CPU 資源並びにディスク資源の動的投入を可能とする SAN 型 PC クラスタにおける実行時負荷調整機構 —データマイニングアプリケーションを用いたケーススタディ—

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性能価格比の面から PC クラスタが注目され、導入が進んでいる。しかし、従来の Shared Nothing 構成の PC クラスタでは、個々のノード毎にディスクの管理がなされており、データベースアプリケーション実行時に、状況に応じて実行ノード数、ディスク数を変更する等の制御が困難である。負荷分散性能におけるこのような限界は、PC クラスタの利便性や利用効率を低下させる原因になり得る。著者らはデータベースアプリケーションを対象に、SAN 型 PC クラスタ上で共有ディスクアクセス方式を行うストレージ仮想化機構を開発し、その上での負荷分散処理を利用して、アプリケーション実行時に動的に必要な資源を投入する機構を実現した。本論文では両機構の設計について解説を行い、データマイニングアプリケーションを用いた実験による評価を示し、ストレージ仮想化機構の負荷分散処理での有効性を示す。

Load Balancing System during Execution on SAN-connected PC Cluster for Dynamic Expansion of CPU and Disk Resource — A Case Study of Data Mining Application —

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 attractive as a next-generation database platform. However, it is difficult for the conventional shared-nothing system of PC cluster to change the number of active nodes and disks during application execution, since each node manages its own storage device. Such limit of load balancing performance causes inconvinience and unusefuleness of PC cluster system to large database applications. In this paper, we design and implement *Storage Virtualizer*, which enables shared-disk access on the SAN-connected PC cluster, and *Dynamic Resource Injection*, where the system can inject CPU power and extend I/O bandwidth by adding idle nodes and unused disks dynamically. As a result of experiment, we show that both Storage virtualizer and Dynamic Resource Injection are efficient for large data mining processing and confirm the advantage of the SAN-connected PC cluster for large database applications.

1 はじめに

PC クラスタはスケーラビリティと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。しかし、従来の PC クラスタはストレージドライブが各ノードに従属する Shared Nothing 方式が主であり、ノード数を増やして CPU パワーを増大させたり、ディスク数を追加して入出力帯域を拡張するといった動的な資源管理が困難である等、特に、巨大なディスク上のデータを扱うデータベースアプリケーションでは、負荷分散性能に限界が見られた。このような特徴は PC クラスタの利便性や利用効率を低下させる要因である。実運用環境での大規模データベースプラットフォームとしての利用拡大にあたり、PC クラスタにおける、CPU パワーおよび入出力双方の動的資源管理を行う機構は極めて重要な機能の一つと考えられる。

著者らは SAN(Storage Area Network) 技術を PC クラスタに導入し、大規模データベース処理に適したストレージ仮想化機構を用いた共有ディスクアクセス方式を実装し、さらに動的負荷投入方式について検討を行った。本論文ではストレージ仮想化機構による負荷分散処理方式の有効性を確認するため、並列データマイニング処理を取り上げ、実装上で評価を行い、それを利用した動的な資源投入の実験結果について述べる。

なお、並列データマイニング処理は大容量かつスキュー (偏り) のあるトランザクションデータを扱い、その性能が CPU パワーおよび入出力性能双方に依存する。このため、効率のよい処理には演算処理および入出力双方の負荷分散制御が必須であり、提案するシステムの検証に理想的なアプリケーションである。

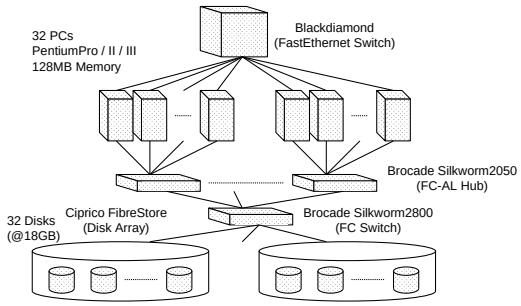


図 1: SAN 型 PC クラスタシステムの概要

表 1: SAN 型 PC クラスタシステムの諸元

共通性能

Main Memory	128Mbytes
OS	Solaris 8 for x86
FastEthernet NIC	Intel EtherPro/100+
Fibre Channel HBA	Emulex LP8000
Disk Array	Ciprico Fibre Store (Seagate Cheetah 18GB)

