論　文

SAN結合PCクラスタにおけるストレージ仮想化機構を用いた動的負荷分散ならびに動的資源調整の提案とその評価

合田　和生[†]　　田村　孝之[†]　　小口　正人[††]　　喜連川　優[†]

Proposal and Evaluation of Dynamic Load Balancing and Dynamic Resource Allocation Using Storage Virtualizer on SAN-connected PC Cluster

Kazuo GODA[†], Takayuki TAMURA[†], Masato OGUCHI[††], and Masaru KITSUREGAWA[†]

あらまし　PC クラスタは価格性能比の面から注目されており，特に大規模データを扱うアプリケーションにおいては今後中心的な役割りを果たすと期待されている．しかし，これまで PC クラスタで主に採用されてきた非共有ディスク (Shared Nothing) 方式のストレージアーキテクチャでは，特に大規模データを扱う場合，負荷の偏りを均衡化する動的負荷分散には限界があり，また，ノード数やデータ配置を変更する動的資源調整が困難であった．一方，近年ではファイバチャネルに代表されるストレージエリアネットワーク (SAN) が普及し，当該技術により，種々のストレージ装置を論理的に集約，管理するストレージ仮想化機構が提案され，利用されるようになりつつある．本論文では，ノード間の演算処理の負荷分散やノード数の増減が可能であるアプリケーションを対象に，負荷の偏りに対しより高い均衡化能力を有する動的負荷分散と，CPU 演算能力としてのノード数と IO 帯域としてのディスク数を実行時に変更可能な動的資源調整の実現を目的とし，共有読み込み及び動的デクラスタリングなる新たな機能を有するストレージ仮想化機構を提案する．また，SAN 結合 PC クラスタ上で，並列データマイニングアプリケーションを用いた実験を行い，提案手法の有効性を明らかにする．

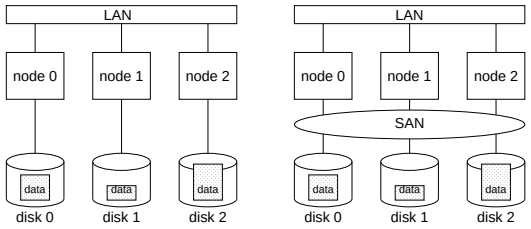
キーワード　PC クラスタ, ストレージエリアネットワーク (SAN), ストレージ仮想化機構，動的負荷分散, 動的資源調整

1. まえがき

PC クラスタは価格性能比の面から広く注目されている．特に，意思決定支援やデータマイニングなどの大規模データを扱うアプリケーションにおいては，今後中心的な役割りを果たすと期待されている．PC クラスタに代表される並列計算機のストレージアーキテクチャは，非共有ディスク (Shared Nothing) 方式と共有ディスク (Shared Disk) 方式に分類される．非共有ディスク方式では各々のノード (PC) が直接接続されたディスクを有する (図 1(a) 参照)．アプリケーションが処理するデータは予めノード間に分散してディスクに格納され，各ノードは直接接続されたディスクに対してのみアクセスを行う．一方，共有ディスク方式はディスクをネットワークにより接続しノード間で共有する (図 1(b) 参照)．アプリケーションが処理するデータはネットワーク上のディスクに格納され，ノードは全てのディスクに直接アクセス可能である．

従来，PC とストレージを接続する汎用のネットワーク技術が存在しなかったことから，PC クラスタでは主に非共有ディスク方式が広く採用されると同時に，当該方式の研究が行われて来た．しかし，非共有ディスク方式の PC クラスタでは，ディスク上のデータが個々のノードによって管理されているため，ノード間でディスク上のデータを移送することによって，ノード間の負荷調整を行おうとすると，データ転送は LAN を経由することになり，一般にオーバヘッドが大きく，とりわけ，大規模データを扱うアプリケーションにおいては動的負荷分散には限界がある．同時に，ノード

[†] 東京大学生産技術研究所, 東京都
Institute of Industrial Science, The University of Tokyo, 4-6-1, Komaba, Meguro-ku, Tokyo, 153-8505
[††] お茶の水女子大学理学部, 東京都
Faculty of Science, Ochanomizu University, 2-1-1, 0tsuka, Bunkyo-ku, Tokyo, 112-8610



(a) shared-nothing (b) shared-disk

図 1: 非共有ディスク方式と共有ディスク方式
Fig. 1 Shared-nothing vs. shared-disk

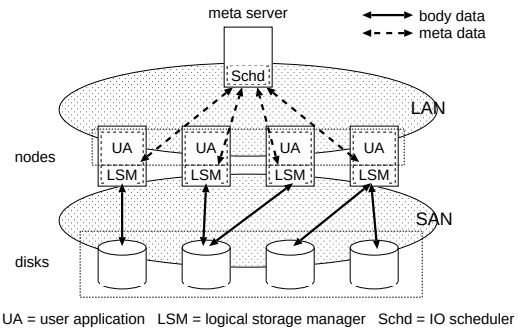


図 2: ストレージ仮想化機構の概要
Fig. 2 Overview of storage virtualizer

数やデータ配置を変更する動的資源調整も容易ではない．

一方，近年ではファイバチャネルに代表されるネットワーク技術の向上によりストレージエリアネットワーク (SAN) が普及し，当該技術により種々のストレージ装置を論理的に集約，管理するストレージ仮想化機構が提案され，利用されるようになりつつある [1] [2] [3] [4] [5]．しかし，これらのストレージ仮想化機構は，IO アクセスの論理的一貫性の保証や，従来分散していたストレージ管理を一元化することによる管理コストの低減を主な目的としており，著者の知る限り，アプリケーションの動的負荷分散や動的資源調整に係る問題の解決手法を提案したものはない．現時点では，ファイバチャネルは高価であることから，メインフレームや UNIX サーバに利用されることが多いが，低価格化が急速に進み，既にブレードサーバのオプションともなっており，今後 SAN は PC クラスタに広く利用される可能性がある．

