データベースシステム 117−12
（1999. 1. 23）

並列SQLによる大規模PCクラスター上の相関ルールマイニング：C実装プログラムとの性能比較

イコ プラムディオノ　新谷 隆彦　田村 孝之 *　喜連川 優
東京大学生産技術研究所

あらまし

蓄積されたデータベースが大きくなるにしたがってそのデータベースから隠された付加価値を発掘するデータマイニングの重要性が広く認識されるようになった.現在のデータベースの主流は関係データベースシステムであるのでSQLを用いたデータマイニングの実現が広く望まれるが、SQLベースのデータマイニングは専用プログラムに比べて性能の面では劣ることが知られている. ここで大規模なPCクラスター上でSQLベースのデータマイニングの並列化の評価を報告する.通常の相関ルールマイニングでは4ノードを用いた並列化はC言語で書かれるプログラムと同等の性能を示し、階層関係を考慮した相関ルールマイニングの場合でも6 - 8ノードで互角の結果が得られることが分かる.

Parallel SQL Based Association Rule Mining on Large Scale PC Cluster: Performance Comparison with Directly Coded C Implementation

Iko Pramudiono　Takahiko Shintani　Takayuki Tamura †
Masaru Kitsuregawa
Institute of Industrial Science, The University of Tokyo

Abstract

Data mining is becoming increasingly important since the size of databases grows even larger and the need to explore hidden rules from the databases become widely recognized. Currently database systems dominated by relational database and the ability to perform data mining using standard SQL query will definitely ease implementation of data mining. However SQL based data mining performance is known to fall behind specialized implementation. In this paper we presented an evaluation of data mining based on parallel SQL on large scale PC cluster. For flat association rule mining, parallelizing SQL query using 4 processing nodes the performance is even with C based program while in case of generalized association rule with taxonomy, we achieve comparable result with 6-8 processing nodes.

1 序論

データベースから隠された付加価値を抽出することが研究者のみならず、ビジネス界からも注目を浴びている. その背景には近年増え続けるデータベースのデータ量と競争に勝ちぬくためにデータウェアハウスに代表されるような情報の活用が不可欠という認識がある.

データマイニングの代表的な処理は相関ルールの抽出である. 相関ルールとはある対象オブジェクトの間の共起関係を示す. 処理負荷が重いことから効率的な処理を目指していくつかのアルゴリズムが提案されてきており、[1][2][3]、実装方式として専用のプログラムに頼らざるを得ない状況となっている.

一方、解析対象データのほとんどが関係データベースシステムによって管理されていることから関係データベース自身でデータマイニングが行えることが期待される. システムとの親和性とコストの面からそのメリットは大きいと考えられる. 関係データベース問合せ言語 SQL を用いた実装は柔軟性と移植のしやすさというさらなるメリットも秘めている. そのため最近では SQL 及びその拡張を用いたデータマイニングが研究されている [4]. しかし、一般的に性能の面では専用プログラムに大きく劣っていることが知られている.

この課題を克服するために SQL 問合せの改良もさることながら、実行環境の性能も重要な要素となる. 近年の計算機の性能向上ならびに並列処理機能の進歩により、SQL を基にしたデータマイニングでも十分な性能を期待できる.

ここでは我々が開発してきた大規模 PC クラスター上に SQL 処理系を用いたデータマイニングを実装し、直接 C 言語で記述されたデータマイニングプログラムと比較し、関係エンジンの並列化による性能向上率と C 記述から SQL への効率の低下割合を SQL マイニングの有効性を明確にする.

2 SQL による相関ルールマイニング

2.1 相関ルールマイニング

相関ルール (association rule) は, データマイニングで得られる情報の代表的なものの一つである. 簡単な例として「パンとバターを購入した客の 80% は, 牛乳も購入する」というルールがあげられる. ここで 80% はこのルールの確信度と呼ぶ. 他の重要なパラメータはあるアイテム集合の支持度である. この例では { パン, バター, 牛乳 } が一つのアイテム集合をなしている.

