分散型データベースシステムＪＤＤＢＳにおけるスキーマ層設計とアクセス問題について

（財）日本情報処理開発協会　　竜沢 誠

1. 序

分散型データベースシステム（以下ＤＤＢＳと記す）とは、ネットワークによって結合されたデータベースシステム（以下ＤＢＳと記す）を、１つの論理的なＤＢＳに仮想化したシステムである。ＤＤＢＳ実現には、大別して分割型（top-down）と統合型（bottom-up）との２つのアプローチがある。分割型としては、SDD-1[ROTH80], distributed INGRES[STON77]等がある。これらは、同種ＤＢＳから成っていることが特徴の１つである。一方、統合型では、既存ＤＢＳから論理的に新たなＤＢＳが構成されることになる。例としては、POLYPHEME[ADIB78]、及び我々のＪＤＤＢＳ[TAKI78, 79, 80a,b,c,d]がある。統合型では、一般に構成ＤＢＳは異種である。

本論文では、統合型ＤＤＢＳとしてのＪＤＤＢＳ（Jipnet ＤＤＢＳ）の全体アーキテクチャと、これに基づいたスキーマ層設計問題とアクセス問題への解を示す。２.ではＤＤＢＳ検討上の仮定を、３.では全体アーキテクチャを、４.ではスキーマ層設計を、５.ではアクセス問題を論じる。

2. 仮定

統合型ＤＤＢＳでは、異種のＤＢＳの統合が必要となる。ここでＤＢＳの異種性とは、a)データモデル、b)言語、と格納されたデータの意味（これは、a)とb)とによって記述されている）の各々の相違と定義する。即ち、ＤＤＢＳから見た各ＤＢＳは、あるモデルと言語、及びこれに基づいたデータ意味記述（スキーマ）とによって特徴づけられるブラックボックスと考えられる。

通信ネットワークは、基本的通信手段、i.e. 発信地－目的地間のトランスペアレントな高信頼通信手段を提供できることとする。

3. ＤＤＢＳ問題と全体アーキテクチャ －四層スキーマ構造（ＦＳＳ）[TAKI78, 79]

統合型ＤＤＢＳ（以下単にＤＤＢＳと記す）は、ネットワーク上に分散した異種ＤＢＳを統合したものであることから、従来の集中型ＤＢＳ問題に加えて次の２点が主要問題となる。

a) 各ＤＢＳのモデル（とこのモデルに基づいた）言語の各々の相違の解決。

b) 複数のＤＢＳと１つの論理的ＤＢＳ（i.e. ＤＤＢＳ）との関係を明らかにする（各ＤＢＳの格納データベースの意味の相違の解決）。

前者を異種性問題、後者を分散問題と呼ぶ[TAKI78]。

ＤＤＢＳのスキーマ層の決定、i.e. 全体（gross）アーキテクチャは、この２つの問題を解決するものでなければならない。我々は全体アーキテクチャとして四層スキーマ構造（four-schema structure）（以下ＦＳＳと記す）を設けた[TAKI78, 79]。これは、まず各ＤＢＳのデータベースの意味の記述系としてのモデルと言語とを同種化（共通化）し、ついで、各データベースの意味から新たなデータ記述を生成しようとするものである。ＦＳＳは、a)４つのスキーマ層、b)隣接スキーマ層間のマッピング、c)マッピング情報（directory/dictionary）とから成っている[see Fig. 3.1]。ＦＳＳの４つのスキー・

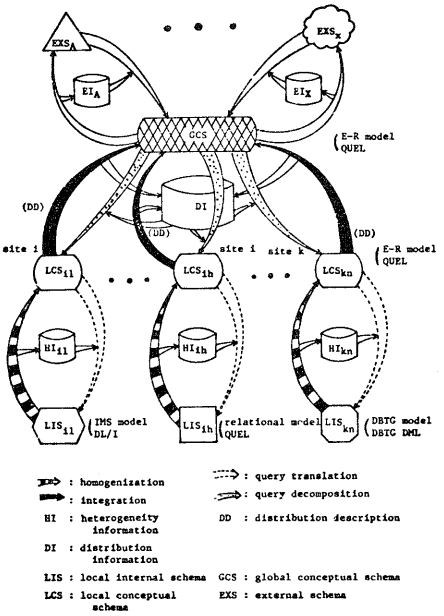

Fig. 3.1 FSS

マ層は、a)ローカル内部スキーマ(LIS)、b)ローカル概念スキーマ(LCS)、c)全体概念スキーマ(GCS)、d)外部スキーマ(EXS)である。ローカル内部スキーマは、各DBSがDDBS環境下でサポートし得るデータの記述であり、既存DBSのスキーマ又はサブスキーマに対応している。ローカル概念スキーマは、DDBS全体で共通なモデルと言語によって LISを記述したものである。ここにおいて、各DBSのモデルと言語との異種性(異種性問題)が解決されたことになる。全体概念スキーマは、LCSの集合から生成された、DDBS全体に対する統一的なデータ記述、i.e. あるネットワーク共同体にとって意味のあるデータの記述、である。この層において、分散問題、i.e. 各DBSの意味の相違、が解決されたことになる。即ちDDBSは、1つの論理的な巨大DBSに仮想化されたことになる。外部スキーマは[ANSIX75]におけるものと等価であり、DDBS固有の問題ではない。よって本論文ではEXSを考えない。

