C5-2

SAN結合PCクラスタ上での動的資源割り当てを用いた並列データマイニング処理

合田 和生† 田村 孝之‡ 小口 正人§ 喜連川 優†

†東京大学生産技術研究所
‡三菱電機株式会社
§中央大学研究開発機構

概要

次世代のデータベースプラットフォームとしてPCクラスタが注目されているが、並列分散システムの元ではノードごとの負荷の偏りが並列効果を阻害することが古くから知られ、研究が行われてきた。特にPCクラスタではノード数が膨大で異質のノードが混在することが想定されるため、実運用の為には非均質環境に於ける効果的な負荷分散機構が求められている。一方、従来利用されて来たPCクラスタではストレージがノードに従属するShared Nothing方式のディスクアクセスが主流であったが、次世代のストレージ技術として注目されているStorage Area Network(SAN)の登場により、Shared Disk方式のアクセス方法が検討されている。そこで、本論文では並列相関ルール抽出処理を例に、SANを適用したPCクラスタにおける動的負荷分散機構に付いて設計および実装による評価実験を行う。さらに、未利用のCPUおよびディスク資源を含めて負荷分散を行うことで、実行時に系の性能限界を判断してCPUパワーおよびI/O帯域を動的に拡張する機能を実現し、評価実験を行う。いずれのケースにおいても従来のShared Nothing方式のPCクラスタよりも高い性能が得られ、SAN結合PCクラスタの有効性が示された。

Parallel Data Mining with Dynamic Resource Allocation on SAN-connected PC Cluster

†GODA Kazuo, ‡TAMURA Takayuki, §OGUCHI Masato and †KITSUREGAWA Masaru

†Institute of Industrial Science, The University of Tokyo
‡Mitsubishi Electric Corporation
§Research and Development Initiative, Chuo University

Abstract

The PC cluster system is becoming remarkable as a next-generation database platform. But on such a parallel distributed system, the load skew across nodes aggravates the parallel efficiency. Especially the PC cluster system has lots of various nodes, and then efficient load balancing technology in heterogeneous environment is required for practical use. In addition, although shared-nothing storage access has been major in the conventional PC cluster system, today shared-disk access is being researched and developed since Storage Area Network(SAN) came on stage. In this paper, parallel association rule mining is picked up and we designs and implements dynamic load balancing system with idle nodes and unused disk drives on our SAN-connected PC cluster, so that resources such as CPU power and I/O bandwidth can be dynamically expanded. We show the experimental evaluation, which results in the advantage of SAN-connected PC Cluster.

1 はじめに

近年 PC クラスタはコストパフォーマンスの面から、次世代の大規模並列計算機システムの中心的な役割を果たし、多くのデータベースアプリケーションのプラットフォームとなるべく注目されている。一方、並列分散システムの下では、演算処理にともなうノード間の負荷の偏りが並列効果を阻害することが古くから知られ研究されて来た。データベースにおけるスキューに関しては、Walton[3] らによって分類がなされて、近年問い合わせ実行時に動的に負荷分散処理を行う機構の研究 [13][9] が行われて来た。

ところで従来利用されてきた PC クラスタでは、全てのストレージドライブが各ノードに従属する Shared Nothing 方式が主であったが、次世代のストレージ技術として注目されている SAN(Storage Area Network) の登場により、共有ディスク (Shared Disk) 方式のディスクアクセス手法が検討されている。また PC クラスタシステムが普及するに伴い、資産の効率的利用という側面から、高速なノードと低速なノードが混在した環境での負荷分散処理や、処理を行っていない未利用資源の利用を実行時に行う機構が求められている。

そこで、本論文ではデータベースアプリケーションの一つである、データマイニングアプリケーションにおける相関ルール抽出処理を例に、SAN を適用した PC クラスタにおけるストレージ仮想化機構を提案し、それを用いた動的負荷分散機構の設計、実装を行う。非均質環境においてノード間の負荷バランスを調節することによりボトルネックを除去し並列効果を高めると共に、さらに未利用のノードやディスクを含めて負荷分散を行うことで、実行時に動的に CPU パワーや I/O 帯域を拡張する機能を実現した。

本論文では先ず、第 2 章において SAN 技術と実験に用いた SAN 結合 PC クラスタシステムに付いて述べ、第 3 章においてデータマイニングにおける相関ルール抽出処理を解説する。第 4 章ではストレージ仮想化機構と負荷分散機構の設計を行いその評価実験から有効性を確認する。その後、第 5 章では負荷分散機構を利用してノード数の増加による CPU パワーの動的投入と動的デクラスタリング [14] による I/O 帯域の拡張を併用することで、系のバウンディングファクタにより CPU およびディスク双方の資源を動的に割り当てる実験を行う。最後に第 6 章でまとめを行う。

2 SAN と SAN 結合 PC クラスタ

2.1 Storage Area Network(SAN)

SAN(Storage Area Network) は 1 台のホストコンピュータと従属的な関係を保って来たストレージデバイスを、専用のネットワークによって N:N に結合するものである。従来ストレージの共有は IP ベースの LAN 経由で NFS(Network File System) 等の NAS(Network Attached Storage) 方式を用いて行われており、LAN の輻輳の原因となっていた。

特にデータベースでは巨大なシーケンシャルファイルをたびたび扱うため、NAS 方式のストレージ共有は、TCP/IP のプロトコルソフトウェア処理による CPU 負荷が無視できない程となる。一方、表 1 の比較が示す通り、LAN(Gigabit Ethernet) では 1500B 毎に CPU が介在する必要があるが、現行 SAN で支配的な FC(Fibre Channel) では CPU の I/O 命令一つで 128MB のデータを扱えるなど、シーケンシャルデータの通信に優位な設計がなされている為、ストレージアクセスの CPU コストを低く抑えることができる。