本論文では，ノード間の演算処理の負荷分散やノード数の増減が可能であるアプリケーションを対象に，SAN を用いることにより共有ディスク方式のストレージアーキテクチャを採用する PC クラスタにおいて，負荷の偏りに対しより高い均衡化能力を有する動的負荷分散と，CPU 演算能力としてのノード数と IO 帯域としてのディスク数を実行時に変更可能な動的資源調整の実現を目的とし，共有読み込み及び動的デクラスタリングなる新たな機能を有するストレージ仮想化機構を提案する．また，ファイバチャネルによって接続された SAN 結合 PC クラスタを用いて実装を行い，並列データマイニングアプリケーションを用いた実験により提案手法の有効性を示す．著者らは [6] において並列データマイニングに限定した負荷分散ならびに資源投入の方式を提案したが，本論文は資源の投入と除去双方を可能とするとともに，大規模データを扱うアプリケーションに広く利用可能な方式を提案する．

本論文の構成は以下のとおりである．2. では提案するストレージ仮想化機構と，その機能である共有読み込み及び動的デクラスタリングを説明する．3. ではストレージ仮想化機構を用いた動的負荷分散及び動的資源調整の手法を述べる．4. では実験に用いる並列データマイニングのアルゴリズムを紹介する．5. では本論文の実験環境として SAN 結合 PC クラスタを説明し，実装したストレージ仮想化機構の基本性能，及び並列データマイニングアプリケーションを用いた動的負荷分散，動的資源調整の評価実験を示す．6. ではまとめと今後の課題について述べる．

2. ストレージ仮想化機構

2.1 ストレージ仮想化機構の概要

本論文で提案するストレージ仮想化機構は専用のノードで動作するメタサーバと，複数のノード内の LSM(Logical Storage Manager) により構成される．ストレージ仮想化機構は SAN 内のディスクから仮想化ストレージ空間を構成する．LSM は論理ボリュームマネージャ及びファイルシステムの機能を有しており，アプリケーションは LSM を通じて仮想化ストレージ空間中のファイルへ IO アクセスを行うことが可能である．この際，メタサーバは仮想化ストレージ空間に関するメタ情報の一元管理を行う．図 2 にストレージ仮想化機構の概要を示す．メタサーバと LSM のメタ情報交換には IP 通信による LAN を，LSM とディスクとのデータ転送には SAN を用いる．非共有ディスク方式のストレージアーキテクチャでは，他ノードのディスクへアクセスする場合，LAN 経由でノード間のデータ交換を行う必要があり，CPU 及び IO 双方への負荷が無視できない．一方，共有ディスク方式のスト

レージ仮想化機構では，ノードは全てのディスクへ直接アクセス可能であるため，IO アクセスによる他ノードの性能への影響は少ないことが期待される．

仮想化ストレージ空間上のファイルはメタサーバによりディスクへの割り当てが管理される．このため，ファイルが複数のディスクに分散された場合もアプリケーションは単一のファイルとして扱うことが可能である．本論文のストレージ仮想化機構は，さらに以下の機能を有する．

- 共有読み込み
- 動的デクラスタリング

これらは通常のストレージ仮想化機構には無い機能であるが，共有ディスク方式のストレージアーキテクチャにおける大規模データを扱うアプリケーションの動的負荷分散及び動的資源調整に有効であると考えている．本章では以降，2 つの機能を解説する．

なお，本論文が共有読み込みと動的デクラスタリングの提案及び有効性の検証に焦点を絞っていること，ならびに既にストレージ仮想化機構の一貫性管理に関しては研究がなされ [1] [2] [3] [4]，その成果が利用可能であることから，書き込み操作における厳密な一貫性管理に関しては本論文の対象外とする．

2.2 共有読み込み

大規模データを扱うアプリケーションでは，巨大なファイル全体を読み込み，処理する場合がある．例えば，一般的なデータマイニングのアルゴリズムではトランザクションデータベースを繰り返し，読み込む必要がある [7]．また，データウェアハウスでは，データ整形やインデクス構築，意志決定支援問い合わせのために同様の処理が行われる．非共有ディスク方式のストレージアーキテクチャでは，予めファイルを複数のノードに分割して格納し，その後，アプリケーションを実行する．この場合，処理対象となるデータは処理要求に依存するため，各ノードが読み込むデータ量は必ずしも同一となるとは限らず，またノードの演算能力が異なることがあるなど，処理負荷が均一化されず，並列度が低下することが多く見られる．

一方，本論文のストレージ仮想化機構は，共有ディスク方式のストレージアーキテクチャの下でこの問題の解決を目指し，共有読み込みなる機能を有する．共有読み込みでは，メタサーバがシークポインタを管理することにより，ファイルを共有するノードが分担して，ファイル全体を読み込むことができる．