ノード別性能

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

2 SAN 型 PC クラスタにおけるデータベースプラットフォーム

2.1 SAN 型 PC クラスタ

SAN(Storage Area Network) は、複数のホストコンピュータと複数のストレージデバイスを専用のネットワークによって相互接続することで、複数のホストからストレージをハードウェアレベルで共有する技術である。従来 NFS 方式等のストレージ共有に用いられた LAN(Ethernet) では 1500B 毎に CPU が介在する必要があるが、現行 SAN で利用されている FC(Fibre Channel) では CPU の I/O 命令一つで最大 128MB のデータを扱う事が可能で、データベースアプリケーションのディスクアクセスに有利である。

本論文では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 型 PC クラスタを構築した。システムの概要を図 1 に、構成の諸元は表 1 に示す。

2.2 ストレージ仮想化機構

著者らは SAN 型 PC クラスタシステム上で、データベースアプリケーションを対象として LVM(Logical Volume Manager) とメタサーバ (Meta Srv) からなるストレージ仮想化機構を開発し、同システム上で実装した。ストレージ仮想化機構を用いたデータベースアプリケーションの動作方式を図 2 に示す。LVM は

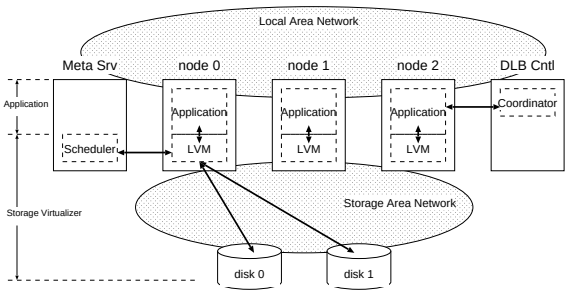


図 2: ストレージ仮想化機構を用いたアプリケーション動作方式

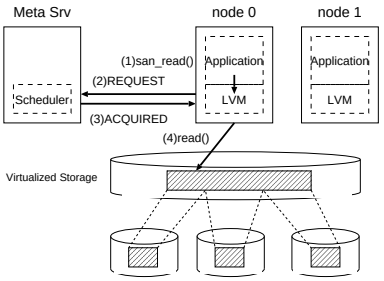


図 3: メタサーバを用いたディスクアクセス手続き

メタサーバと協調して、アプリケーションに対して仮想化されたストレージ空間へのアクセスを提供する。本論文のシステムでは LVM は通常のシステムコール (`read()` 等) に代わる API(Application Programming Interface) の形で実装されており、アプリケーションは `san_read()` 等の関数を呼び出すことで仮想化されたストレージにアクセスすることができる。一方、負荷分散コントローラ (DLB Cntl) はアプリケーション独自の知識を必要とする制御を行う。

SAN 環境で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。ストレージ仮想化機構では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共に、システム全体の入出力アクセスの管理を行う。

図 3 に具体的な動作として、ストレージ仮想化機構に於けるディスク読み出し処理の例を示し説明する。まず、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 (REQUEST メッセージ) 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、SAN デバイス上の領域を予めアプリケーションによって指定された単位 (ロック粒度) で LVM 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 LVM に通知 (ACQUIRED メッセージ) する。これを受け、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、上位アプリケーションに受け渡す。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

合は直接デバイスへの 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 上に分散されたオブジェクトは複数のホストコンピュータから仮想的に一つのオブジェクトとして共有されることになる。このためノード間のデータ量スキューは仮想化機構によりほぼ吸収される。また、後述するが、各ノードの CPU 性能の違い等による演算処理のスキューも仮想化機構によりある程度吸収することができる。一方、極端に CPU 処理負荷が偏った場合には、次節の負荷分散機構の助力を必要とする。

2.3 ストレージ仮想化機構における負荷分散処理

ノード間のディスクアクセスデータ量の差はストレージ仮想化機構で吸収され、各ノードの処理はほぼ同時に終了するため、アプリケーション側の負荷制御は非常に単純化することができる。

一般にデータベースアプリケーションでは、各ノードに於いて他ノードからデータを受信して処理を行うプロセスの優先度を高くする。このとき、特定のノードで受信プロセスが CPU パワーを使いきってしまう場合、受信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ず、定常的に受信キューが溢れる結果、システム全体の性能を低下させてしまう。このような事態を避けるため、アプリケーションの負荷分散機構はデータを受信するプロセスの CPU 利用率を監視し、それが 100%未満になるように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演算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、受信プロセスが CPU パワーを独占しているノード (ボトルネックノード) が存在する場合、負荷制御が必要と判断し、ノード間の演算負荷 (スレッドやメモリ上のデータ構造等) のマイグレーション計画を立てる。
3. 負荷分散コントローラが全ノードにマイグレーション計画を通知し、それを受け取ったノード間でマイグレーションを行う。