アイテム集合 X の支持度 (support) $support(X)$ は $\mathcal{D}$ 全体に対し X を含むトランザクションの割合であり, 次式で求められる.

$$support(X) = \frac{n(X)}{n(\mathcal{D})} \quad (1)$$

また, 確信度 (confidence) $confidence(X \Rightarrow Y)$ は $\mathcal{D}$ の中で X を含むトランザクションのうち, X と Y を共に含むトランザクションの割合, すなわち

$$confidence(X \Rightarrow Y) = \frac{support(X \cup Y)}{support(X)} \quad (2)$$

によって定義される.

相関ルールの抽出問題とは, ユーザによって指定された最小支持度 (minimum support) と最小確信度 (minimum confidence) を満足する全てのルールを見出すことである.

この問題は次の 2 段階の処理で求める.

1. ラージアイテム集合 (large itemset: 最小支持度を満足するアイテム集合) をすべて見つけ出す.
2. ラージアイテム集合から, 最小確信度を満足する相関ルールを導出する.

この内、第 1 段階の処理の負荷が重く効率的なアルゴリズムが求められ、ここで詳細に議論する. なお第 2 段階処理を行う SQL 問合せの一例は [4] で報告された.

2.2 SQL によるマイニングの実装

純粋な SQL を用いた最初のデータマイニングの実装は Houtsma らによって提案された SETM である [3].SIGMOD98 では有名な Apriori アルゴリズムを SQL-92 及びその拡張であるオブジェクト志向 SQL で実装したものは Sarawagi らによって報告されたが、純粋な SQL の実装は性能的に非効率的だという結論を与えている [4].

ここで我々は汎用性の高い純粋な SQL 実装である SETM を選び、並列化によって SQL 化に伴う性能劣化をどれだけ補うことができるかに焦点を当てる.

使用した SQL 問合せを図 1に示す. また比較として Apriori アルゴリズム [2] を実装した C 言語の相関ルールマイニングプログラムを使用する.

*現在、三菱電機 (株) 情報通信システム開発センター
†Communication System Development Center, Mitsubishi Electric

```
CREATE TABLE SALES (id int, item int);

- PASS 1

CREATE TABLE C_1 (item_1 int, cnt int);
CREATE TABLE R_1 (id int, item_1 int);

INSERT INTO C_1
     SELECT    item AS item_1, COUNT(*)
     FROM      SALES
     GROUP BY item
     HAVING COUNT(*) >= :min_support;

INSERT INTO R_1
     SELECT    p.id, p.item AS item_1
     FROM      SALES p, C_1 c
     WHERE      p.item = c.item_1;

- PASS k
CREATE TABLE RTMP_k (id int, item_1 int,
   item_2 int, ... , item_k int)
CREATE TABLE C_k       (item_1 int,
   item_2 int, ... , item_k int, cnt int)
CREATE TABLE R_k       (id int, item_1 int,
   item_2 int, ... , item_k int)

INSERT INTO RTMP_k
     SELECT    p.id, p.item_1, p.item_2, ... ,
                 p.item_k-1, q.item_k-1
     FROM      R_k-1 p, R_k-1 q
     WHERE      p.id       = q.id
       AND      p.item_1 = q.item_1
       AND      p.item_2 = q.item_2
                          .
                          .
                          .
       AND      p.item_k-2 = q.item_k-2
       AND      p.item_k-1 < q.item_k-1;

INSERT INTO C_k
     SELECT    item_1, item_2, ..., item_k,
                 COUNT(*)
     FROM      RTMP_k
     GROUP BY item_1, item_2, ..., item_k
     HAVING COUNT(*) >= :min_support;

INSERT INTO R_k
     SELECT    p.id, p.item_1, p.item_2, ...,
                 p.item_k
     FROM      RTMP_k p, C_k c
     WHERE      p.item_1 = c.item_1
       AND      p.item_2 = c.item_2
                          .
                          .
                          .
       AND      p.item_k = c.item_k;

DROP TABLE R_k-1;
DROP TABLE RTMP_k;
```

図 1: SQL query to mine flat association rules

トランザクションデータは（トランザクション ID, アイテム）という第一正規化の形として保存される. パス 1 での処理はそれぞれのアイテムを数え、最小支持度を満たすものをパス 1 のラージアイテム集合であるテーブル C_1 に入れる. さらにトランザクションデータから最小支持度を満足するトランザクションのみを選び、長さ 1 のトランザクションデータテーブル R_1 に入れる.