現在、SAN では仮想化 (Virtualization[2]) と呼ばれる技術が注目されている。仮想化は接続された複数の多様なストレージデバイスから論理的なオブジェクトを構成することで、システムの性能改善や管理の効率化を図る技術である。例えば、VERITAS[10] はそのファイルシステムで複数のネットワークでまたがった物理ディスクを一つの論理ボリュームに見せる仮想化を行っている。また、Vicom の Service Virtualization Router[8] や DataCore の SAN Symphony[6] は FC ネットワーク上のルータやスイッチにおける In-Band 型のソリューションである。一方、Compaq の VersaStor[4] の様な HBA(Host Bus Adapter) における仮想化技術も見られる。現在これら汎用の技術は、ストレージ管理コストの削減をターゲットにしているが、今後 iSCSI[5] の登場によりストレージのネットワーク化が進み、多様

表 1: SAN と LAN の設計比較 [11][12]

	SAN	LAN
Media	Fibre Channel	Gigabit Ethernet
Throughput	1.063Gbps (2-10Gbps)	1.25Gbps (-10Gbps)
Transfer unit	frame/sequence/exchange (128MB/sequence)	frame (1500B/frame)
Error control	Hardware CRC	-
Routing	static	dynamic/static

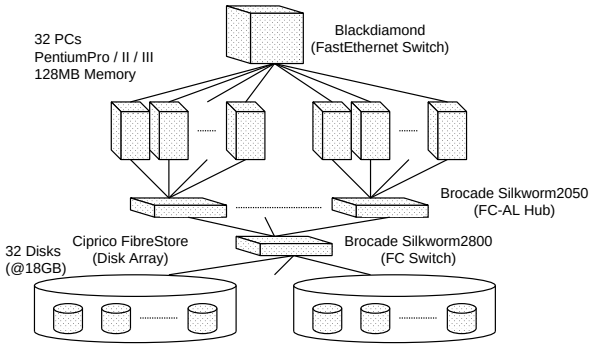


図 1: SAN 型 PC クラスタシステムの所元

表 2: 機器の設定

PC ノード共通性能

Main Memory	128Mbytes
OS	Solaris 8 for x86
FastEthernet NIC	Intel EtherPro/100+
Fibre Channel HBA	Emulex LP8000

PC ノード別性能

CPU	Chipset
Pentium III 800MHz	Intel 440BX AGPset
Pentium II 450MHz	Intel 440BX AGPset
PentiumPro 200MHz	Intel 440FX

な環境で様々なサービスを提供する仮想化が登場すると見られている。

2.2 SAN 結合 PC クラスタ

実験システムとして 32 ノードの PentiumPro/II/III を搭載した PC および FC ディスクアレイを FastEthernet および Fibre Channel によって接続した SAN 結合 PC クラスタを構築してある。システムの概要を図 1 に、所元は表 2 に示す。

3 並列データマイニング

3.1 相関ルール

本資料ではデータベースアプリケーションの一つとして、相関ルール抽出処理を対象とする。相関ルールは以下の様に定義される。アイテム集合を $I = \{i_1, i_2, \cdots, i_m\}$、トランザクションデータベースを $D = \{t_1, t_2, \cdots, t_n\}(t_i \subseteq I)$ とした際に、各要素 t_i をアイテム集合と呼び、k 個の組合せの物を長さ k のアイテム集合と呼ぶ。相関ルールは $X \Rightarrow Y$ で表現され、$X, Y \subseteq I, X \cap Y = \emptyset$ を意味する。相関ルールは支持度 $sup(X \Rightarrow Y)$ と確信度 $conf(X \Rightarrow Y)$ の二つのパラメータを持ち、前者は D 全体に対し、X と Y をともに持つ確率 $Prob(X \cap Y)$ と、後者は更に $Prob(X \cap Y)/sup(X)$ と定義される。

相関ルール抽出処理はトランザクションデータベース D に対して支持度 $sup(X \cap Y)$ の最小値および確信度 $conf(X \Rightarrow Y)$ の最小値が与えられた際に、これらを満足するルールを見出すことである。この処理は以下の二つのステップによって行われる。

1. 与えられた最小支持度を満たすアイテム集合 (ラージアイテム集合) を全て抽出する。
2. 得られたラージアイテム集合から最小確信度を満たす相関ルールを得る。

この第 2 ステップは限られた個数のルールをフィルタする処理であるため、比較的軽負荷の処理であるのに対し、第 1 ステップは巨大なトランザクションデータベースを繰り返し検索し支持度を調査するため、非常に多くの時間を必要とする演算である。このため、相関ルールの抽出アルゴリズムは第 1 ステップの効率化に焦点を当てている。

3.2 相関ルール抽出処理の並列処理

この相関ルール抽出の代表的なアルゴリズムとしては、IBM アルマデン研究所の Agrawal[1] による Apriori が良く知られている。Apriori では k 個のアイテムの組合せを k-itemset、長さ k のラージアイテム集合を L_k、長さ k の候補アイテム集合を C_k とする。そこで、$k(\geq 2)$ に対して以下の処理を行う。