上記マイグレーションによりボトルネックノードの負荷が他のノードに分散され、ボトルネックが除去される。このような制御系は Shared Nothing 環境では全てのノードの処理がほぼ同時に完了すべく、厳密な系のモデル化とパラメータの観測を必要とする。しかし本論文の提案システムでは、ストレージ仮想化機構がノード間のディスクアクセスデータ量のスキューを自動的に吸収するので、受信プロセスのボトルネックを除くことが出来れば、厳密な制御系設計を行う必要はない。つまり、負荷分散コントローラとストレージ仮想化機構は、前者が厳密な制御は難しいがボトルネックを除去し、後者がその誤差を自動的に吸収するという相補的な関係で機能する。

2.4 動的資源投入

ストレージ仮想化機構上での負荷分散処理を利用して、アプリケーション実行時に CPU パワーを追加し、ならびにディスクの入出力帯域の拡張を行う動的資源投入機構を以下の通り設計した。

2.4.1 CPU パワーの動的投入

CPU パワーの動的投入としては、本論文のシステムでは遊休ノードの有効利用について検討する。

遊休ノードの CPU パワーを実行時に投入する機構は負荷分散コントローラの機能として実現する。本機能ではアプリケーション開始時には予め指定されたノード数のみでアプリケーションの処理を行う。その後、負荷分散コントローラが統計情報を収集し、系の CPU バウンドから CPU パワー投入の必要性を判断すると、必要な CPU パワーに応じて遊休ノード上にプロセスを起動し、アプリケーション処理を開始する。この時、アプリケーションは LVM を通じて仮想化ストレージ空間へのディスクアクセスを行う。さらに処理によっては、遊休ノードの追加によりボトルネックノードが新たに発生する場合がある。そのような場合は、負荷分散コントローラによるマイグレーションにより追加ノードに演算負荷 (スレッドやデータ構造等) が割り当てられる。しかし、この際、追加ノードに関しては統計情報を持っていないため正確なマイグレーション量を算出することができない。仮に多くの負荷を配布しすぎた場合、配布を受けたノードがクラスタ全体のボトルネックとなる恐れがあるため、初回のマイグレーション量は抑制し、統計情報を得て後、積極的なマイグレーションを行う。

2.4.2 入出力帯域の動的投入

SAN 環境での入出力帯域拡張は、ストレージ仮想化機構内に動的デクラスタリング機構 [4] として実現する。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利用のストレージ空間に投機的にコピーを行い、後で並列アクセスを行うことにより入出力帯域を拡張するもので、繰り返しディスク上の同じファイル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 4 に動的デクラスタリングの概念図を示し、その制御方式について以下に説明する。

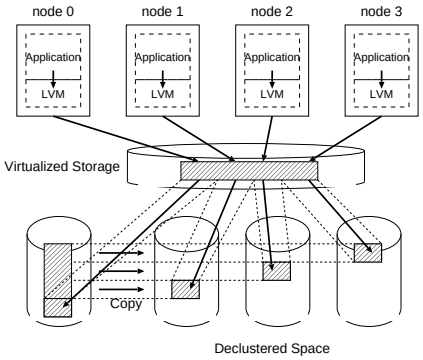


図 4: 動的デクラスタリングの概要

1. 動的デクラスタリングの対象となるオブジェクトについて、アプリケーションが明示的にフラグを立てることで、ストレージ仮想化機構はそのオブジェクトのデクラスタリングを有効にする。

2. 1 巡目のディスクリードの際は、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とコピーの作成先メタ情報を併せて送信する。

3. デクラスタ作成メッセージを受けた LVM はコピー元デバイスからデータを読み込みアプリケーションに渡すとともに、未利用デバイスのデクラスタ空間にデクラスタコピーを作成する。

4. 2 巡目以降の該当オブジェクトのディスクリードでは、LVM はノードのバウンディングファクタ (CPU バウンドか入出力バウンドか) を判断し、メタサーバに通知する。通知を受けメタサーバは、系全体が入出力バウンドと判断すると、自身の管理するメタ情報を更新してコピー元からデクラスタ空間へと切替えを行う。