図 3 に共有読み込みの概要を示す．アプリケーションは共有読み込みを開始する前に，共有されるファイルと同ファイルを共有するノード集合 (共有グループ) をメタサーバに通知する必要がある．その後，アプリケーションはファイルをオープンする際に SHARED_READ フラグを用いることにより，共有グループ内で当該ファイルをオープン済みのノード間で共有読み込みを有効にすることができる．まず，アプリケーションから初めて読み込み要求 shared_read() が呼び出されると，LSM 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，ファイル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 LSM 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等のメタ情報を LSM に通知 (ACQUIRED メッセージ) する．これを受け，ノードの LSM はディスクへの読み込みを行い，必要なデータを取得し，上位のアプリケーションに受け渡す．2 回目以降の shared_read() 呼び出しに対しては，ロックしている未読み込み領域が残っている場合はそのままディスクへの読み込みを行い，ロック領域を全て読み込み済みの場合は改めてメタサーバへの問い合わせを行い，新たな領域のロックを取得する．以上の手続きを繰り返すことにより，ファイル中のデータをノードの要求に応じて分配することが可能になる．また，共有読み込みに新たに参加するノードは，他のノードが既に共有読み込みを行っている最中に，同様に SHARED_READ フラグを用いてファイルをオープンすることにより，途中から共有読み込みを開始することが可能である．

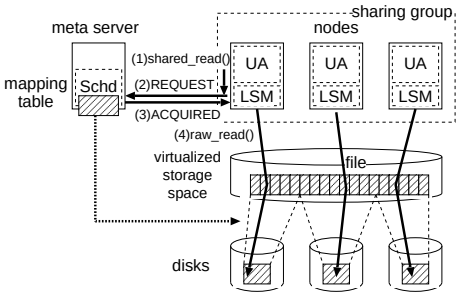


図 3: ストレージ仮想化機構における共有読み込み
Fig. 3 Shared read on storage virtualizer

ストレージ仮想化機構ではファイルのディスクへの割り当てはメタサーバによって管理され，各ノードのアプリケーションはこれを考慮する必要がない．す

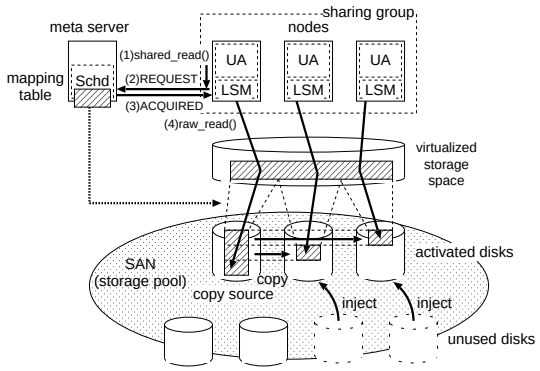


図 4: ストレージ仮想化機構における動的デクラスタリング
Fig. 4 Dynamic declustering on storage virtualizer

なわち，ディスク間でファイルのデータ量が偏っていたとしても，その偏りはストレージ仮想化機構により自動的に吸収される．また，共有読み込みを用いることにより，実行時にアプリケーションからの読み込み要求に応じてデータがノード群に分配される．即ち，5.3 に示す並列データマイニングを用いた実験が示す通り，アプリケーションを実行している全ノードは当該機構によりほぼ同時刻にファイルの読み込みを終了するため，ノード間で発生し得る演算負荷の偏りも，ある程度共有読み込み機能により吸収可能なことが期待される．

2.3 動的デクラスタリング

大規模データを扱うアプリケーションでは，その性能が IO 帯域に依存する場合がある．このため，適切な IO 帯域をアプリケーションに割り当てることが必要である．

本論文のストレージ仮想化機構は，共有読み込みにおいてファイルアクセスの IO 帯域を動的調整することを目的に，動的デクラスタリング [8] なる機能を有する．動的デクラスタリングはデータが存在しているディスクからデータを分割して複数の未利用ディスクへ実行時にコピーを行い，後の IO アクセスを並列化し，IO 帯域を拡張することを可能とする．コピー前のアクセス方式と，動的デクラスタリングによって並列化されたアクセス方式を切り替えることにより，アプリケーションの実行状況に応じて IO 帯域を 2 段階に調節することが可能となる．図 4 に動的デクラスタリングの概要を示す．

アプリケーションは動的デクラスタリングを開始する前に，対象ファイルと分割コピーを格納する未利用ディスクの数(注1)をメタサーバに通知する必要がある．この通知を受け，メタサーバは指定された数の未利用ディスクをストレージプール内から選択し，当該アプリケーション専用の分割コピー空間として割り当てる．その後，アプリケーションはファイルをオープンする際に，`DYNAMIC_DECLUSTER` フラグを用いることにより，当該ファイルに関する動的デクラスタリングを有効にすることができる．1 巡目の読み込みの際は，メタサーバから LSM に対して ACQUIRED メッセージを通知する際に，分割コピーを作成する旨とコピーの作成先メタ情報が併せて送信される．分割コピー作成メッセージを受けた LSM は，コピー元ディスク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，割り当てられた未利用ディスク群に分割してコピーを作成する．コピー作成は LSM 内で非同期的に行われるため，アプリケーションの IO アクセス性能に及ぼす影響は小さいことが期待される．アプリケーションがファイル全体を読み込むことにより，ファイル全体の分割コピーが生成される．このため，2 巡目以降のファイル読み込みでは分割コピーが完成した後の任意の時点で分割コピーへのアクセスを活性化することが可能である．なお，動的デクラスタリングによって分割コピーが作成されたディスクは，メタサーバによってアプリケーションの実行が終了する際に解放され，未利用ディスクに復帰する．