他のパスでは長さ k の候補アイテム集合 RTMP_k は長さ k-1 のトランザクションデータをセルフジョインさせることで作成される. そしてそれぞれの候補アイテム集合の支持度を数え上げ、最小支持度を超えるアイテム集合を長さ k のラージアイテム集合 C_k に入れる. 最後にそのラージアイテム集合を用いてアイテム集合が一致する長さ k のトランザクションデータテーブル R_k が作成される.

3 並列実行環境

3.1 ATM 結合 PC クラスタ

PC クラスタ NEDO-100 は, 図 3に示すように 200 MHz Pentium Pro を搭載する PC 100 台を 155 Mbps の ATM および 10 Mbps Ethernet の 2 系統のネットワークで相互結合したシステムである. 各 PC ノードの構成を表 1に示す. ATM スイッチには 128 ポートの UTP-5 インタフェースを持つ日立製 AN1000-20 が採用され, あらかじめ PVC で全てのポート間のコネクションが設定されて用いられている.

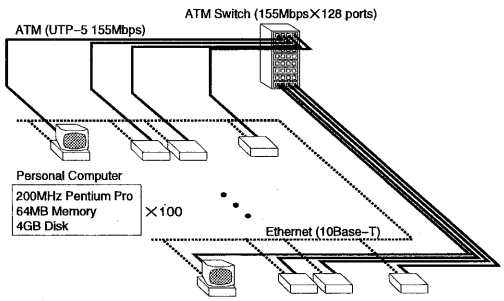

図 2: PC cluster architecture

表 1: Configuration of each PC

CPU	Intel Pentium Pro 200MHz
Chip Set	Intel 440FX
Main Memory	64MB
Disk Drive for databases	Seagate Barracuda (Ultra SCSI, 4.3GB, 7200rpm)
SCSI HBA	Adaptec AHA 2940 Ultra Wide
ATM NIC	Interphase 5515 PCI ATM Adapter
OS	Solaris2.5.1 for x86

3.2 並列関係データベースシステム

100 台の PC クラスタ NEDO-100 で動作する並列関係 データベースシステムである DBKernel が実装されている. 実装においてはカーネルレベルでの実装を目指すものの、可搬性が重視されているため、現在ユーザレベルのスレッドを用い, TCP/IP に

より通信が行われているが, プロセッサの高い性能によりオーバヘッドはほとんど問題にならないといえる.

図 3は DBKernel のアーキテクチャを示す. ユーザからの SQL 問合せはフロントエンドでコンパイルされ、プロセッサのネーティブな実行コードが出力される. 問合せの実行は管理プロセスによって開始され、実行コードが各ノードにブロードキャストされる. 実際の実行は各ノードの実行プロセスが新たなスレッドを作成することによって行われる.

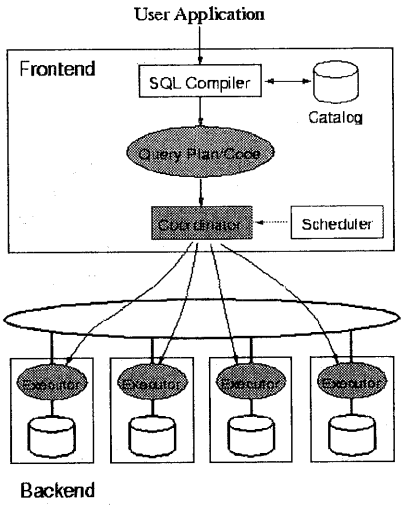

図 3: DBKernel architecture

TPC-D ベンチマークの問合せを実行した結果, 商用並列システムと比較しても極めて高い性能が得られていることが確認されている [6].

4 並列化

4.1 並列化の実行方式

後に実行トレースで示されるように候補アイテム集合作成の処理は各ノードで行われるが、アイテム集合の支持度の集計は 2 段階に分けられる. まず各ノードでアイテム集合のローカル支持度を求め、その結果をある他のノードに送る. そこで全体の支持度がローカル支持度の合計から得られ、最小支持度を満たすラージアイテム集合が決定される. ラージアイテム集合が全ノードにブロードキャストされた後、トランザクションデータ作成の処理がラージアイテム集合を用いて各ノードで行われる.