5. この切替えにより、次回の REQUEST 以降、LVM のデクラスタ空間へのアクセスが活性化し、並列入出力アクセスにより帯域が拡大する。

遊休ノードの追加による CPU パワーの動的投入および動的デクラスタによる帯域拡張を併用することで、系がど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出来る。つまり、CPU バウンド状態にある時には遊休ノードを投入することで演算性能向上を、入出力バウンド時には動的デクラスタリングによる入出力帯域拡張を行うことが可能である。

3 並列データマイニング

本論文ではストレージ仮想化機構による動的負荷分散処理を利用して、並列相関ルール抽出処理をデータベースアプリケーションの例に取り上げ、実験を行う。本章で相関ルール抽出処理の説明を行う。

3.1 相関ルール

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 に対して支持度の最小値および確信度の最小値が与えられた際に、これらを満足する相関ルールを見出すことである。この処理は 1) 与えられた最小支持度を満たすアイテム (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、の 2 ステップで行われる。

相関ルール抽出処理の第 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、重負荷の処理である。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 並列相関ルール抽出処理

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が、本論文では Apriori をもとに提案された並列アルゴリズム HPA(Hash Partitioned Apriori)[2] を利用する。HPA 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 として、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 を作成する。C_k 内の各候補アイテムにハッシュ関数を適用する事で対応するノードを決定し、当該ノードのメモリ上の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、前項と同じハッシュ関数により送信先ノードを決定し、その 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の生起回数の数え上げを行い、支持度を求める。

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 プロセスの長さ k のアイテムの組合せ作成処理と RECV プロセス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 CPU パワーを必要とし、マイニング実行時間の大半を占める。対して、パス 3 以降の後半のパスに於いては、k が大きくなるに従い CPU 負荷を減らし、処理はディスクの入出力性能に依存する。この性質から、HPA では第一にパス 2 の実行時間の改善を行い、続いて後半のパスの入出力性能を向上させることが全体の高速化に結びつく。

3.3 並列相関ルール抽出に於ける負荷分散処理

本論文ではストレージ仮想化機構上で負荷分散コントローラを HPA アルゴリズムに対応させた。HPA アルゴリズムでは、RECV 処理が受信プロセスに相当するため、RECV プロセスの CPU 利用率を監視し、CPU パワーを独占しないよう制御を行う必要がある。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 RECV プロセスの CPU 利用率を L_{RECVi} とする。L_{RECVi} は RECV プロセスが受信するデータ流量 (V_{Ri}) に比例するため、以下のように表すことができる。

$$L_{RECVi} \quad = \quad \alpha\gamma_i V_{Ri} \ \sim \alpha\gamma_i C_i \sum_j V_{Rj}$$

ここに、α は RECV 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付き係数であり、ハッシュテーブルの再配置時の移動量決定に用いる[1]。添字 i はノード番号を表す。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 $L_{RECVi} < 1 - \epsilon$ が満たされるべく制御を行う必要がある[2]。

上式から L_{RECVi} はノードのハッシュ表によって負荷調整が可能であることから、HPA アルゴリズムでは L_{RECVi} を観測変数として $\alpha\gamma_i$ を算出し、C_i を操作変数とするハッシュラインマイグレーション (Candidate Migration) により負荷調整を行えるようにした。

4 ストレージ仮想化機構の性能評価

4.1 メタサーバの基本性能

LVM はメタサーバへ LAN 経由で問い合わせを行うため、LVM のディスクアクセス基本性能を測定した。ノードとディスクドライブを同じ個数用意し、ノード

[1] C_i の観測は [3] 手法を用いる。
[2] ϵ は本論文の実験では 5-10%程度取っている。

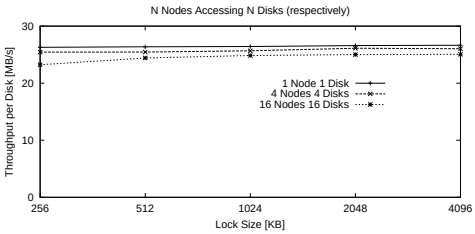


図 5: ノード-ディスク 1 対 1 アクセスのディスクスループット

表 2: 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	500,000 (42MB)	500,000 (42MB)	500,000 (42MB)	2,500,000 (210MB)
d2	200,000 (16.8MB)	200,000 (16.8MB)	200,000 (16.8MB)	3,400,000 (285.6MB)