共通モデルの設定においては、スキーマ層設計用のモデルと、これの実現/処理用のモデルとを考える必要がある。本には、設計用にE-Rモデル[CHENP76]を、処理用にリレーショナルモデル[CODDE70]を用いた。設計時には、より多くのデータの意味記述を必要とし、処理においては、より簡単さが要求されるからである[DAKIM79]。

層間のマッピングは、a)設計マッピングとb)アクセスマッピングとの2つから成っている。設計マッピングは、各スキーマ層がどのように生成されるかを示している。FSSでは、先ずLISからLCSが生成され(同種化)、ついでLCSの集合からGCSが生成され(統合化)る。

アクセスマッピングは、上位層のアクセス要求(問合せと呼ぶ)を、いかに下位層に変換するかを表もしている。FSSでは、GCS層の問合せをLCS層に変換し(問合せ分割)、各LCS問合せをそのDBSで実行可能なアクセス要求に変換し(問合せ変換)する。

アクセスマッピングに必要な情報(マッピング情報)は、スキーマ層設計時に全体データディレクトリ(GDD)として設計管理者によって生成される。同種化で生成される異種性情報(HI)は、各DBSでのLISとLCSとの対応を表もし、問合せ変換で用いられる。統合化で生成される分散情報(DI)は、LCSの集合とGCSとの対応を表もし、問合せ分割で用いられる。

上述してきたFSS概念をより具体化した図をFig.3.2に示す。記述法は[ANSIX75]に基づいている。図の上半分は設計を、下半分はアクセスを表もしている。設計及びアクセス機能は、互いにGDDを介して通信する。図中の各処理機能及びインタフェース、ユーザ及び管理者の役割の明確化によって DDBSの全体像をとらえることが出来ると考えている。

4. スキーマ層設計

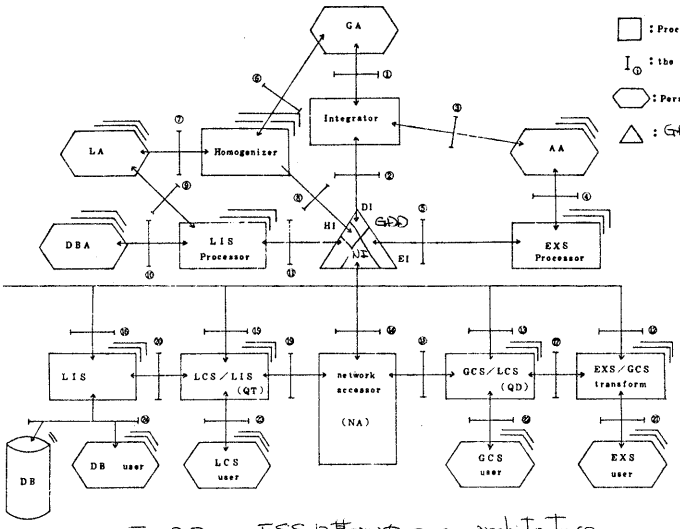

Fig.3.2 FSSに基づいた gross architecture

スキーマ層設計マッピングとしては、同種化と統合化とがある。

4.1 同種化[TAKIMI方]

同種化(homogenization)は、LISからLCSを生成する過程である。これは、各DBSに固有なLISを、共通モデルと言語によって共通記述することによってなされる。我々が採用したE-Rモデルと、現存の代表的な3つのモデル(relational, DBTG, IMS)との要素の対応をFig.4.1に示す。よって、LISからLCSは、対応するモデル要素を1対1に対応させることによって変換できる。

E-R model	value-set	attribute	entity-set	relationship-set
relational	domain	attribute	relation	relation
DBTG	item のカーラン	item	record-type	set-type link record-type
IMS	field のカーラン	field	segment-type	hierarchical path