4.2 データセット

この実験で用いたデータは Apriori アルゴリズムと共にに報告されたプログラムで作成された [2]. そのパラメータは表 2の通りである. データはトランザクション ID によって各ノードに分割される.

表 2: Dataset paremeters

File size	11MB
Number of transactions	200000
Average transaction length	10
Number of items	2000

4.3 性能評価

いくつかの最小支持度に対して実験を行った結果を図 4に示す. このデータセットに対して同様の処理を行う C プログラムと比べれば平均的に 4 ノードで同じ性能が出せることが分かる. 図 5から並列化の効果を示すスピードアップ比は飽和する傾向があるもののかなり理想線に近いと言えよう. 各ノードで処理されるデータの量が小さくなるに従って処理オーバヘッドとネットワーク I/O のコストが相対的に大きくなることによると考えられる.

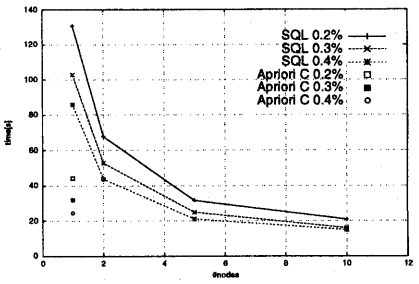

図 4: Execution time(200k)

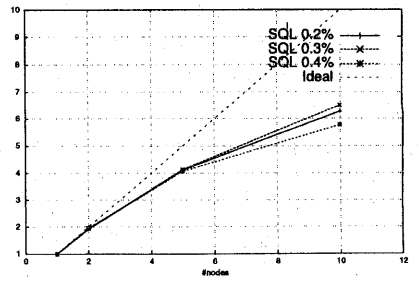

図 5: Speedup ratio(200k)

図6に最小支持度が0.5%の時、パス毎の実行時間が示される. よく知られるように候補アイテム数が最も多いパス2の処理が実行時間の大半を占める.

図7は最小支持度0.5%の時、パス毎のスピードアップ比を示す. この最小支持度では全ての処理を終えるのに8つのパスを要する. 後半のパスでは候補アイテム数が劇的に減少するので並列化オーバヘッドの影響が無視できなくなる.

候補アイテム集合のサイズによって並列度を変えることでパスの多い問合せ実行の高速化を図ることができる. つまり、後半のパスでは実行ノードの数を減らすことでオーバヘッドを減少させることができる. この最適化はこれから検討課題とする.

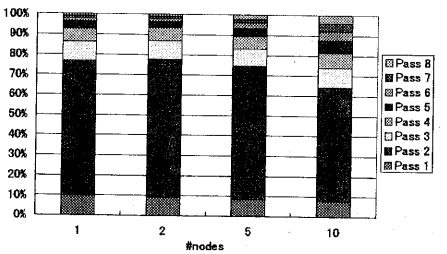

図 6: Passes contribution in exec time (200k 0.5%)

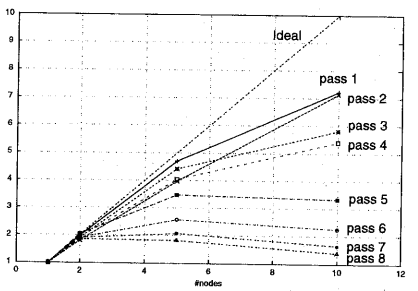

図 7: Speedup ratio analysis on passes(200k 0.5%)