Number of transactions / Size of data volume

i-ディスク i という固定的な一対一対応でアクセスさせる場合、ノード数とメタサーバのロック粒度を変化させてディスク 1 台あたりの平均スループットを計測した。測定結果を図 5 に示す。この結果、メタサーバへの問い合わせは大きな性能上の障壁とはならないことが分かる。なおこれを含めて以後、ディスクアクセスのバッファサイズは 64KB とする。

4.2 負荷分散性能

ストレージ仮想化機構上の上記の負荷分散処理を評価するために、表 2 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。c0 から c3 で CPU パワーの性能差が、d0 から d2 でデータスキューが大きくなり、系に取って c3d2 が最も苛酷な条件となる。実験では、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、メタサーバのロック粒度は 512KB とした。

各 CPU 性能差およびデータスキューの組合せに対し、SAN 環境のストレージ仮想化機構で負荷分散コントローラによる 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 3(a) に実測の実行時間を示す。この時、全てのケースでパス 2 は CPU バウンドであり、

本実験に於いては各ノードのメモリサイズ、ディスク I/O 性能が一定であるため、CPU 性能差による処理時間差と負荷制御効果を区別するために、CPU 性能/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基準に 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値を表 3(b) に示している。また、比較のために [3] 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 3 で “SAN” が SAN 環境でストレージ仮想化機構を用いた負荷分散、“SN” が Shared Nothing 環境の負荷分散を表す。[3] 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, Shared Nothing 間で Candidate Migration の定義は異なる。† は Candidate Migration、‡ は Transaction Migration のトリガが引かれなかったことを表す。

ストレージ仮想化機構単独の場合　表 3 からストレージ仮想化機構単独での負荷分散性能を検証する。データスキューに付いては、d0 と d1,d2 の比較でストレージ仮想化機構下では悪化が 4%以内に収まっていることから、仮想化機構でほぼ吸収することが出来ることが分かる。例えば、c0 ケースでは Shared Nothing 環境下では Candidate Migration と Transaction Migration 双方を用いても正規化実行時間が d0, d1, d2 の各ケースで 100, 120, 136 と悪化しているのに対し、ストレージ仮想化機構下では 98, 102, 102 と比較的安定している。

CPU 性能のスキューに関しては、c1 のような緩やかな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 上で Candidate Migration を行わないケースの実行トレースを図 6 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。

一方、一台の低性能ノードが全体のボトルネックとなるさらに苛酷な CPU のスキューである c2 や c3 ケースでは、ストレージ仮想化機構のみでは不十分である。これは、d0 時に c0 から c3 で正規化実行時間が 98 から

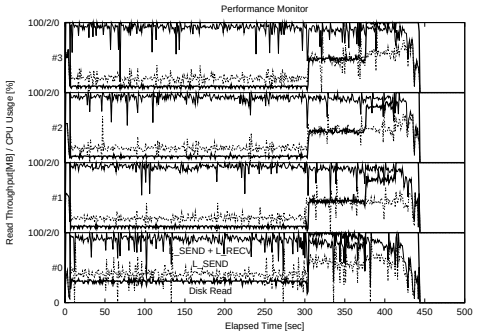


図 6: 実行トレース (c1d0 ケース, SAN 適用)

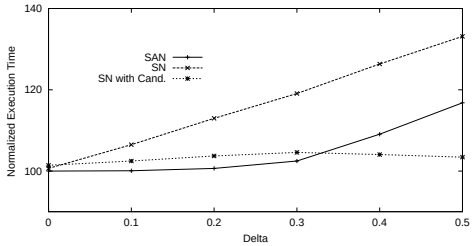


図 7: ハッシュライン配布の偏りに対するストレージ仮想化機構のスキュー吸収 (正規化実行時間)

175 へと 78%も悪化していることから分かる。しかし、加えてデータのスキューがある複合ケースの場合、たとえば、c3 ケースで比較すると、Shared Nothing 環境下では Candidate Migration と Transaction Migration 双方を用いても d1 時に 192、d2 時に 270 であるのに対し、ストレージ仮想化機構下では d1 時に 176、d2 時に 176 と有利な結果が出ている。これは Shared Nothing 構成の PC クラスタではデータマイニングの実行時間は非常にデータスキューに影響を受けやすいことを示している。