分割コピーへのアクセスの活性化はストレージ仮想化機構内部で自動的に行われる．メタサーバは各ノードから統計情報として IO 転送スループットを収集(注2)し，監視する．ここでメタサーバは下記の式により，IO 利用率 $L_{IO}(0 \leqq L_{IO} \leqq 1)$ を算出する．

$$L_{IO} = \frac{\sum_{i \in AD} T_i}{\sum_{i \in AD} Tmax_i} \quad (1)$$

AD はファイルが格納されているディスク集合を，T はディスクの IO 転送スループットを，$Tmax$ はディスクの IO 帯域を表す．L_{IO} が投入閾値 θ_{inject} を超えている場合，メタサーバは IO 帯域の拡張が必要であると判断し，自身の管理するメタ情報を更新して，未利用ディスクに作成された分割コピーへのアクセスを活性化させる．この活性化により，次回の REQUEST

(注1)：未利用ディスク数は，アプリケーションのユーザが期待する最大実効 IO 帯域に基づき決定する．
(注2)：統計情報は LSM の送信する REQUEST メッセージに埋め込まれてメタサーバに通知されるため，オーバヘッドは発生しない．

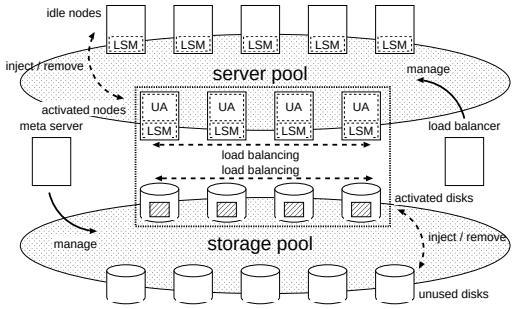


図 5: ストレージ仮想化機構を用いた PC クラスタシステムの概要

Fig. 5 Overview of PC cluster system using storage virtualizer

以降，IO アクセスの並列度が向上し，IO 帯域が拡大する．

一方，L_{IO} が除去閾値 θ_{remove} を下回った場合，IO 帯域に余剰があると判断し，メタサーバは自身の管理するメタ情報を更新して，追加された分割コピー空間へのアクセスを停止する．このため，IO アクセスの並列度が低下し，IO 帯域が減少する．

ストレージ仮想化機構は動的デクラスタリングを用いることにより，実行時にファイルアクセスの IO 帯域を 2 段階に調整することを可能とする．多くの PC クラスタなどの大規模システム設備においては，幾らかの空きディスクが存在することが想定される．動的デクラスタリングはアプリケーションを実行する時点においてこれらの利用可能な資源を一時的に用いることにより，性能の向上を図り，同時に，恒常的にコピーを置く場合に生じる複製管理における煩雑な問題も回避することができる．

3. ストレージ仮想化機構を用いた動的負荷分散と動的資源調整

3.1 概　　要

前章において述べたストレージ仮想化機構の適用により，大規模データを扱うアプリケーションの動的負荷分散及び動的資源調整を可能とする PC クラスタシステムの設計を行った．全体の概要を図 5 に示す．

システムはストレージ資源の集合であるストレージプール，及びそれを管理するメタサーバ，ノード資源の集合であるサーバプール，それを管理する負荷分散器から構成される．ストレージプールではストレージ仮想化機構によりストレージ資源が管理され，ノードのアプリケーションプロセスはストレージ仮想化機構を通じて，IO アクセスを行うことができる．サーバプール内では負荷分散器がノードへのアプリケーションプロセスの割り当てを行い，必要に応じてノード間の負荷分散を行う．この際，負荷分散器はアプリケーション固有の知識を用いることがある．

3.2 動的負荷分散

ノード間の負荷分散はアプリケーションの負荷分散器により行われる．従来，大規模データを扱うアプリケーションの負荷分散に関しては，非共有ディスク方式のストレージアーキテクチャにおいてアプリケーション毎に提案されてきた．これらの負荷分散手法は，本論文の負荷分散器にそのまま応用することができるが，ストレージ仮想化機構において共有読み込みを用いることにより，より単純な制御系でより高い負荷偏りの均衡化能力が期待されることを以下に示す．

大規模データを扱うアプリケーションでは，負荷分散はノード間の大きな演算負荷の偏りにより特定のノードがシステム全体のボトルネックとなることを防ぐと共に，ディスク上の未処理データ量を均衡化し，全ノードの処理の終了時刻を等しくする必要がある．非共有ディスク方式の下では，アプリケーションの負荷モデルに基づき，負荷マイグレーションによってノード間で演算負荷とデータ量を厳密に調整する必要がある．しかし，データ量の調整はノード間で大量のデータ交換を伴い，CPU 及び IO 双方にとって大きな負荷となり得る．また，汎用の構成を有する PC クラスタにおいて高精度の負荷観測や IO 操作は困難であり，誤差は無視できず，厳密な制御を期待することは容易ではない．例えば, [9] は並列データマイニングアプリケーションを例に，非共有ディスク方式の PC クラスタにおいて各ノードがディスク上のデータの読み込みを完了する時刻を予測し，その時刻を均衡化する制御を提案している．同手法に関しては，本論文では 5.3 に示す比較実験が示す通り，ノード間の負荷の偏りが軽微である幾つかのケースで有効に機能するが，演算負荷の偏りが極めて大きい場合，及びノード間でディスク上のデータ量に大きな差が存在する場合，ノード間でデータ量を適切に調節することができず，実行時間の悪化を十分に防ぐことができない場合があることが分かっている．

本論文のストレージ仮想化機構では共有読み込みを導入することにより，ファイル内のデータは各ノードに自動的に配分され，全てのノードがほぼ同時刻に処理を終了することが期待されると同時に，ノード間