4.4 実行トレース

DBKernelの実行トレース機能を用いて実行中のリソースや処理の詳細を調べることができる. 図8には最小支持度0.5%の時の実行トレースが示される. パス2は3秒から23.5秒までの間を占める. そのうち、候補アイテム集合の作成処理は図中のフェーズ# 1に示されるように3.5秒から12秒の間にある. このフェーズでのハッシュテーブルへのプローブ操作が100%のCPU利用率と低いディスク読み込みスループットとして現れるが、フェーズの後半にジョインの結果をディスクに書き込むため、処理速度がディスクI/O限界に拘束される. プローブ操作のある他のフェーズ、例えばトランザクションデータテーブルの作成処理 (フェーズ #3, 21.5秒~ 23.5秒) やハッシュテーブル更新を伴うローカル支持度の集計処理 (フェーズ #2前半, 16.5秒~ 21.5秒) でCPUの利用率がほぼ100%である. しかし、全体の支持度の集計処理 (フェーズ #2後半, 12秒~ 16.5秒) ではノード同士がローカル支持度を交信するため、かなり大きなネットワークスループットが観察される.

全体として処理速度がCPUの性能によって制限されることが分かる.CPUの性能が改善されつづけるので好ましいものである.

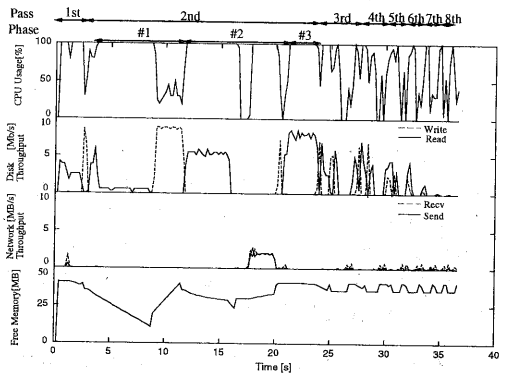

図 8: Execution trace(200k 0.5% 5 nodes)

5 階層構造を考慮した相関ルール

ほとんどの場合, アイテムに対して, 階層的な分類 (taxonomy) が可能である. [5] 例えば, 図9のような分類を考えることが出来る. この例では「寿司は日本の食べ物」及び「寿司は食べ物」という関係が表される. 応用によって特有の知識を持たせることでより興味深いルールを発見することができる.

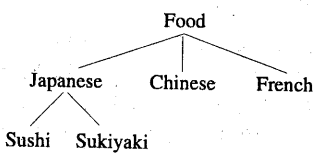

図 9: Taxonomy example

簡便のため、階層構造で上位アイテムに位置するものを祖先 (ANC)、下位アイテムに位置するものを子孫 (DESC) と呼ぶことにする. 分類階層テーブル TAXONOMY は (DESC、ANC) という形を持つ. 上述の例を分類階層テーブルとして表すと表 3のようになる.

表 3: Example of TAXONOMY table

DESC	ANC
Sushi	Japanese Food
Sushi	Food
Sukiyaki	Japanese Food
Sukiyaki	Food
Japanese Food	Food
Chinese Food	Food
French Food	Food

5.1 SQL 問合せ

階層構造を考慮した相関ルールマイニングのアルゴリズムは基本的に通常 (flat) の相関ルールマイニングと変わらないが、全てのアイテムの祖先も加えなければならないため、処理時間が増大する. しかし階層の性質を利用して処理量を減らすこともできる. ここでは次の最適化を図っている：

1. 祖先と子孫の両方を含むアイテム集合を取り除く.
   祖先と子孫を含むルールは常に正しいので冗長である. トランザクションデータを絞り込む.
2. パス 1 ラージアイテム集合に含まないアイテムの祖先を分類階層テーブルから削除する.
   明らかに後の処理に使用されないので削除できる. 例えば「寿司」というアイテムがパス 1 のラージアイテム集合になければ {Sushi、Japanese Food} と {Sushi、Food} を分類階層テーブルから削除できる. この最適化によって分類階層テーブルを百分の一に小さくできる. 数パスに渡る長い処理に対して効果がある.