更にストレージ仮想化機構に於ける CPU 処理負荷分散の効果を示したグラフが図 7 である。ここでは、c0d0 のケースを用い、同じ CPU 性能のノード上で本来 Node 1 に配布される予定のハッシュラインのうち、割合 $\delta(0 \leq \delta \leq 1)$ のハッシュラインを Node 0 に配布したものである。このため、Node 1 が比較的軽いノードに、Node 0 が比較的重いノードになり、そのスキューの度合は δ が大きくなればなるほど苛酷になる。この時 δ を変化させ、実行時間を測定したものを示している。この場合、ストレージ仮想化機構のみによって約 30%程度までの CPU 処理のスキューが自動的に吸収が可能で、逆にそれを越えるスキューの場合は Candidate Migration によるボトルネック除去が必要であることが分かる。

ストレージ仮想化機構上で負荷分散処理を行う場合　ストレージ仮想化機構のみでは解決出来なかった苛酷な CPU スキューケースでは、$L_{RECVi} \geq 1 - \epsilon$ となるボトルネックの RECV プロセスが発生している。そ

表 3: パス 2 の実行時間

(a) 実測値

Case	Storage	Contorol	d0	d1	d2
c0	SN	-	222	295	377
		Cand.	222†	266	302
		Cand.+Trans.	222†‡	267‡	303‡
	SAN	-	218	226	226
		Cand.	218†	226†	226†
c1	SN	-	330	472	572
		Cand.	311	354	416
		Cand.+Trans.	311‡	344	415
	SAN	-	307	314	314
		Cand.	307†	313†	314†
c2	SN	-	322	426	556
		Cand.	259	314	394
		Cand.+Trans.	259	299	386
	SAN	-	269	279	279
		Cand.	242	249	249
c3	SN	-	601	825	1079
		Cand.	331	563	738
		Cand.+Trans.	309	525	708
	SAN	-	478	481	481
		Cand.	268	270	270

(b) 正規化値

Case	Storage	Contorol	d0	d1	d2
c0	SN	-	100	133	170
		Cand.	100†	120	136
		Cand.+Trans.	100†‡	120‡	136‡
	SAN	-	98	102	102
		Cand.	98†	102†	102†
c1	SN	-	100	143	173
		Cand.	94	107	126
		Cand.+Trans.	94‡	104	126
	SAN	-	93	95	95
		Cand.	93†	94†	95†
c2	SN	-	129	171	223
		Cand.	104	126	158
		Cand.+Trans.	104	120	155
	SAN	-	108	112	112
		Cand.	97	100	100
c3	SN	-	220	302	395
		Cand.	121	206	270
		Cand.+Trans.	113	192	259
	SAN	-	175	176	176
		Cand.	98	99	99

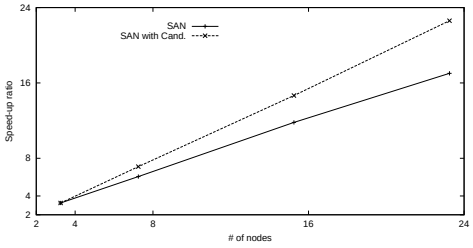


図 8: パス 2 のスピードアップ曲線

表 4: ノード/ディスクデータの配置

ノード配置

Node [0-1]	Node 2	Node [3-5]	Node [6-23]
PentiumIII 800MHz	PentiumPro 200MHz	PentiumII 450MHz	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

こで、さらにストレージ仮想化機構上で 3.3. で述べた Candidate Migration によりボトルネックを除去する。この結果、表 3 の (SAN, Cand.) 行に示すとおり、すべてのあらゆるスキューケース下で実行時間の悪化を 2%以内に納めることが可能となった。

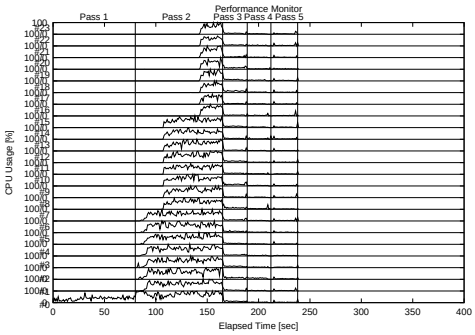
4.3 多ノード環境

多ノード環境での性能を調査するために、先の d0 のディスク配置、c3 の CPU 配置の下でさらに 800MHz ノードを数台追加し、パス 2 の実行時間を計測した。スピードアップ曲線を図 8 に示す。ストレージ仮想化機構単独の場合、スケールアップが劣る結果になっているが、これは 800MHz ／ 200MHz という大きな CPU 性能差により、200MHz のノードに於いて TCP/IP 通信処理の負荷が無視できなくなっていると考えられる。一方、ストレージ仮想化機構上で負荷分散処理を行う場合は、十分なスケールアップを呈している。