の緩やかな演算負荷の偏りも吸収可能である．アプリケーションの負荷分散器は各ノードが扱うべきディスク上のデータ量を考慮する必要がなく，ノード間で演算負荷を調整し，特定のノードがシステム全体のボトルネックとなることを防ぐだけで十分である．このため，非共有ディスク方式における負荷分散と比較して，制御系を大幅に単純化することが可能である．アプリケーションの知識を用いる負荷分散器が大きな演算負荷の偏りを除去し，一方，ストレージ仮想化機構が共有読み込みによるデータの自動分配によりその誤差を吸収し，両者が相補的に機能することにより，負荷の偏りに対してより高い均衡化能力の達成が期待される．

3.3 動的資源調整

動的資源調整は，実行時にアプリケーションに割り当てる資源を調整することにより，利用可能な資源を活用し性能を最大化することを目指す．一般に PC クラスタ上で実行されるアプリケーションの性能は種々の資源によって決定されるが，本論文ではその性能を決定する要素として，CPU 演算能力及び IO 帯域を仮定する．

非共有ディスク方式の PC クラスタでは，データが各ノードの個別のディスクに格納されるため，CPU とディスクが密に結合しており，演算能力としての CPU 数と IO 帯域としてのディスク数を独立に変更することは実質的に困難である．すなわち，大規模データを扱うアプリケーションにおける柔軟な動的資源調整は未着手であったと言える．一方，共有ディスク方式のストレージ仮想化機構の下では，データは各ノードのアプリケーションから独立した機構によって管理される．実行状況に応じてアプリケーションのノードへの割り当てを変更することにより，CPU 演算能力の調整を行い，同時にデータのディスクへの割り当てを変更することにより，IO 帯域の調整を行うことが可能である．これらの資源調整は以下に説明する 2 つの方式によりそれぞれ行われる．

CPU 演算能力の動的調整は，サーバプール内でアプリケーションの演算処理を実行する有効ノード数を変更することにより行われる．アプリケーションがより多くの CPU 演算能力を必要としている場合には，サーバプール内でアプリケーションを実行していないノードの一部へ演算負荷を移送することにより，当該ノードは有効ノード群に投入される．この際，新たに投入されたノードは直ちに共有読み込みによる IO アクセスを開始することが可能である．逆に，CPU 演算能力が余っている場合には，一部の有効ノードの演算負荷を全て他の有効ノードに移送することにより，当該ノードは有効ノード群から除去される．これらの制御はノード間の負荷マイグレーションを通じて負荷分散器により管理される．投入，除去の判断は有効ノードの平均 CPU 利用率を監視し，閾値との比較により行う．ユーザは初期ノード数，ノードの投入，除去数，閾値，制御間隔を制御ポリシーとして設定することが可能である．

IO 帯域の調整は，2.3 で述べたストレージ仮想化機構の動的デクラスタリングを用い，サーバプール内で対象ファイルへの IO アクセスの並列度を 2 段階に変更することにより行われる．ユーザはアプリケーション開始時点では未利用であり分割コピーの作成先となるディスクの数，閾値，制御間隔を制御ポリシーとして設定することが可能である．

4. 並列データマイニング

本論文では大規模データを扱うアプリケーションの一例として，代表的なデータマイニング手法である相関ルール抽出処理を取り上げる．相関ルール抽出処理はノード間で大量のデータを交換するため，負荷の偏りは強く性能に悪影響を及ぼす．また，その振る舞いは問題設定に依存する．すなわち，相関ルール抽出処理は複数のパスから構成されるが，パス毎に CPU バウンドであったり，IO バウンドであったりするという特徴を有する．同時に，その挙動は一般に事前に予測することは極めて困難である．動的負荷分散と動的資源調整の評価に適したアプリケーションであるといえる．

相関ルールは以下のように定義される．全てのアイテムの集合を $I = \{i_1, i_2, \cdots, i_m\}$，トランザクションデータベースを $D = \{t_1, t_2, \cdots, t_n\}(t_i \subseteqq I)$ とした際に，各要素 t_i をアイテム集合 (itemset) と呼び，k 個の組合せのものを長さ k のアイテム集合 (k-itemset) と呼ぶ．相関ルールは $X \rightarrow Y$ で表現される．ここで，$X, Y \subseteqq I, X \cap Y = \emptyset$ とする．相関ルールは支持度 (support) と確信度 (confidence) の 2 つのパラメータを持つ．アイテム集合 X の支持度 $sup(X)$ は，D 全体に対しトランザクションが X を含む確率を表す．相関ルール $X \rightarrow Y$ の支持度 $sup(X \rightarrow Y)$ は，D 全体に対しトランザクションが X と Y を共に含む確率 $sup(X \cup Y)$ を表す．また，相関ルール $X \rightarrow Y$ の

確信度 $conf(X \rightarrow Y)$ は，D 全体に対し X を含むトランザクションが X と Y を共に含む条件付き確率 $sup(X \rightarrow Y)/sup(X)$ を表す．

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 $sup(X \rightarrow Y)$ の最小値及び確信度 $conf(X \rightarrow Y)$ の最小値が与えられた際に，これらを満足する全てのルールを見出すことである．この処理は，Step 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，Step 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 つのステップで行われる．

相関ルール抽出処理の Step 2) 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，比較的軽い負荷の処理であるのに対し，Step 1) は巨大なトランザクションデータベースを繰り返し走査し支持度を調査するため，重い負荷の処理である．このため，相関ルールの抽出アルゴリズムに関する研究は Step 1) の効率化に焦点を当てたものが中心であり，本論文では以降 Step 2) に関する議論は行わず，Step 1) の処理に限って述べる．