Fig. 4.1 モデルの対応

図4.2は、DBTGモデルとE-Rモデルとの要素の対応を示している。レコード型は事象集合に、セット型とリンクレコード型(eg. R)は関係性集合に変換される。

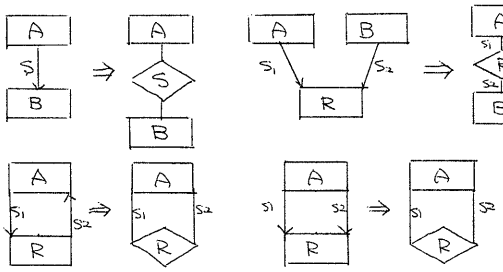

Fig.4.2 DBTG と E-R との対応

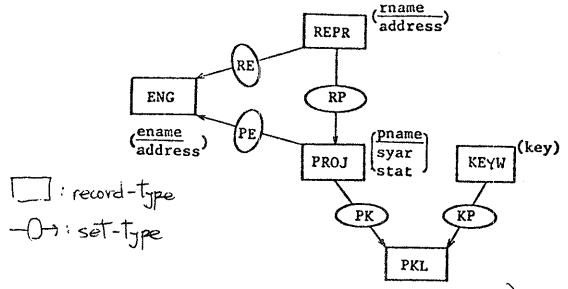

Fig.4.3 LISの例(DBTGモデル)

```
ESR REPR (rname, address)
    ENG  (ename, address)
    PROJ (pname, syar, stat)
    KEYW (key)
RSR RP   (rname, pname)
    RE   (rname, ename)
    PE   (pname, ename)
    PKL  (pname, key)
```

Fig.4.4 Fig.4.3のLCS

例として、Fig.4.3のDBTG LISを考えてみよう。レコード型REPRは事象集合リレーション REPR (rname, address) となる。セット型RPは、事象集合リレーション(ESR) REPRとPROJを結びつける関係性集合リレーション RP (rname, pname) となる。rnameは、REPRのキー、pnameはPROJのキーである。PKLはリンクレコード型なので、PROJとKEYWを結びつける関係性集合リレーション(RSR) PKL (pname, key) となる。

Fig.4.2のようなDBTGモデル構造に加えて、Fig.4.5の様なモデル構造がDBTGモデルは持っている。これらの情報は、異種性情報(HI)[Fig.4.6]として、リレーション形式で格

a) レコード型
- データ項目構成
- key　CALC with DNA
-　　CALC with DA

b) セット型
- 親子レコード型
- メンバシップクラス
- ソート　with DNA
　　DA
- structural constraint

Fig. 4.5 DBTG モデル構造

ESR (entity-set-name, area-root-name, number-of-keys, access-mode, degree, width, size, protection-flag, integrity-flag, cardinality)
ATT (es-rs-name, es-rs-type, attribute-no, attribute-name, value-set, role-of-attribute, character/decimal, length, protection-flag, integrity-flag, cardinality, selectivity)
RSR (relationship-set-name, source-es-name, destination-es-name, access-mode, relationship-construct, degree, width, size, source-set-name, destination-set-name, protection-flag, integrity-flag, cardinality)
c.f. es: entity-set　rs: relationship-set

Fig 4.6 HIスキーマ

納される。LISの表わす意味的構造はE-Rモデル[see Fig.4.2]で表わし、LISモデル固有の構造はHIとしてデレクトリに格納する。

我々の同種化手法は、検索要求の処理を行なうために十分である[see 5.1]。更新要求を処理するためには、LISとLCSとの等価性の実現が必要になる。これは今後の課題である。又、我々の手法を他のモデル(e.g. IMS)に拡張することは容易である。

4.2 統合化 [TAKIM80c,d]

統合化は、LCSの集合からGCSを生成する過程である。統合化では、共通記述された各DBSの格納DBの意味から、新たなデータの意味を生成することが問題となる。我々は、意味の対応と付与は全体管理者(GA)が行なうものとし、スキーマ層の実現モデルとしてのリレーショナルモデルの生成について考える。即ち、GCSリレーションを、LCSリレーションから定義する言語(GSDLと呼ぶ)を設けた。この定義は、リレーショナルモデルにおける視野(view)の定義に似ている。関係計算に基づいた視野定義[STONM76]では、複数リレーション間の演算として結合のみが許される。しかしDDBSでは、結合に加えて和(union)演算が複数リレーション演算として必要になる。

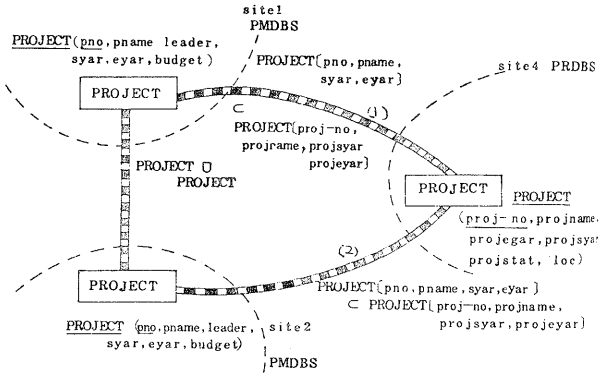

Fig.4.7 セマンティックリンク

```
drange (p1, p2, p) (PROJECT:1, PROJECT:2, PROJECT:4);
define ESR PROJECT(pno, pname, manager , budget, loc)
(1) { (p1.pno, p1.pname, manager,=p1.leader p1.budget, p.loc)
      where p1.pno = p.proj-no and p1.pname = p.projname and
            p1.syar = p.projsyar and p1.eyar = p.projeyar :
(2) { (p2.pno, p2.pname, manager = p2.leader, p2.budget, p.loc)
      where p2.pno = p.proj-no and p2.pname = p.projname and
            p2.syar = p.projsyar and p2.eyar = p.projeyar ;
```

```
drange (l1, l2, ···, lm)(L1 : s1, L2 : s2, ···, Lm:sm);
define <type> <gcs-rel-name> (<gcs-att-list>)
    <sub-def>{:<sub-def>};
```