1. L_{k-1} から C_k を作成する。
2. トランザクションデータベース D を検索し、各候補アイテム集合の支持度を計測する。
3. C_k の中から最小支持度を満足するものを取り出し、L_k とする。

上記で長さ L_k を求める手続きをパス k と呼び、L_k が空になるまで、繰り返される。

本論文では上記 Apriori をもとに提案された並列アルゴリズム HPA(Hash Partitioned Apriori)[7] を利用する。HPA ではクラスタ内の各ノードに SEND プロセ

スおよび RECV プロセスの二つのプロセスを配置し、L_k が空になるまで以下の手順でパス k を繰り返す。

1. SEND プロセスは前パス $k-1$ で得られたラージアイテム集合 L_{k-1} を基に長さ k の候補アイテム集合 C_k を作成し、ネットワークを介してハッシュ分散により RECV プロセスに送信し、ハッシュ表に挿入する。

2. SEND プロセスはディスク上のトランザクションデータベースから長さ k のアイテムの組合せを逐次作成し、ハッシュ分散により RECV プロセスに送信する。RECV プロセスではハッシュ表を用いて候補アイテムを数え上げを行う。

3. トランザクションデータベース全てが検査された後、RECV プロセスは最小支持度を満たすラージアイテム集合 L_k を求め、全 SEND プロセスにブロードキャストする。

上記で計算機に対しては 2. に於ける SEND の長さ k のアイテムの組合せ作成処理と RECV のハッシュ表検索が大きな負荷となりえ、特に最も多くの候補アイテム集合が発生するパス 2 が最も CPU に負荷を与え、マイニング実行時間の大半を占める。対して、パス 3 以降の後半のパスは徐々に CPU への負荷を失い、結果、処理はディスクの I/O 性能に依存する。この性質から、HPA では第一にパス 2 の実行時間を改善を行い、続いて後半のパスのディスク I/O 性能を向上させることが全体の高速化に結びつく。

4 SAN を利用した動的負荷分散機構

4.1 SAN を利用した負荷分散機構の概要

並列データマイニングを先述の SAN 結合 PC クラスタシステム上で実装するため、従来 Shared Nothing に於いて行われて来た負荷分散機構 [9] をもとに負荷分散機構の設計を行う。図 2 に本システムの論理構成を示す。

図中マイニングノード (node[0-2]) の HPA App の部分においてマイニング処理を行う SEND/RECV プロセスが動作する。SEND プロセスは SAN 上のトランザクションデータベースにアクセスする際、下

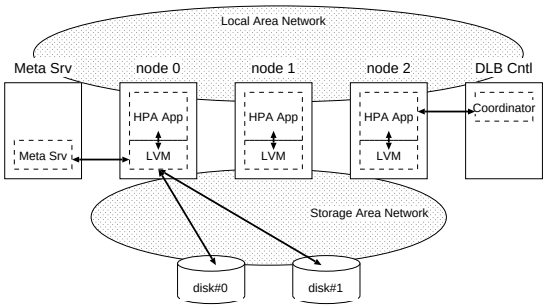


図 2: SAN 環境に於けるデータマイニングアプリケーション制御モデル

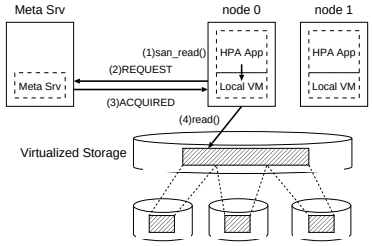


図 3: メタサーバを用いたディスクアクセス手続き

層の LVM(Logical Volume Manager) を用いてアクセスを行う。LVM はアプリケーションから呼ばれる API(Application Programming Interface) の形で実装されており、別ノードに用意する SAN メタサーバ (Meta Srv) と協調する形で、通常のディスクアクセスシステムコール (`read()` 等) に代わるサービスを提供する。本論文ではクラスタ内部のストレージデバイス、SAN メタサーバおよび LVM を合わせてストレージ仮想化機構を呼ぶ。一方、負荷分散コントローラ (DLB Cntl) は、HPA のプロセスと協調して負荷情報を収集し、負荷均衡化の為のマイグレーションプランニングやトリガ実行等のアプリケーション固有の知識を必要とする負荷分散処理を取扱う。

4.2 ストレージ仮想化機構

SAN 環境に於いてはストレージデバイスが複数のホストコンピュータによって共有される。このため、ホストコンピュータ間でディスクアクセスのメタ情報の一貫性を保持する必要がある。そこで、本論文のシステムでは先述の SAN メタサーバがメタ情報の一元管理を行い、一貫性の保持を行うとともに、システム全体の I/O アクセスの管理を行う。

図 3 にその具体的な動作としてディスク読み出し処

理の例を示して説明する。まず、HPA アプリケーションからはじめて san_read() が呼び出されると、LVM はメタサーバに問い合わせを行う。メタサーバは自身の管理しているメタ情報に基づいてプランを立てて、SAN デバイス上の領域をある粒度でマイニングノード用にロックを行い、その領域のデバイス情報やオフセット情報などのメタ情報をマイニングノードに通知する。通知を受け、マイニングノードの LVM は SAN デバイスへの read() を行い、必要なデータを取得し、上位の API 呼び出し元に受け渡す。2 回目以降の san_read() 呼び出しに対しては、自身がロックしている領域が残っている場合はそのままデバイスへの read() を行い、領域が枯渇した場合は改めてメタサーバへの問い合わせを行う。