相関ルール抽出処理の代表的なアルゴリズムとしては，Apriori [7] が良く知られており，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) [10] を利用する．ここで，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．HPA は PC クラスタ内の各ノードに SEND 及び RECV の 2 つのプロセスを配置し，$k = 1$ を初期値として，以下のように L_k を求める手続きであるパス k を逐次的に行う．

（1） SEND プロセスはラージアイテム集合 L_{k-1} をもとに長さ k の候補アイテム集合 C_k を作成する．ただし，$k = 1$ の時は，候補アイテム集合 C_k は全てのアイテムの集合 I に等しいとする．C_k 内の各候補アイテム集合にハッシュ関数を適用することにより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を走査し，長さ k のアイテムの組合せを逐次作成すると共に，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．

（4） $L_k \neq \emptyset$ の場合，k に $k+1$ を代入して (1) へ戻る

上記では，(2) における SEND プロセスの長さ k のアイテムの組合せ作成処理，及び RECV プロセスのハッシュ表検索が重い負荷となる．一般的な小売り履歴データ等では，最も多くの候補アイテム集合が発生するパス 2 が CPU 演算能力を必要とし，実行時間の大半を占める．一方，パス 3 以降の後半のパスにおいては，k が大きくなるに従い演算負荷が減少し，処理は IO 性能に依存する [11]．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 IO 性能を向上させることが全体の高速化に結びつく．

5. SAN 結合 PC クラスタを用いた実装と評価実験

5.1 SAN 結合 PC クラスタと実装

実験システムとしてファイバチャネルによる SAN を用いた SAN 結合 PC クラスタを開発した．概要を図 6 に示す．PC クラスタ内では最大 32 台のノード及び最大 32 台のディスクがギガビットイーサネット及びファイバチャネルによって同時接続可能である．ノードは 3 種類 (Pentium III / II / Pro) を準備し，各実験毎にその組合せを変更した．全てのノードはギガビットイーサネットスイッチに接続されており，また，4 台のノードを以って 1 つの FC-AL(ファイバチャネル・アービトレイテッドループ) を構成し，FC-AL を介してファイバチャネルスイッチに接続される．4 台のディスクアレイ筐体はそれぞれ 8 台のディスクドライブを格納し，4 台のディスクドライブ毎に 1 つの FC-AL を構成し，FC-AL を介してファブリックスイッチに接続される．また，32 台のノードとは別に，メタサーバ及び負荷分散器が動作する制御 PC 及びクライアント PC が用意されている．

SAN 結合 PC クラスタ上に，ストレージ仮想化機構とそれを用いた動的負荷分散，動的資源調整を実装し，さらに当該システム上で並列データマイニングアプリケーションを構築した．ストレージ仮想化機構の LSM はユーザ空間 API(Application Program Interface) として，メタサーバは制御 PC 上で動作する常駐プロセスとして，それぞれ実装した．並列データマイニングは各ノードにアプリケーションプロセス

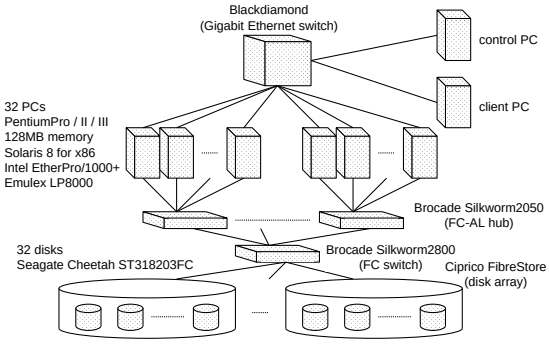


図 6: SAN 結合 PC クラスタの構成
Fig. 6 Organization of SAN-connected PC cluster

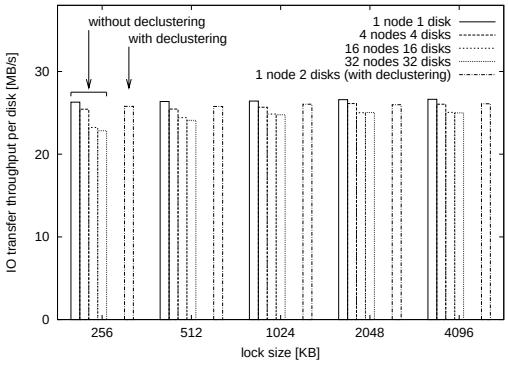


図 7: ストレージ仮想化機構の IO 基本性能
Fig. 7 Basic IO performance of storage virtualizer

である RECV，SEND を配置し，制御 PC 上で負荷分散器を実行する．全てのノード間通信は TCP/IP によるソケット接続を用いる．

本実験中では，非共有ディスク方式における負荷分散と，提案する共有ディスク方式におけるストレージ仮想化機構を用いた負荷分散を比較する．非共有ディスク方式では負荷分散器は [9] の制御手法により，ノード間でメモリ上のハッシュラインとディスク上のトランザクションデータを交換することによる演算負荷とデータ量の調整を行う．一方，ストレージ仮想化機構では負荷分散器はノード間でメモリ上のハッシュラインを交換することによる演算負荷の調整 [6] を行う．この際，双方とも負荷マイグレーション駆動のための閾値を 0.2 とした．