GSDLは、上記の様に定義される。drange文は、サイトsiにあるLCSリレーションLiに対して組変数liを定めている。define文はGCSリレーションを定義する。GCSリレーションは、先ずLCSリレーションの結合を<sub-def>で定義し、ついでこれらの結果の和として定める。<sub-def>は、次の形式を持つ。

Fig.4.8 GCSリレーションの定義

<sub-def> ::= (<target-list>) where <qual>

<target-list>は、<qual>によるLCSリレーションの結合結果のスキーマを、後の

和を取るために同一にするためにある。LCSリレーション間の関係性は、図4.7に示す様なセマンティックリンクによって示される。セマンティックリンク(SL)は、2つのLCSリレーション間に存在し、互いに関連する部分の集合論的関係性を示している。例えば、PROJECT O PROJECTは、サイト1及び2にあるPROJECTリレーションが和をとれることを示している。(1)のSLは、PROJECT(at 1)の部分(i.e.射影)が、PROJECT(at 3)の部分の部分集合であることを表わしている。図4.8は、図4.7のSLに基づいて定義されたGCSリレーションPROJECTを示している。(1)は、サイト1と3との結合を、(2)はサイト2と3との結合を表わしている。

この様にして定義された実現/処理モデルとしてのGCSリレーションの意味、即ち事象集合か関係性集合かは全体管理者(GA)によって定義される。セマンティックリンクは、GCS設計のツールである。

5. アクセスマッピング

アクセスマッピングは、上位層の問合せを、全体ディレクトリ(GDD)内のマッピング情報を用いて、下位層の問合せに変換する過程である。ここでは、問合せ変換[TAKIM80a,b]と問合せ分割[TAKIM80c,d]とについて論じる。

このマッピングは、一般にa)問合せ表現の変換とb)最適化とから成っている。前者は、上位層問合せが参照するモデル構造を、下位層のものに変換する。後者は、表現変換された問合せを、下位層での最適構造に変換する。

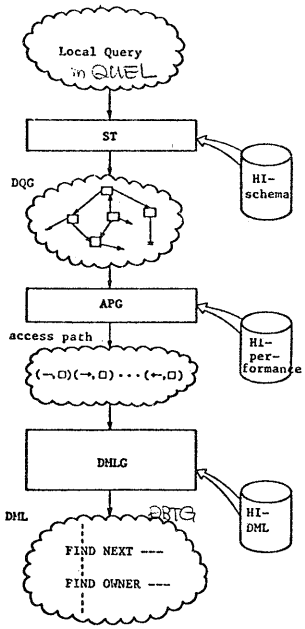

Fig.5.1 問合せ変換の概要

5.1 問合せ変換[TAKIM80a,b]

問合せ変換とは、LCS問合せを、そのサイトで実行可能なLIS問合せに、異種性情報(HI)を用いて変換するプロセスである。我々は、LCS問合せとしてQUELを、LISモデルとしてDBTGモデルを採用した場合の問合せ変換システムを実現した。

5.1.1 仮定

問合せ変換を行なう上で次の様な仮定を設けた。

a) LCS問合せ(以下問合せと記す)はaggregate関数を持たない。(aggregate-free)

b) 問合せは積正規形である。但し、各disjunctは、同一変数を参照するpredicateの和でもよい。

c) 問合せ内の結合は、LCS内のESR-RSRのリンクにそってのみ許される。簡単化のために等価結合のみを考える。

d) 問合せとしては検索のみを考える。

e) 応答時間は、アクセスされるオカーランス数に比例するとする。

5.1.2 構造変換-表現の変換

問合せは、LCSリレーションを参照する非手続的QUEL[HELD78]問合せである。例として、Fig.4.3に対する"DBSを1975年以来研究しているプロジェクトの代表者の部下が同じプロジェクトに属している時、そのプロジェクト名と部下の名前を求めよ"

という問合せを考えてみよう。これはQUELによって Fig.5.2 の様に書ける。問合せは、Fig.5.3 の様なグラフ(リレーショナル問合せグラフ(RQG))として表もせる。ノードは組変数を、ノード間のリンクは結合式を、→は結果属性を、⊣は制限式を表もしている。このRQGはリンクの表もす論理式の積のみを表もせることを注意しておく[sec 2. の仮定 b)]。

```
range (e, p) (ENG, PROJ);
range (re, pe, rp, pkl) (RE, PE, RP, PKL);
retrieve into R (p.pname, e.ename) where
pkl.key = "DB" and pkl.pname = p.pname and
p.pname = rp.pname and rp.rname = re.rname
and re.ename = e.ename and e.ename =
pe.ename and pe.pname = p.pname and
p.syar ≥ 1975 and p.stat = "ON" ;
```