この機能により SAN 上の複数ディスクのファイルは、複数のホストコンピュータから仮想的に一つのファイルとして共有されることになる。このため、ディスク間のデータ量の偏り、例えば関係データベースに於ける TPS(Table Placement Skew)[3] や実行中の CPU 負荷によって発生したデータ量の差異等は、ストレージ仮想化機構によって吸収することができる。

4.3 負荷分散コントローラ

並列分散環境におけるアプリケーションの負荷調整はメモリ上のデータ構造やスレッドのマイグレーションによって行われるが、Shared Nothing 環境においてはノード間の実行時間の差異を解消するため、ディスク上のデータ量を考慮してマイグレーション量を決定する必要があった。しかし、メモリ上のデータ構造やスレッドのマイグレーションのマイグレーションによるディスク間データ量の調整はスキューが小さい場合は可能であるが、大きくデータ量がひずんだ場合には必ずしも十分でなかった。[9] ではさらにディスク上のデータを LAN 経由で読み出すことでデータスキューを解消する方式 (Transaction Migration) を導入していた。しかし、Transaction Migration を利用してもなお、未利用のノードに処理を拡張するケース等に於いては、ネットワークトラフィックが偏るなどに理由により十分対応しきれていなかった。

本論文ではストレージ仮想化機構により、負荷分散コントローラはデータ量の差異を考慮する必要がない。そのため CPU バウンド領域に於いて、それぞれの時刻に於ける各ノードの CPU 負荷が、相互にボトルネックにならないように調節することで済む。HPA に於いては RECV プロセスの優先度が高く、特定のノードで RECV プロセスのみが CPU パワーを使いきってしまう場合、RECV プロセスは他のノードから送られてくるデータを十分処理しきれておらず、結果全ノードの性能を低下させていると考えられるからである。

このため、負荷分散コントローラの制御方式は以下の様に定式化することができる。

ノード i に於けるノード全体、RECV, SEND プロセスの CPU 利用率をそれぞれ L_i, L_{RECVi}, L_{SENDi} とする。L_{RECVi} は RECV プロセスが受信するデータ流量 (V_{Ri}) に比例し、L_{SENDi} は SEND プロセスが送信するデータ流量 (V_{Si}) に比例するため、以下のように表すことができる。

$$\begin{aligned} L_i &= L_{RECVi} + L_{SENDi} \\ L_{RECVi} &= \alpha\gamma_i V_{Ri} \\ &\sim \alpha\gamma_i C_i \sum V_{Si} \\ L_{SENDi} &= \beta\gamma_i V_{Si} \end{aligned}$$

ここに、α, β は RECV, SEND の処理の負荷係数、γ_i は CPU 係数、C_i は RECV プロセスのハッシュテーブル重み係数とする。添字 i はノード番号を表す。また、

$$L_i \leq 1$$

の条件を満たす必要がある。

この時、系内で RECV プロセスがボトルネックにならない為には全ノードで $L_{RECVi} < 1 - \epsilon$[1]が満たされるべく制御を行う必要がある。このため、負荷分散コントローラは以下の手続きを定期的に実施する。

1. 負荷分散コントローラはマイニングノードから統計情報を収集する。

2. 得られた統計情報 (L_{RECVi}) から、各ノードの $\alpha\gamma_i$ を算出する。$L_{RECVi} < 1 - \epsilon$ を満たさないノードが存在する場合、マイニングノード間でのハッシュテーブルの再配置 (Candidate Migration) が必要であると判断し、C_i を操作変数として、全ノードの L_{RECVi} がほぼ均衡化される方向にマイグレーション計画を建てる。

[1] ϵ は本論文の実験では 5-10%程度取っている。

表 3: ノード/ディスクデータの配置

ノード配置

Case	Node 0	Node 1	Node 2	Node 3
c0	PentiumIII 800MHz	PentiumIII 800MHz	PentiumIII 800MHz	PentiumIII 800MHz
c1	PentiumIII 800MHz	PentiumII 450MHz	PentiumII 450MHz	PentiumII 450MHz
c2	PentiumIII 800MHz	PentiumIII 800MHz	PentiumIII 800MHz	PentiumII 450MHz
c3	PentiumIII 800MHz	PentiumIII 800MHz	PentiumIII 800MHz	PentiumPro 200MHz

データ配置

Case	Disk 0	Disk 1	Disk 2	Disk 3
d0	1,000,000 (84MB)	1,000,000 (84MB)	1,000,000 (84MB)	1,000,000 (84MB)
d1	200,000 (16.8MB)	200,000 (16.8MB)	200,000 (16.8MB)	3,400,000 (285.6MB)

Number of transactions / Size of data volume

3. 負荷分散コントローラが全マイニングノードにマイグレーション計画を通知し、それを受け取ったマイニングノード間でハッシュラインのマイグレーションを行う。

4.4 CPU バウンド領域での評価実験

4.4.1 4 ノード非均質環境

上記の負荷分散機構を評価するために、表 3 の PC ノードの組合せ、トランザクションデータベース配置を用意した。この時、トランザクションデータベース内のアイテム数は 5000 とし、1 トランザクションあたりの平均アイテム数は 20 とした。また最小支持度は 0.7%とした。ディスクアクセスのバッファサイズは 64KB、SAN メタサーバのロックを行う粒度は 512KB とした。