階層構造を考慮する相関ルールマイニングの SQL 問合せを図 10に示す. パス 1 で分類階層テーブルを用いてそれぞれのアイテムの祖先を全てトランザクションデータに加える. そしてトランザクションデータを絞り込む時に再び分類階層テーブルを用いるが、第 2 最適化に従い、パス 1 のラージアイテム集合に含まれるアイテムだけからなる分類階層テーブルのサブセットであるテーブル TAX_H を作成する.

```
CREATE TABLE SALES (id int, item int);

- PASS 1

CREATE TABLE C_1 (item_1 int, cnt int);
CREATE TABLE R_1 (id int, item_1 int);

CREATE TABLE TAXONOMY (desc int, anc int);

INSERT INTO RXTD
     SELECT    p.id, p.item
     FROM      SALES p;

INSERT INTO RXTD
     SELECT DISTINCT p.id, p.anc
     FROM      SALES p, TAXONOMY t
     WHERE     p.item = t.desc;

INSERT INTO C_1
     SELECT    item AS item_1, COUNT(*)
     FROM      RXTD
     GROUP BY item
     HAVING COUNT(*) >= :min_support;

INSERT INTO TAX_H
     SELECT       t.desc, t.anc
     FROM         C_1 c, TAXONOMY t
     WHERE        t.desc = c.item_1

INSERT INTO R_1
     SELECT    p.id, p.item AS item_1
     FROM      RXTD p, C_1 c
     WHERE     p.item = c.item_1;

- PASS k
CREATE TABLE RTMP_k (id int, item_1 int,
   item_2 int, ... , item_k int)
CREATE TABLE C_k       (item_1 int,
   item_2 int, ... , item_k int, cnt int)
CREATE TABLE R_k       (id int, item_1 int,
   item_2 int, ... , item_k int)

INSERT INTO RTMP_k
     SELECT    p.id, p.item_1, p.item_2,
                  ... , p.item_k-1, q.item_ k-1
     FROM      R_k-1 p, R_k-1 q
     WHERE     p.id       = q.id
       AND     p.item_1 = q.item_1
       AND     p.item_2 = q.item_2
                          .
                          .
                          .
       AND     p.item_k-2 = q.item_k-2
       AND     p.item_k-1 < q.item_k-1
       AND (p.item_k-1, q.item_k-2)
         NOT IN (SELECT    *    FROM TAX_H t
           WHERE     t.anc = p.item_k-1
             AND     t.desc = q.item_k-1);

INSERT INTO C_k
     SELECT    item_1, item_2, ..., item_k,
                  COUNT(*)
     FROM      RTMP_k
     GROUP BY item_1, item_2, ..., item_k
     HAVING COUNT(*) >= :min_support;

INSERT INTO R_k
     SELECT    p.id, p.item_1, p.item_2, ...,
                  p.item_k
     FROM      RTMP_k p, C_k c
     WHERE     p.item_1 = c.item_1
     AND       p.item_2 = c.item_2
                          .
                          .
                          .
     AND       p.item_k = c.item_k;

DROP TABLE R_k-1;
DROP TABLE RTMP_k;
```

図 10: SQL query to mine generalized association rule

他のパスでは候補アイテム集合を作成する際、第1最適化を施す. 分類階層テーブルのサブセット TAX_H を用いて絞り込む.

5.2 データセット

実験で用いられるデータは前節と同様のプログラムで表4にあるパラメータで作成された. トランザクションデータは各ノードに分割されるが、分類階層テーブルは各ノードに複製が保持される.

表 4: Dataset paremeters

File size	4MB
Number of transactions	50000
Average transaction length	5
Number of items	10000
Number of roots	100
Number of levels	3-4
Average fanout	5

5.3 性能評価

図11は二つの最小支持度に対する実行時間である. 分類階層を考慮する相関ルールマイニングの SQL 実装はそれの C 実装に比べて 6 ～ 8 ノードで並列化することで十分匹敵する処理速度を実現できることが分かる. 分類階層テーブルを処理する必要があるため、並列化オーバヘッドも大きくなり効率が若干落ちる.

図12はスピードアップ比を示す.10 ノード以上のノードでは性能がやや飽和する. 原因は各ノードのデータサイズが小さく並列化で得られるゲインはオーバヘッドによって打ち消されると考えられる.

そこで前節で述べた後半の実行の最適化も有効であろう.

