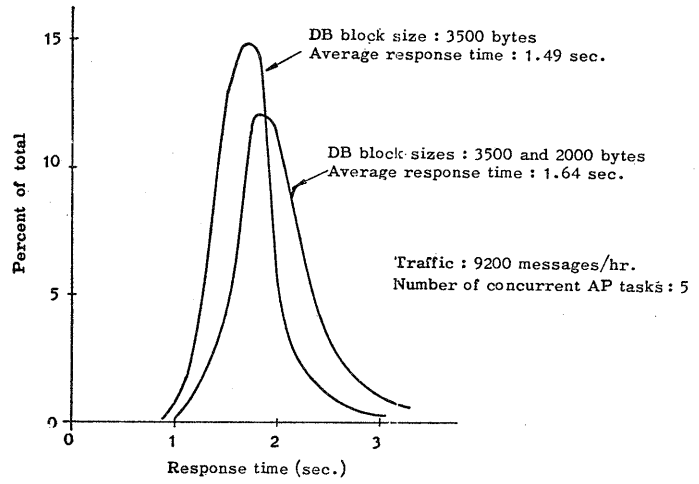

Figure 6 - Response time distribution (3)

Figure 5は、データ・ベースのブロック・サイズを統一した場合と、統一しない場合のレスポンス・タイムの分布を示したものである。それ以外の条件は、両者全く同一である。ブロック・サイズを統一しなかった場合の平均レスポンス・タイムは、統一した場合の2倍近くになっている。これは、前に述べたように、Buffer Managerが、データ・ベース共通のバッファ・プールをダイナミックに管理していることによる。異なるブロック・サイズがあると、バッファ・プールを使用しているうちに、バッファ・サイズが、不ぞろいになり、Garbage Collectionが必要となる。そして、バッファのコレクション中は、他のタスクが待ち状態となり、マルチ・タスクの効果が活かされない。そのために、レスポンス・タイムが大きくなってしまう。この場合は、2種類のブロック・サイズであったが、これが更に多種類のブロック・サイズを持てば、その影響は、ますます大きくなる。また、この影響は、Buffer Managerのコール頻度が増加すればする程、顕著になる。
Figure 6は、Figure 5と同じ条件のもとで、メッセージ・トラフィックだけを、9200メッセージ／時に変えた場合の結果を示す。この場合には、両者の間に、Figure 5のような大きな差違はないことがわかるであろう。

4、モデルの検証

シミュレーションにおいて、最も注意しなければならないことは、その結果の妥当性についてである。シミュレーションは、モデルができ上がると、それらしい結果が出てくるため、つい結果を信用しがちである。これを防ぐためには、何らかの形でモデルの妥当性の検証を行なう必要がある。もしもモデルが、特定のシステム建設の事前評価にのみ使用されるものであれば、そのシステムができ上がってしまえば意味を持たなくなるため、モデルの作成に当たっては、できるだけ重要なファクタが漏れないようにするしかない。しかしながら、本モデルのように、ＯＳおよびＤＢＭＳがソフトウェア部分のほとんどであり、かつそれらは汎用的に使用されるものである場合には、ＤＢＭＳが完成した段階で、実測により検証するのが最も適している。

ここで１つ考えておかなければならないことは、実際との誤差がどの位の範囲に入っていれば、そのモデルが妥当と言えるかという問題である。にとえば、設計の初期段階にあっては、最大５０％程度で許される場合があるであろうし、マクロ・シミュレータＣＡＳＥでは、日立中研での使用経験上、約２０％と考えられる。これは、入力の精度も良くなっていなければならないのはもちろんである。さて、我々のモデルの場合は、ＤＢＭＳの方式検討をも考慮して、Event－Driven型のコンピュータ専用シミュレータを用いて、詳細なモデル作りを目標としているため、その誤差は１０％以内に納まることを目標とした。

以上のような背景から、我々はハードウェア・モニタを使用してデータ・ベース・システムの実測を行なった。測定対象システムは、２種類で３０ケース、取得したデータは、下記の項目である。

- ＣＰＵアクティブ時間
- ＣＰＵアクティブでスーパーバイザ状態である時間
- 実行命令数
- スーパバイザ状態における実行命令数
- チャネル・ビジー時間
- ディスクのシーク、サーチ／リード、ライトおよびＲＰＳの各状態の生起回数と時間

上記データから、シミュレータの特性を勘案し、精度を代表するデータとして、下記項目を抽出してシミュレーション結果との比較を行なった。

- 経過時間
- 各処理状態におけるＣＰＵ利用率
- データ・ベース・コール当たりの、ＣＰＵステップ数、処理時間、ディスクＩ／Ｏ回数とＩ／Ｏ時間
- チャネル・ビジー時間

その結果、誤差１０％以内という目標を満たしていることがわかった。

5、おわりに

データ・ベース・システムの性能の一評価方法として、我々が開発したシミュレーション・モデルを中心に述べた。最初に書いたように、オンライン・データ・ベース・システムは、コンピュータ・システムの中でも、複雑なシステムであり、まだ解明されるべき、多くの分野を残している。これらの解明のためには、シミュレーションに限らず、解析的方法、モニタリング技法などのいろいろな、

性能評価技法を有機的に駆使して進めていくことが重要であろうと考えている。

6. 参考文献

(1) F. Nakamura, I. Yoshida, H. Kondo "A Simuration Model for Data Base System Performance Evaluation" Proc. of Ncc, 1975

(2) 吉田, 近藤, 中村, 葛西, 渡辺"データ・ベース・マネジメント・システムの一評価、(その1, その2)"情報処理学会第14回大会予稿集, 昭48.

(3) 中村, 森, 吉田他"データ・ベース・マネジメント・システムのシミュレーション(その1, その2, その3)"情報処理学会第15回大会予稿集 昭49.

(4) M. E. Senko et al "Data Structures and Accessing in Data-Base Systems" IBM Systems J. No.1, 1973

(5) V. Y. Lum, H. Ling, M. E. Senko "Analysis of a Complex Data Management Access Method by Simulation Modeling" Proc. of FJCC, 1970.