各 CPU スキューおよびデータスキューの組合せに対し、SAN 環境下で Candidate Migration を無効にした場合、有効にした場合それぞれのケースのパス 2 の実行時間を測定した。表 4 に示す。この時、CPU/ディスク共スキューのない、c0d0 に於ける Shared Nothing で無制御のケースを 100 とし、必ずしも正確な比較にはならないが CPU クロック数を以って正規化を行った。比較のために [9] で述べられている Shared Nothing 環境に於ける負荷制御手法を検証実験した結果を併せて掲載する。表 4 で SAN Enable が SAN 適用の負荷分散、SAN Disable が [9] で述べられている Shared Nothing 下の負荷分散を表す。[9] 方式ではディスク上のデータ量を評価関数に含めて Candidate Migration を行い、解

表 4: パス 2 の実行時間 (正規化)

Case	SAN	Contorol	d0	d1
c0	Disable	-	100	170
		Cand.	100†	136
		Cand.+Trans.	100†‡	136‡
	Enable	-	98	102
		Cand.	98†	102†
c1	Disable	-	100	173
		Cand.	94	126
		Cand.+Trans.	94‡	126
	Enable	-	93	95
		Cand.	93†	95†
c2	Disable	-	129	223
		Cand.	104	158
		Cand.+Trans.	104	155
	Enable	-	108	112
		Cand.	97	100
c3	Disable	-	220	395
		Cand.	121	270
		Cand.+Trans.	113	259
	Enable	-	175	176
		Cand.	98	99

決できない場合は LAN 経由でディスク上のデータの均衡化を図る (Transaction Migration)。このため、SAN, Shanred Nothing 間で Candidate Migration の定義は異なる。† は Candidate Migration、‡ が Transaction Migration のトリガが引かれなかったことを表す。

データスキューに関しては、d0 と d1 の比較から、SAN の適用により実行時間の悪化を回避していることが分かる。つまり、SAN の適用によってデータスキューは容易に解決できることが分かる。一方、CPU のスキューに関しては、上記中では c2,3 のケースが、一台の低性能ノードが全体のボトルネックとなる苛酷なケースであるが、SAN 環境下では Candidate Migration と併せて適切に負荷制御が行われていることが分かる。

また、c1 のような緩やかな CPU スキューに関しては、Candidate Migration の必要がなくとも負荷制御が可能であることが分かる。c1d0 で SAN 下で Candidate Migation を行わないケースの実行トレースを図 4 に示す。この図では下から Node [0-3] の各リソースを表し、太線は SEND のディスクリードスループット、点線は SEND の CPU 利用率 L_{SENDi}、実線は SEND と RECV の CPU 利用率 $L_{SENDi} + L_{RECVi}$ を表す。時刻約 10-300 秒がパス 2 である。ハッシュテーブルは等量分散されているので、相対的に速い CPU を持つ Node 0 の L_{RECVi} は少なくなるが、余剰 CPU パワーを使用して SEND が多くのデータ処理を行っていることが分かる。Node [1-3] の低速ノードでは CPU パワーのほとんどを RECV 処理が使用しているが、独

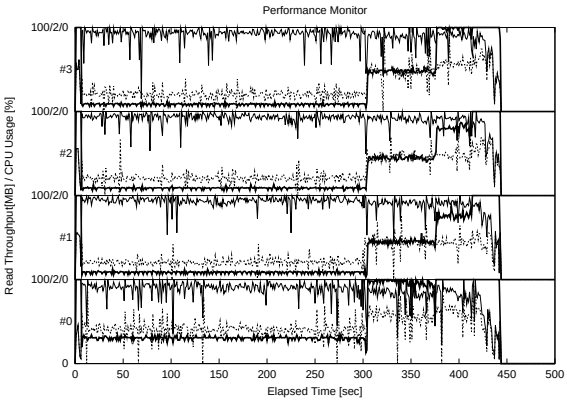
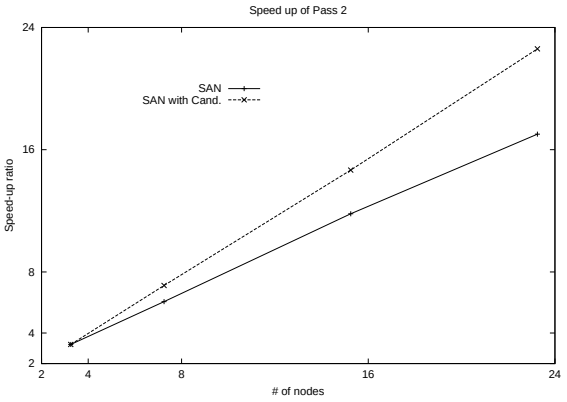


図 4: 実行トレース (c1d0 ケース, SAN 適用)



PentiumIII(800MHz)*$N-1$ 台+PentiumII(200MHz)*1 台の組合せ (横軸は 800MHz を 1 としてクロック周波数で正規化)

図 5: パス 2 のスピードアップ曲線

占はしていないので Node 0 に対してボトルネックにならず、全ノードの CPU パワーが有効に利用されている。c1 のようなケースに於いてはアプリケーションが専用の負荷分散機構を有しない場合でもストレージ仮想化機構のみによって自動制御が可能であることが示された。

4.4.2 多ノード環境

次にスケーラビリティを調査するために、先の d0 のディスク配置の元で、「PentiumIII(800MHz) * N 台 + PentiumII(200MHz) * 1 台」の組合せを用意し、パス 2 の実行時間を計測した。スピードアップ曲線を図 5 に示す。この図に於いては横軸のノード数は PentiumIII(800MHz) を 1 とし、クロック周波数で正規化を行ったものを用いている。