Fig.5.2 QUEL LCS query

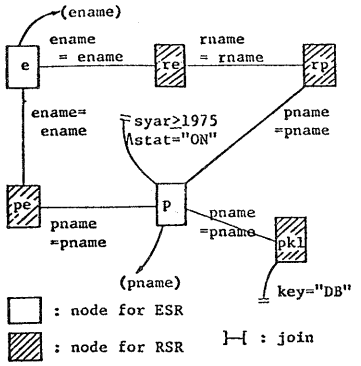

Fig.5.3 RQG

先ず、Fig.4.1 の対応表を用いて、LCS要素をLIS要素で置き換える[Fig.5.4]。この時、LCS上に対応するLIS要素のないもの、e.g. Fig.5.4 の pkとkp、と、LCS上には存在するが問合せには現われないLIS要素(e.g. rとe)とがある。これらを隠れ構造(LHS)と呼ぶ。特に前者を暗隠れ構造(IHS)、後者を明隠れ構造(EHS)と呼ぶ。これらの隠れ構造は、LISとLCSとの対応情報(i.e. 要種性情報)[Fig.4.6]を用いて明らかにされる。最終的にDBTG問合せグラフ(DQG)と呼ばれるグラフ表現[Fig.5.5]が得られる。DQGは、LCS問合せの意味を、DBTGモデル要素によって、非手続的に表もしたものである。

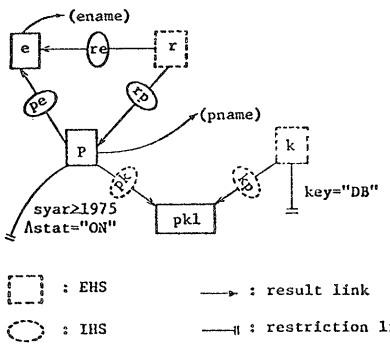

Fig.5.4 RQG要素の置換

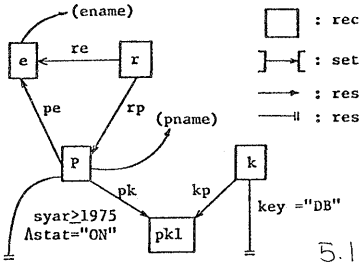

Fig.5.5 DQG

5.1.3 アクセスパスの生成

非手続的DQGからアクセスシーケンスを得ることが次の問題となる。DQGからは、一般に複数のアクセスパスを得れるが、次の目標を達するようなパスを選択する。即ち、

a) 中間結果数を最小化し、b) アクセスされるオカーランス数をなるべく少なくする。a)は、中間結果の管理(ファイルと処理)を簡単化するために必要である。b)は、よりよい応答時間を得るために重要である。

我々のアクセスパス生成アルゴリズム(DFAと呼ぶ)は、グラフを線型にサーチすることによって、ただ1つの(中間)結果ファイルを必要とするだけで、かつアクセスオカーランス数を納得出来る程度に減少させることができる。このアルゴリズムの詳細は、Fig.5.6に示す。DFAによって、DQGからアクセスパスを表もす木(アクセス木(AT)と呼ぶ)が生成される。AT枝は、DQGアーク(i.e. セット型)にユニークに対応するが、あるDQGノード(レコード型)に対応するATノードは、冗長に存在する。この様なATノードを合流ノードと呼ぶ。合流ノードの存在は、あるレコード型が異なったセット型を介して複数回アクセスされることを示している。よって、1つのアクセスパス内でアクセスされ条件式を満足する合流ノードのオカーランスは同一でなければなら

0. [initialization]
   let STB(x) be an adjacency node list for the node x in which branches $(x,y_1),\ldots,(x,y_n)$ are sorted in the ascending order of the occurrences to be accessed (OCA) of the adjacent nodes, i.e.

   $OCA_{y_1} \leq \ldots \leq OCA_{y_n}$, where $OCA_{y_i} = OCA_x \cdot S_{xyi}$

   for all the nodes (x) in the DQG, mark them NEW and create STB(x); AT ← Λ; pushdown (Λ);
1. select the starting node(x), which has the least OCA in all nodes;
2. pushdown(x); if x is marked NEW, then mark it OLD; link the AT node x to the DQG node x. if x is marked OLD, then mark both it and the AT node linked to the DQG node x CONFLUENT.
3. if STB (x) is empty, go to 7.
4. [search STB(x)]
   get the first pair (x, y_1) from STB(x);
5. link the node y_1 to the AT;
6. [delete (x, y_1)]
   delete (x, y_1) from STB(x) and delete (y_1, x) from STB (y_1); x ← y_1; go to 2;
7. [pop up]
   popup (x); if x = Λ, terminate; go to 3;