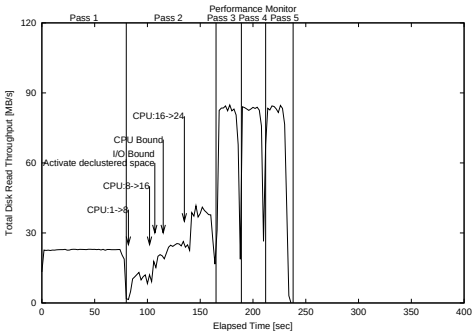
5 動的資源投入の評価実験

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帯域拡張を実装し、実験を行った。この際には表 4 に示すシステム構成を用いた。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 1.6%、ロック粒度は 1024KB とした。アプリケーションは開始当初、1 ノード (Node 0) のみで処理を行うものとする。

実験結果の実行トレースを図 9 に示す。図 9(a) では各ノードの CPU 利用率を、図 9(b) ではシステム全体のディスクリードのスループットを示している。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。続いてパス 2 が開始し、負荷分散コントローラは 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 プロセスが処理を開始するともに、Candidate Migration によって RECV プロセス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、入出力バウンドへと変化するするため、メタサーバは入出力バウンドを検出し直ちにデクラスタ空間へのアクセスの活性化が行われる。このため、入出力帯域が拡大し再び 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 以降の入出力バウンド時



(a) 各ノードの CPU 利用率



(b) システム全体のディスクリードスループット

図 9: 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 1.6%)

のスループットが約 3.6 倍に改善している。

動的資源投入による性能改善を実行時間により確認するため、表 4 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、CPU パワーの動的投入のみを行う場合、および CPU パワー・入出力帯域双方の投入を行う場合それぞれの各パスの実行時間を測定したものを、図 10 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、16 ノードで入出力バウンドを検出してそれ以上資源投入は行われない。一方、入出力帯域の拡張も用いるケースでは、パス 1 がデクラスタコピー作成のため 16%のオーバヘッドがあるものの、パス 2 は 24 ノードまで拡張することができ、1 ノード時に比べ 18 倍の性能に達している。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 以降の入出力バウンドパスの改善は 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。

6 まとめ

SAN 型 PC クラスタ上でストレージ仮想化機構を開発し、データマイニングアプリケーションを例に負荷分散性能に関する実験を行った。この結果、ストレージ仮想化機構が緩やかなスキューに対する自動調節機能を持っていることを確認した。系が極端に偏った場合

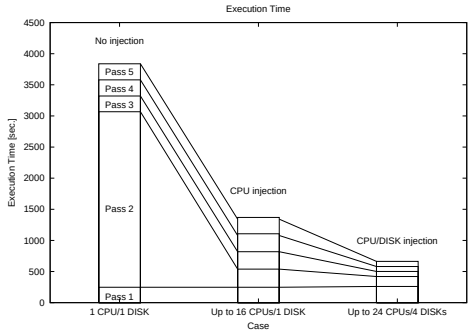


図 10: 動的資源投入による実行時間の比較 (最小支持度 1.6%)

には、アプリケーション知識を必要とする負荷分散処理が必要であるが、ストレージ仮想化機能と相補的に有効に機能することが分かった。このため、アプリケーションに会わせた負荷分散処理を有す Shared Nothing 構成の PC クラスタと比較し、十分に高い負荷分散性能を持つことを確認した。

また、ストレージ仮想化機構上に CPU パワーおよび入出力帯域の動的な投入を実現し、実験を行った。アプリケーション実行中にバウンドしている資源を検知することで、実行時に必要な資源を投入し、アプリケーション処理の性能を向上させることが可能になった。

本論文ではデータマイニングアプリケーションを用いて実験を行ったが、提案手法は他のデータベースアプリケーションでも有効であると考えている。今後、新たに大規模関係データベース演算等に適用し、実験により確認を行う予定である。また、動的資源投入に関しては本論文の実験段階ではボトルネック資源の投入による性能改善を中心に検討したが、不要な資源の削除など、より適切な資源調整手法についても今後実験を行い、システムが動的にアプリケーションに最適な資源を割り当てる動的資源管理機構の開発を行いたい。

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

[3] 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.

[4] 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.