図 5 から非常に苛酷なケースに於ける多ノード環境に於いても、ほぼ線形に近いスピードアップを呈していることから、提案手法の有効性が示された。

5 動的資源投入

5.1 ノード数の増加による CPU パワーの動的投入

PC クラスタは多数のノードを有しているが、アプリケーションの要請によって必要なリソースを適宜決定する必要がある。例えば HPA アプリケーションに於いてはパス 2 が非常に多くの CPU パワーを消費し、実行時間全体に対して支配的である。この場合に、クラスタシステムの利用していないノードに処理を拡張し、CPU パワーを増加することが出来れば、実行時間の改善が出来る。

しかし、従来 Shared Nothing 環境ではデータがノードに依存していたため、ノード数の変更に関してはディスク上のデータの再配置や [9] に於ける Transaction Migration のように非常にコストのかかる制御が必要であった。しかし、それを用いてもなお、前章の評価結果が示すように、負荷分散性能には限界が見られた。一方、SAN 環境ではストレージとノードは独立であるため、柔軟な CPU パワーの投入が可能である。ここでは、先述のストレージ仮想化機構とその上での負荷分散機構を利用して、実行時に追加的にノードを投入し、CPU バウンド時の実行時間を短縮させる制御を行う。

CPU パワーの動的投入は負荷分散コントローラの機能として設計する。まず、アプリケーション開始時には利用が想定される全てのノードにアプリケーション駆動の為のプロセスを生成しておく。そのうち当初は少数ノードのみが HPA アプリケーションの処理を行う。その後は以下の手続きを繰り返し行う。

1. 負荷分散コントローラはマイニングノードから統計情報を収集する。
2. 得られた統計情報から、系全体が CPU バウンドしていると判断され、投入可能なノードが存在する場合、新規投入ノードに投入処理を開始するよう通知する。
3. 通知を受けたノードでは、SEND プロセスが直ち

に SAN メタサーバを介してディスクをマウントし、ディスクアクセスを開始する。

4. 負荷分散コントローラは Candidate Migration を利用して、新規投入ノードにハッシュラインの配布を試みる。この時、新規投入ノードでは $C_i = 0$ であるため、統計情報 (L_{RECVi}) から $\gamma_i\alpha$ が算出できず、マイグレーション時の C_i 配布量が決定できない。そのため、ごく少数のハッシュラインを配布し $\gamma_i\alpha$ 値が得られるのを待つ。

5. $\gamma_i\alpha$ 値が得られたのち、改めて Candidate Migration を利用して負荷の調整を行う。

5.2 動的デクラスタリングによる I/O 帯域拡張

CPU パワーの動的投入を行うと、ノード数かある値に達した時点で系は CPU バウンドから I/O バウンドへと変化する。また、パス 3 以降の後半のパスは I/O バウンドである。このような I/O バウンドした系では I/O 帯域を拡張させることで、実行時間を短縮させることができる。

本論文では I/O バウンド時の性能改善のため、動的デクラスタリング機構 [14] を用る。これはデータが存在しているストレージデバイスからデータを分割して未使用のストレージ空間にコピーを行い、並列にアクセスを行うことで、ディスク I/O の帯域を拡張するもので、データマイニングの様な繰り返しストレージデバイスをシーケンシャルアクセスするアプリケーションに非常に有効である。図 6 に動的デクラスタリングの概念図を示す。

そこで、動的デクラスタリング機構をストレージ仮想化機構の機能として追加を行った。その制御方式について説明する。

1. パス 1 に於いては、SAN メタサーバから LVM に対して ACQUIRED メッセージを通知する際に、デクラスタを作成する旨 (DODECLUSTER) とコピーの作成先メタ情報を併せて送信する。

2. DODECLUSTER メッセージを受けた SEND プロセス (LVM) はコピー元デバイスからトランザクションデータベースを読み込み、処理を行うとともに未使用デバイスにデクラスタコピーを作成する。

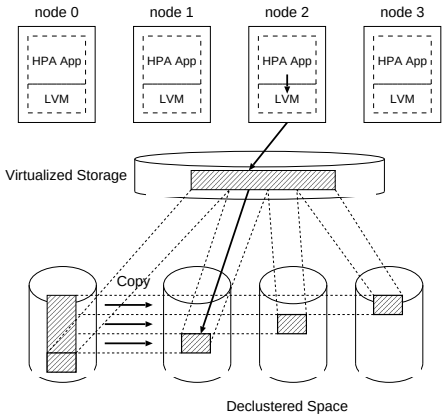


図 6: 動的デクラスタリングの概要

3. パス 2 開始以後、SEND プロセス (LVM) はノードのバウンディングファクタ (CPU バウンドかディスク I/O バウンドか) を判断し、SAN メタサーバに通知する。通知を受け SAN メタサーバは、系全体がディスク I/O バウンドと判断されると、自身の管理するメタ情報を更新してコピー元から Declustered Space へと切替えを行う。

4. この切替えにより、次回の REQUEST 以降、マイニングノードの Declustered Space へのアクセスが活性化する。

5.3 動的資源投入の評価実験

ノード数の増加による CPU パワーの動的投入および動的デクラスタリングによる I/O 帯域拡張を用いることで、系がどの資源の性能にバウンドした状態であるかを適切に判断して、資源の拡張を行うことが可能になる。つまり、CPU バウンド状態にある時には CPU パワーの拡張を、ディスク I/O バウンド時には I/O 帯域拡張を行うことが可能である。

ノード数の増加による CPU パワーの動的投入および動的デクラスタリングによる I/O 帯域拡張の評価を行うため、実験を行う。実験には表 5 に示すものを用いた。表中 Unused Storage をデクラスタコピーの作成先として利用する。また、最小支持度は 1.5%、ロック粒度は 1024KB とした。アプリケーション開始時には Node 0 のみが活性化しているとする。