Fig.5.6 DFA

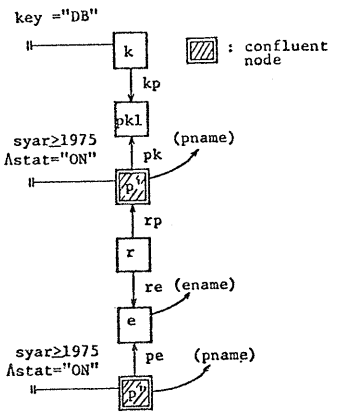

Fig.5.7 Fig.5.5のAT

ない。このため、合流ノードではカウンタの保持とともに、中間結果が必要となる。しかし、DFAでは、あるDQGノードに対する全ての合流ノードは、この中の1つを根ノードとする部分木内に存在することが保障されている。このことは、根ノードのカウンタはキー値のみを保持すればよい。即ち全体として1つの中間結果があればよいことになる。これは、DFAの最大の長所である。

DFAでは、HI内のカーディナリティ、選択度、結合度というパフォーマンス情報を用いて、次のノードを決定する時のアクセスされるオカーランス数の見積もりを行ない、一番少ないノードを次のノードとしている。このことによって、早い時期に不要なアクセス空間を縮少でき、b)の目標を達成できる。アクセス・オカーランス数は次の様にして見積もられる。Aを現在のレコード型、Bをセット型Sを介してリンクされたレコード型とし、C_A, C_Bを各々のカーディナリティとする。S_Aを、Aに関する制限式の選択度、OCA_Aを、現在アクセスされるオカーランス数、sl_{AB}をAに対するBの結合度(Aからの親ならば $sl_{AB} \geq 1$, 子ならば $sl_{AB} = 1$)とする。BのAを介してアクセスされるオカーランス数 OCA_B は次のようになる。

$$OCA_B = OCA_A * S_A * sl_{AB} \quad \cdots (1)$$

DFAによって生成されたアクセス木(AT)を、Fig.5.7に示す。

5.1.4 DMLの生成

アクセス木(AT)をpre-orderにたどることによって得られる枝(i.e. セット型)とその下についたノード(i.e. レコード型)との対のシーケンスは、求めるアクセスパスを表わしている。この対をアクセス単位と呼ぶ。各アクセス単位は、10個のアクセスパターンとの照合が行なわれ、対応するDMLブロックが生成される。これらのDMLブロックのシーケンスは、DMLプログラム(COBOL)の手続部となる。パターンの詳細については[HDBS80]を参照されたい。以上の様にして、我々は最終的に、Fig.5.2のQUELし問合せを、Fig.5.8の様なDMLプログラムに変換できる。

5.1.5 まとめと今後の課題

DFAは、中間結果をただ1つだけしか必要としない点で運用上優れたアルゴリズムである。アクセスされるオカーランス数について、次のレベルまでの見積もりをやっているだけであるが、より深いレベルまで調べることも出来る。しかし、

```
*---------- DBTG DML ----------
        MOVE FALSE TO LFOUND.
L0101.  CALL  GET-NEXT-VALUE ((KEYWORD = "DATABASE" ),VAL,MODE).
        IF  MODE = 'END'  GO TO TERM.
        MOVE VAL TO KEYWORD IN KEYWORDS.
        FIND ANY KEYWORDS.
        IF  NOTFOUND = 'YES'  GO TO L0101.
        GET KEYWORDS.
*
L0201.  IF  L0210 = TRUE  GO TO L0202.
        MOVE FALSE TO LFOUND.
        IF  KEYW-PROJ IS EMPTY  GO TO L0101.
        MOVE TRUE  TO L0210.
        MOVE FALSE TO L0211.        GO TO L0203.
L0202.
        IF  L0211 = TRUE  OR  LFOUND = TRUE
                MOVE TRUE  TO L0211
            ELSE MOVE FALSE TO L0211.
L0203.  FIND NEXT PROJ-KEYW-LINK WITHIN KEYW-PROJ.
        IF  ENDSET NOT = 'YES'  GO TO L0204.
        MOVE L0211 TO LFOUND.
        MOVE FALSE TO L0210.
        IF  L0211 = FALSE  GO TO L0101.
                           GO TO L0101.
L0204.  GET PROJ-KEYW-LINK.
*
L0301.  FIND OWNER WITHIN PROJ-KEYW.
        IF  NOTFOUND NOT = 'YES'  GO TO L0303.
L0302.  MOVE FALSE TO LFOUND.
        GO TO L0201.
L0303.
        GET PROJECT.
        IF  NOT ( PROJSTAT = "ON" AND PROJSYAR GE  1975 )  GO TO
          L0302.
        MOVE TRUE TO LFOUND.
        CALL  RESULT (PROJNAME ).
```

```
L0401.  FIND OWNER WITHIN REPR-PROJ.
        IF  NOTFOUND NOT = 'YES'  GO TO L0403.
L0402.  MOVE FALSE TO LFOUND.
        GO TO L0201.
L0403.
        GET REPRESENTATIVE.
        MOVE TRUE TO LFOUND.
*
L0601.  IF  L0610 = TRUE  GO TO L0602.
        MOVE FALSE TO LFOUND.
        IF  REPR-ENGI IS EMPTY  GO TO L0201.
        MOVE TRUE  TO L0610.
        MOVE FALSE TO L0611.        GO TO L0603.
L0602.
        IF  L0611 = TRUE  OR  LFOUND = TRUE
                MOVE TRUE  TO L0611
            ELSE MOVE FALSE TO L0611.
L0603.  FIND NEXT ENGINEER WITHIN REPR-ENGI.
        IF  ENDSET NOT = 'YES'  GO TO L0604.
        MOVE L0611 TO LFOUND.
        MOVE FALSE TO L0610.
        IF  L0611 = FALSE  GO TO L0201.
                           GO TO L0201.
L0604.  GET ENGINEER.
        CALL  RESULT (ENGINAME ).
*
L0701.  FIND OWNER WITHIN PROJ-ENGI.
        IF  NOTFOUND = 'NO'  GO TO L0703.
L0702.  MOVE FALSE TO LFOUND.
        GO TO L0601.
L0703.
        GET PROJECT.
        IF  NOT ( PROJSTAT = "ON" AND PROJSYAR GE  1975 )  GO TO
          L0702.
        MOVE TRUE  TO LFOUND.
        CALL  RESULT (PROJNAME ).
        GO TO L0601.
```