5.2 ストレージ仮想化機構の基本性能

ストレージ仮想化機構における共有読み込みの基本性能を測定した．この実験には Pentium III 800MHz のノードを用いた．先ず，ノードとディスクを同じ台数用意し，共有読み込みを行った．対象ファイルはディスク間に均等に分割され，各ディスク内では連続した領域に格納されている．この時，ディスク間で負荷が偏らないようにメタサーバは共有読み込み要求に対して各ノードのアクセス先ディスクをラウンドロビンで分散した．ノード数を 1 台から 32 台，メタサーバのロック粒度を 256KB から 4096KB まで変化させ，ディスク 1 台あたりの平均スループットを計測した．ディスクアクセスバッファは 64KB である．測定結果を図 7 に示す．ノード数，ディスク数の増加に対し，大きな性能の低下は見られない．よって，メタサーバ問い合わせを行うストレージ仮想化機構及びその共有読み込みは，性能上の障壁とはならないことが分かる．

次に，1 台のノードと 2 台のディスクを用意し，一方のディスクのファイルを読み込むと同時に，動的デクラスタリングを用い，もう一方のディスクへコピー作成を行った．同様に，メタサーバのロック粒度を 256KB から 4096KB まで変化させ，読み込みのスループットを計測した．ディスクアクセスバッファは 64KB である．測定結果を，動的デクラスタリングを行わない場合と比較した結果を図 7(with declustering) に示す．計測された未利用ディスクへの動的デクラスタリングによるコピー作成のオーバヘッドは非常に小さい．このため，SAN 結合 PC クラスタ環境において実行時に分割コピーを作成することは合理的であることが分かる．

5.3 ストレージ仮想化機構を用いる動的負荷分散

並列データマイニングアプリケーションにより，ストレージ仮想化機構を用いた動的負荷分散を評価するため，表 1 のノードの組合せ，トランザクションデータベース配置を用意した．ノードの組合せは c0 から c3 で CPU 演算能力の偏りが，トランザクションデータベースの配置では d0 から d2 でデータ量の偏りが大きくなり，c0d0 が偏りのない最も容易な条件，c3d2 が最も苛酷な条件となる．実験で用いたトランザクションデータベースは，小売りデータを模す手法 [7] を用いて生成し，この際，アイテム種類の数は 5000，1 トランザクションあたりの平均アイテム数は 20 とした．データマイニングの問題設定は最小支持度を 0.7%とした．ディスクアクセスバッファは 64KB，メタサーバのロック粒度は 512KB，ネットワークアクセスバッファは 8KB とした．負荷分散器は 1 秒毎にアプリケーションプロセスから統計情報を収集し，5

表 1: ノード及びデータの配置

Table 1 Experimental setup of nodes' CPUs and disks' transaction database

Skew of CPU between nodes

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

Skew of data volume between disks

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)

表 2: パス 2 の実行時間の比較

Table 2 Comparison of execution times of pass 2

CPU skew	system architecture	data skew d0	d1	d2
c0	SN	100	138	162
	SN+LB	100†	112	114
	SV	98	98	98
	SV+LB	98†	98†	98†
c1 (注3)	SN	99	141	166
	SN+LB	95	101	112
	SV	93	94	94
	SV+LB	93†	94†	95†
c2	SN	128	180	220
	SN+LB	102	118	147
	SV	106	107	107
	SV+LB	96	98	98
c3	SN	226	342	412
	SN+LB	113	193	261
	SV	159	160	160
	SV+LB	99	99	99

SN = shared nothing, SV = storage virtualizer
+LB = with load balancer

秒毎に負荷分散のための負荷マイグレーション判断を行う．

各 CPU 演算能力及びデータ量の偏りの組合せに対し，実行時間の最も多くを占めるパス 2 の実行時間を測定した．この際，以下の 4 つの場合を比較した．

- SN: 非共有ディスク方式において負荷分散器を適用しない場合
- SN+LB: 非共有ディスク方式において負荷分散器を適用した場合
- SV: 共有ディスク方式のストレージ仮想化機構上で負荷分散器を適用しない場合
- SV+LB: 共有ディスク方式のストレージ仮想化機構上で負荷分散器を適用した場合

非共有ディスク方式における計測は，SAN 結合 PC クラスタにおいてノード i をディスク i に対応付ける制限を加えることにより実現した．一方，ストレージ仮想化機構においては共有読み込みを用いて計測した．表 2 に測定結果を示す．なお，CPU 演算能力の差による処理時間差と負荷分散効果を区別するために，CPU 演算能力及びデータ量いずれも偏りのない c0d0 における SN の場合を基準とすべく 100 とし，総 CPU クロック数を以って正規化を行った値を示す(注4)．表中 † は負荷分散器において負荷マイグレーションが駆動されなかったことを示す．なお，幾つかのケースで実行時間が 100 未満となるが，これは SN の場合においてトランザクションデータベースを均等にディスク群に分配した場合にも，相関ルール抽出処理の演算負荷はハッシュを用いているため，必ずしも完全には均衡化されないことによる．

5.3.1 ストレージ仮想化機構上で負荷分散器を適用しない場合の動的負荷分散

表 2 のストレージ仮想化機構単独での負荷分散を検証する．データ量の偏りに関しては，d0 と d1,d2 の比較により，ストレージ仮想化機構上では偏りによる悪化がほとんど測定されないことから，ストレージ仮想化機構によりほぼ吸収可能であることが分かる．例えば，c0 ケースにおいては，実行時間が d0，d1，d2 の各ケースで SN の場合は 100,138,162 と悪化しており，SN+LB では 100,112,114 にまで改善することが可能である．しかし，SV の場合の実行時間は 98, 98, 98 であり，ほとんど悪化が計測されず，ストレージ仮想化機構が優位であることが分かる．