まず、CPU パワーの動的投入のみを行った場合の実行トレースを図 7 に示す。パス 1 が約 70 秒程度行われた後、パス 2 が開始し、負荷分散コントローラは

表 5: ノード/ディスクデータの配置

ノード配置

Node 0	Node [1-3]	Node 4	Node [5-23]
PentiumIII 800MHz	PentiumII 450MHz	PentiumPro 200MHz	PentiumIII 800MHz

データ配置

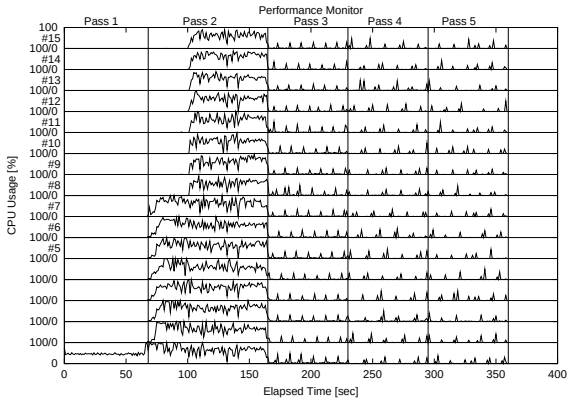
Disk 0	Disk [1-3]
20,000,000 (1.63GB)	Unused Storage

Number of transactions / Size of data volume

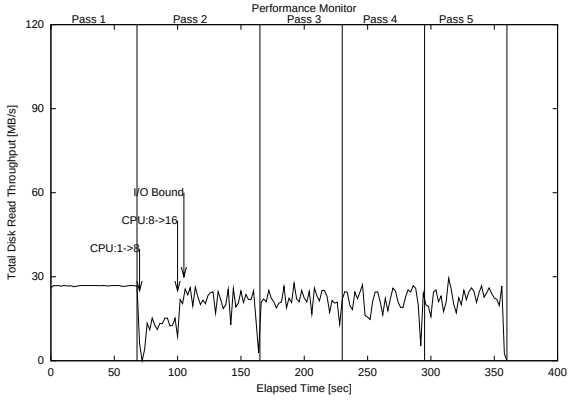
CPU バウンドを判断、直ちに Node [1-7] のノードを追加する。新規投入されたノードは直ちに SEND が処理を開始するとともに、Candidate Migration によって RECV にハッシュラインが配られ、負荷が均衡化している。その後再び、負荷分散コントローラは CPU バウンドを検出し、さらに Node [8-15] の追加を行う。この追加の後も、同じく Candidate Migration によって全ノードの負荷が均衡化するが、ここで系全体のディスクリードのスループットは約 25MB/s に達し、I/O バウンドへと変化する。このため、ノードの追加はこれ以上は行われない。

更に、ノード数の増加による CPU パワーの動的投入と動的デクラスタリングによる I/O 帯域拡張を併用した場合の実行トレースを図 8 に示す。この場合、パス 1 であらかじめ投機的なデクラスタ作成が行われる。パス 2 以降では CPU のみを拡張する図 7 のケースと同じく、CPU バウンドで CPU パワーの動的投入が行われるが、ノード数を 16 台まで拡張した時点で系のディスク I/O バウンドを検出し、直ちに SAN メタサーバにより Declustered Space へのアクセスの活性化が行われる。このため、I/O 帯域が拡大し再び CPU バウンド化している。そこで、再び負荷分散コントローラが CPU の動的投入を指示するため、最終的に 24 ノードまで拡大する。また、パス 3 以降のディスク I/O バウンド時のスループットが約 3.6 倍に改善している。

両手法による性能改善を実行時間により確認するため、表 5 のトランザクションデータベース長の 4 倍のデータを用意し、各パスの実行時間を測定したものを、図 9 に示す。CPU パワーの動的投入のみの場合、CPU バウンドパスであるパス 2 が大幅に改善しているが、さらに I/O 帯域の拡張も用いることでパス 2 は 24 ノードまで拡張した効果から更に改善し、1 ノード時に比べ 18 倍に達している。これは、逐次ノードを増加させる過程を考慮すると、十分な性能改善値であると言える。また、パス 3 以降の I/O バウンドパスの改善は



各ノードの CPU 利用率



システム全体のディスクリードスループット

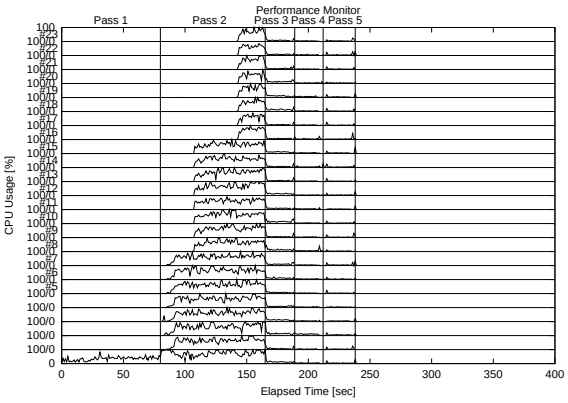
図 7: 実行トレース (24 ノードまでの CPU パワー投入, 最小支持度 1.5%)