Fig.5.8　Fig.5.2のQUELの問合せのおよそのプログラム

アクセスオカーランス数の見積りが、現実にどれだけ応答時間短縮に有効かは今後よく検討する必要がある。実際には応答時間は、より物理的ページアクセスに依存するだろう。しかしこれは、LISではなくDBSの内部スキーマレベルの問題である。各層における最適化と、DDBS全体での最適化との関係は今後明らかにしたい。

不等価結合を処理出来る様にシステムの拡張を行なっている。等価結合はLCS上のESR-RSRリンクのトラバースに変換できるが、不等価の場合には集合演算機能が必要となる。結合について、我々はESR-RSRリンクにそったもの以外は許していない。これ以外の結合を許す[Murok80]ことは、LCSがLIS以上の意味を持つことになってしまう。この問題は、スキーマ層のマッピングの本質的問題として、open questionとしたい。

5.2 問合せ分割 [Takim80c,d]

問合せ分割とは、GCS問合せを各LCS問合せに分割するとともに、必要なサイト間処理を行なうことである。

5.2.1 問合せ変形 - 表現の変換

GCSリレーションは、Fig.4.8の様な関係計算形式で定義されている。GCSリレーションを参照するGCS問合せは、問合せ変形[Stonm76]手法を用いてLCSリレーションのみを参照する問合せに変形できる。このためには、GCS問合せは和正規形に正規化されている必要がある。各conjunctに対して、GCSリレーション定義式の各<sub-def>を用いて問合せ変形とインテグリティのチェックが行なわれる。チェックの結果、矛盾があればこのconjunctは除去される。変形された各conjunctごとに問合せが生成され、最終結果はこれらの問合せの結果リレーションの和となる。以下変形された問合せについて考える。

5.2.2 仮定と目標

変形された問合せを、次に各サイト及びサイト間で実行せねばならない。この時ネットワークは主要なパフォーマンスのボトルネックとなる。ネットワークに対して次の単な仮定を設ける。a) point-to-point, b) no queueing delay, c) 通信コスト≫処理コスト, d) 通信コストは距離含む.

この様なネットワークの下で、問合せ処理は次の目標を達成する必要がある。a) 全通信コストの最小化, b) 応答時間の最小化, c) 処理のための必要情報(i.e. 分散情報DI)の最小化と静的化。これまで[HEDNA78, WONG77]等は、選択度に基づいた静的転送スケジュール決定手法を開発してきている。選択度は、属性の値が均等分散すると共に他の属性とは独立であるとの仮定に基づいている。この仮定が現実にマッチするかは疑わしい。又、こうした情報は、スキーマ情報に較べてより動的な性質を持っている。こうした動的情報を各サイトが持つことは、格納オーバヘッドとなるだけではなく、一致性制御のための莫大な通信オーバヘッドをもたらしてしまう。このため我々の提案するアルゴリズムは、この目標達成を最大目的としている。Yaoのシミュレーション研究は彼のものもWongのものもほとんど同一の結果を示し、問合せ処理の主要因はネットワークのqueueingであることを示している。このことは又、問合せ処理の中で最適化できる部分は一部であり、より運用上の合理化が重要であることを示唆している。この意味で、管理情報としてのDIを最小化し静的化する我々のアルゴリズムは有効であると考える。