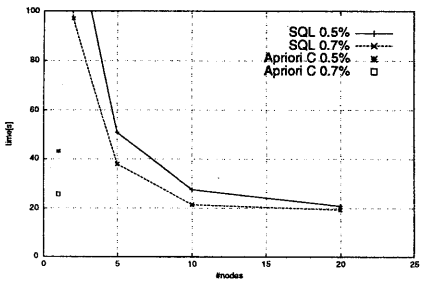

図 11: Execution time(50k with taxonomy)

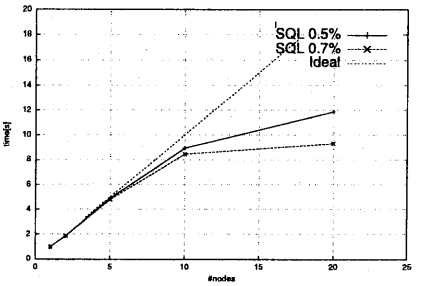

図 12: Speedup ratio(50k with taxonomy)

5.4 実行トレース

実行トレースを図13に示す. 分類階層を考慮する相関ルールマイニングの特徴として、処理時間の大半がアイテム集合の絞り込みを含む候補アイテム集合の作成処理 (フェーズ # 1 ,3 秒～ 17.8 秒) に費されることが分かる. このフェーズではディスク読み出しを伴うハッシュテーブル作成とプローブ及び結果のディスクへの書き込みが 2 回行われる.

支持度の集計処理 (フェーズ #2, 17.8 秒～ 21.5 秒) 及びトランザクションデータテーブルの作成処理 (フェーズ #3, 21.5 秒～ 24.6 秒) は前節で述べた通常の相関ルールマイニングと同様である.

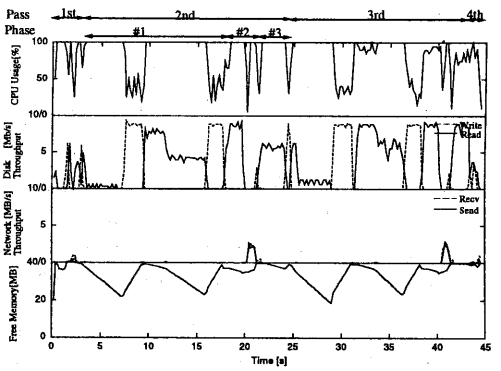

図 13: Execution trace(50k 0.5% 5 nodes)

6 まとめ

ここで C によるデータマイニングのプログラムと SQL による実装の比較を報告した. SQL が常に普及し、度重なる改良によって実行環境も整っているので並列 SQL システムによってかなりの性能が出せると考えられる.

実際の実装を通じて、通常の相関ルールのマイニングに対して4ノードで階層関係を考慮するマイニングでも6～8ノードを用いればCによる実装と同じぐらいの処理速度が実現できることを確認した. このアプローチで専用のプログラムを用意する必要がなくなり、どの関係データベースシステムにも適用できるという利点は明らかである.

現在の並列SQLは高価な超並列計算機に実行されているが、将来より経済的なPCクラスターがその役割を担うことになるだろう. その環境に適した方式が望まれる.

これからの研究の展開としてはより効率的なアルゴリズムの開発や大規模なデータを利用し並列処理システムの典型的な問題であるデータの偏りの対処を検討する.

参考文献

[1] R. Agrawal, T. Imielinski, A. Swami. Mining Association Rules between Sets of Items in Large Databases. In Proc. of the ACM SIGMOD Conference on Management of Data, 1993.

[2] R. Agrawal, R. Srikant. Fast Algorithms for Mining Association Rules. In Proc. of the VLDB Conference, 1994.

[3] M. Houtsma, A. Swami. Set- oriented Mining of Association Rules. In Proc. of International Conference on Data Engineering, 1995.

[4] S. Sarawagi, S. Thomas, R. Agrawal. Integrating Association Rule Mining with Relational Database Systems: Alternatives and Implications. In Proc. of the ACM SIGMOD Conference on Management of Data, 1998.

[5] R. Srikant, R. Agrawal. Mining Generalized Association Rules. In Proc. of VLDB, 1995.

[6] T. Tamura, M. Oguchi, M. Kitsuregawa. Parallel Database Processing on a 100 Node PC Cluster: Cases for Decision Support Query Processing and Data Mining. In Proc. of SC97: High Performance Networking and Computing, 1997.