3.3 倍程度であり、パス 1 も含んだ全パスの実行時間の性能改善は約 5.8 倍であった。

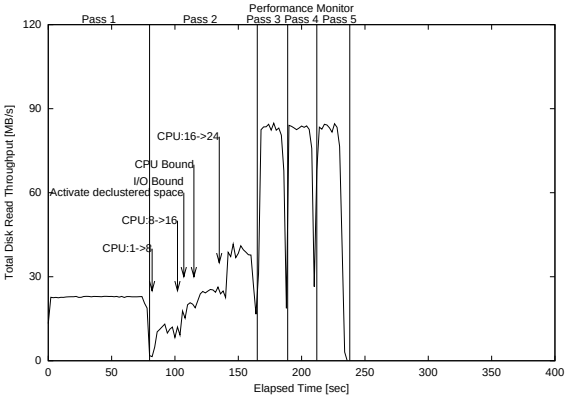
上記の実験結果から独立した 2 つの機能を組み合わせることで CPU パワーおよび I/O 帯域双方の性能限界を検知し、必要な資源を投入することができ、それが性能改善に極めて有効であることが分かった。

6 まとめ

SAN を適用した PC クラスタに於ける相関ルール抽出処理に関して、動的負荷分散機構の設計を行い、実装実験によりその有効性を確認した。データスキューに関してはストレージ仮想化機構によりほぼその弊害を除外することができた。また、苛酷な CPU スキューに関しても Shared Nothing に比べ実行時間が改善し、多ノード環境でも有効であることが分かった。加えて、緩やかな CPU スキューに関してはアプリケーションの知識を必要とする制御が不要で、ストレージ仮想化



各ノードの CPU 利用率



システム全体のディスクリードスループット

図 8: 実行トレース (24 ノード 4 ディスクまでの動的資源投入, 最小支持度 1.5%)

機構のみで解決できることも分かった。

その上で CPU パワーの動的投入と I/O 帯域の拡張を加えて実装し、非均質な PC の組み合わせ環境の元で実験を行った。ノード数の増加による CPU の動的投入は負荷分散コントローラ、動的デクラスタリングによる I/O 帯域の拡張はストレージ仮想化機構によってそれぞれ独立に制御されるが、双方とも有効に機能することが実験によって確認された。

今後はストレージ仮想化機構のみで解決できるスキューの範囲を明確にした上で、その効果を検証する予定である。ストレージ仮想化機構のみでスキューが解決できることはアプリケーション毎に開発する必要のある負荷分散機構を共通の下位レイヤで制御可能にするものであり、数多くのデータインテンシブアプリケーションに対し、アプリケーションの書き換え無しに適応可能と考えられる。また、本論文で提案した Candidate Migration はアプリケーションからの知識が必要となるが、その修正もそれほど大きくなく、アプリケーションへの改変を最小化するツールに関して今後研究を進めて行きたい。

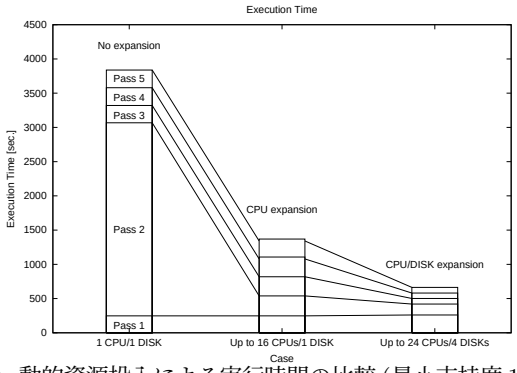


図 9: 動的資源投入による実行時間の比較 (最小支持度 1.5%)

参考文献

[1] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules. In *Proceedings of the Twentieth International Conference on Very Large Data Bases*, 1994.

[2] Charles T. Clark. *The Virtualization of Storage*. White Papers, TidalWire http://www.tidalwire.com/, 2001.

[3] R. M. J. Christopher B. Walton, Alferd G. Dale. A Taxonomy and Performance Model of Data Skew Effects in Parallel Joins. In *Proceedings of the Seventeenth International Conference on Very Large Data Bases*, 1991.

[4] Compaq. Compaq Innovation to Transform Enterprise Storage Deployment, Utilization and Management. Technical report.

[5] R. Hernandez, C. C. Kion, and geoff Cole. IP Storage Networking: IBM NAS & iSCSI Solutions. Technical report, 2001.

[6] SAN Symphony. DataCore Software, http://www.datacore.com/.

[7] T. Shintani and M. Kitsuregawa. Hash Based Pararellel Algorithm for Mining Assocication Rules. In *Proceedings of Parallel and Distributed Information Systems*, 1996.

[8] SV Router. Vicom Systemns, http://www.vicom.com/pdf/bro svrouters.pdf. Technical report.

[9] M. Tamura and M. Kitsuregawa. Dynamic Load Balancing for Parallel Association Rule Mining on Heterogeneous PC Cluster Systems. In *Proceedings of the Twenty-fifth International Conference on Very Large Data Bases*, 1999.

[10] VERITAS Software. http://www.veritas.com/. Technical report.

[11] コンピュータエージ社. データストレージレポート 2001. **月刊コンピュートピア**, 2001.

[12] ファイバチェンネル協議会. **ファイバチャンネル技術解説書**. 論創社, 2001.

[13] 安井, 田村, 小口, and 喜連川. 並列 dbms に於ける動的負荷分散機構の実装. Number 61 in 99, pages 393–398. 情報処理学会, 1999.

[14] 小口, 喜連川. SAN 統合 PC クラスタ上の並列データマイニングのための動的データ・デクラスタリング. In **情報処理学会データベースシステム研究報告**, 2001.