5.2.3 戦略とTSAアルゴリズム

上述した目標を達成するための我々の戦略は次の様である。

a) 各サイトで独立に処理できる部分をまず処理する(初期ローカル問合せ処理)。
b) ある転送処理の結果は、Ackにのせて GCS 問合せの制御ノードに
c) 制御ノードは、その結果に基づいて、次の転送を決める。
d) ある閾値よりも転送コストの小さいリレーションは他のサイトに積極的に転送する。

a)は、サイト間処理を必要とする部分のみをつくり出し、転送コストを短縮できる。d)によって転送のパラレリズムを高め、応答時間を短縮できる。

この戦略に基づいたアルゴリズム(TSAと呼ぶ)の詳細については、[TAKIN80d]を参照されたい。

5.2.4 議論

Yao等の手法に対して、我々の利点は、第1に必要情報が小さくかつ静的である点である。これは実際の運用において重要な長所となる。又、我々の手法は初期ローカル問合せ処理と転送処理とをオーバラップできる。彼は初期ローカル処理コストを0としているが、実際にはqueueing delayの影響によって無視出来ないものである。我々のTSAの動作は、閾値の値に大きく依存することになる。大きな値は転送の並行度を高め、小さな値はこれを高めることになる。シミュレーションを通しての、アルゴリズムの有効性の検討が今後必要となる。Yaoの研究が示しているように、問合せ処理は、今後queueingの効果、必要情報管理等の運用面での検討が必要であると私は確信している。問合せ分割については、[TAKIN80c,d]に詳論されているので参照されたい。

6. 全体の議論

本論文では、JDDBSにおける全体アーキテクチャ、スキーマ層設計、とアクセスについて論じた。問合せ分割の一部(分散問処理TSA)以外はインプリメントを終了している。問合せ分割全体のインプリメントにはネットワークプロトコルの研究と整備が必要である。DDBS処理のために、ISO/SC16における様な巨大プロトコル体系が必要なのかは疑問に思う。我々は、まずローカルネットワークのレベルでのDDBSの実現と、DDBSから現たプロトコルへの要求を整理していきたいと考えている。我々にとって、同時実行制御等の制御問題は既された問題であるが、統合型DDBSにおいて代表サーバーのexplicitな存在はないと考えている。ここにおける代表性とは、4.2で述べた様に意味論的な共有である。

問合せの処理、i.e. 変換と分割、の最適化は個々独立に論じてきたが、全体的な処理の最適化が必要であるように思う。今後の課題としていきたい。

全体アーキテクチャとして、我々は設計用にE-Rモデル、処理用にリレーショナルモデルを用いた。今後、制御問題をふくめて、DDBSに適したモデルとは何かの検討を、評価を含めて再度行なう必要がある。

現在、JDDBSの実現を進めるとともに、これのオフィス情報システム(OIS)への適用を進めている。

謝辞

JDDBSのインプリメントに協力していただいているシステク社の鈴木 信氏に感謝します。又、JIPDEC 開発部長 山本 功子女史の助言と指導に感謝します。

References

[JDDBS80] "分散型リソース処理の研究開発-分散型データベースシステム," JIPDEC 54-S001, Nov. 1980.

[ADIBM78] "A Distributed DBS Using Logical Relational Machines," 4th VLDB, Berlin, Sept. 1978, 450-461

[ANSIX75] "Interim Report of the Study Group on DBMS," 75-02-08, Feb. 1975

[CODDE70] "A Relational Model of Data for Large Shared Data Bank," CACM, Vol.13, No.6, 1970. 377-387

[CHENP76] "The Entity-Relationship Model-Toward a Unified View of Data," TODS, Vol.1, No.1, 1976. 9-36

[HELAG75] "INGRES - A Relational DBS," AFIPS, 1976, 409-416

[HEDNA78] Hevner, A.R. and Yao, S.B., "Optimization of Data Accesses in DDBS," TR281, Purdue Univ., July 1978.

[STONM76] "Design and Implementation of INGRES," TODS, Vol.1, No.3, 1976, 189-222.

[STONM77] "A Distributed Version of INGRES," Berkeley Workshop, 1977, 19-36

[WONGE77] "Retrieving Dispersed Data from SDD-1," ibid, 217-235

[TAKIM78] Takizawa, M. et al., "Resource Integration and Data Sharing in Heterogeneous Resource Sharing System," Proc. ICCC'78, Kyoto, Sept. 1978, pp. 253-258.

[TAKIM79] Takizawa, et al., "The Four-Schema Structure Concept as the Gross Architecture of Distributed Databases and Heterogeneity Problems," JIP of IPSJ, Vol.3, No.3, Dec. 1979. pp. 134-142

[TAKIM80a] Takizawa, M. et al., "分散型データベースにおける問合せ変換," 第20回シンポジウム, Jan. 1980

[TAKIM80b] Takizawa, M. et al., "Query Translation in Distributed Databases," to appear in Proc. of the IFIP'80, Oct. 1980.

[TAKIM80c] Takizawa, M., "Distribution Problems in Distributed Databases - Integration and Query Decomposition," Proc. JIPDEC Seminar, Feb. 1980

[TAKIM80d] Takizawa, M., "Operational Query Decomposition Algorithm," JIPDEC TR80/04, July 1980

[MURUK80] Muroi, K., Tanaka, Y., "CODASYL DBMSの関係インタフェース," 北大工, 1980