CPU 演算能力の偏りに関しては，c1 のような緩やかな偏りの場合，負荷分散器がなくともストレージ仮想化機構によって偏りの吸収が可能であることが分かる．c1d0 のケースで SV の場合の実行トレースを

(注3)：c1 における SV+LB では負荷マイグレーションは駆動されない．このため，統計情報収集のオーバヘッドにより SV+LB に比べ SV が若干有利となるが，表 2 ではその精度により，c1d0，c1d1 の場合はその差が明らかではない．

(注4)：全てのケースでパス 2 は CPU バウンドであり，本実験においては各ノードのメモリや IO 性能がほとんど影響を与えないため，この正規化は厳密には正確でないが，方式間の比較を行うには十分であるといえる．

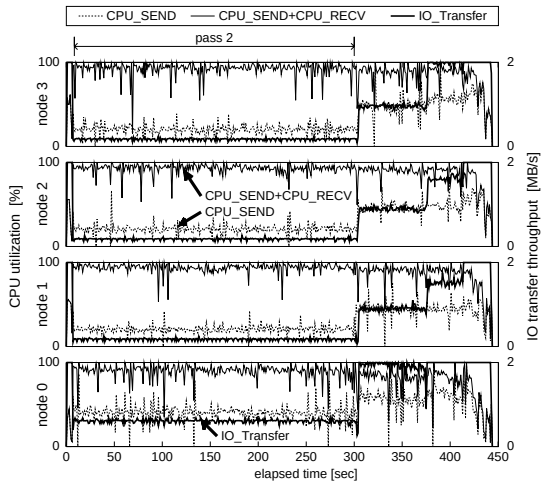


図 8: CPU 利用率 (左軸) と IO 転送スループット (右軸) の実行トレース - SV, c1d0

Fig. 8 Execution trace of CPU utilization (left Y-axis) and IO transfer throughput (right Y-axis) - SV, c1d0

図 8 に示す．図では下から node [0-3] の各資源を表し，点線 (CPU_SEND) は SEND の CPU 利用率，実線 (CPU_SEND+CPU_RECV) は SEND と RECV の CPU 利用率の和，太線 (IO_Transfer) は SEND のディスクの IO 転送スループットを表す(注5)．時刻約 10-300 秒がパス 2 である．ハッシュテーブルは各ノードに等量分散されているため，相対的に速い CPU を持つ node 0 の $L_{RECV i}$ は少なくなるが，余剰 CPU 演算能力を使用して SEND が多くのデータ処理を行っている．一方，node [1-3] の低速ノードでは CPU 演算能力のほとんどを RECV 処理が使用しているが独占はしておらず，node 0 に対してボトルネックになる程ではなく，全ノードの CPU 演算能力が有効に利用されている．

一方，1 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 演算能力の偏りである c2 や c3 ケースでは，SV では不十分である．これは，d0 のデータ配置の下で，c0 から c3 にかけて実行時間が 98 から 159 へと 62%悪化していることから分かる．しかし，加えてデータ量の偏りがある複合ケースの場合，例えば，c3 のケースで比較すると，SN+LB でも d1，d2 のデータ配置において実行時間がそれぞれ 193，261 であるのに対し，SV の場合ではそれぞれ 160，160 と有利な結果を示している．これは非共有ディスク方式の PC クラスタにおける大規模データを扱うアプリケーションの実行時間は，負荷分散器を用いたとしてもデータ量の偏りに影響を受けやすいことを示している．

(注5)：パス 2 における IO 転送スループットの差を確認するため，図 8 において各ノードの IO 転送スループットの最大目盛を 2[MB/s] としているが，パス 3 以降の後半のパスでは 2[MB/s] を超えるスループットを呈している場合がある．

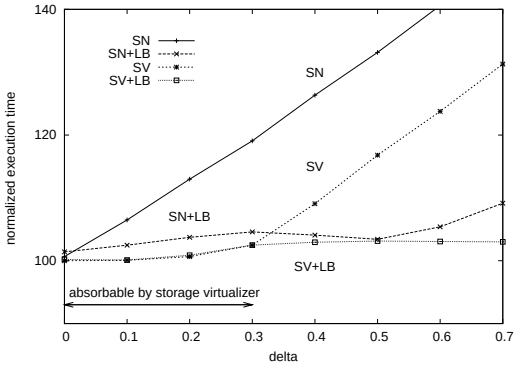


図 9: ストレージ仮想化機構の偏り吸収能力

Fig. 9 Skew absorbability of storage virtualizer

さらにストレージ仮想化機構の偏り吸収効果を示したグラフが図 9 である．ここでは，c0d0 のケースを用い，本来 node 1 に割り当てられる予定のハッシュラインのうち，割合 $\delta(0 \leqq \delta \leqq 1)$ 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，その偏りの度合いは δ が大きく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．非共有ディスク方式である SN の場合，偏りの増加に伴い実行時間が段階的に悪化しているが，ストレージ仮想化機構上では，SV の場合でも約 30%までの偏りが自動的に吸収可能である．一方，約 30%を越える偏りの場合，実行時間が悪化することから負荷分散器による演算負荷の調整が必要である．しかしその場合でも，ストレージ仮想化機構が約 30%までの負荷分散器の制御誤差を吸収することが可能であり，性能の向上に寄与することが期待される．

5.3.2 ストレージ仮想化機構上で負荷分散器を適用する場合の動的負荷分散

ストレージ仮想化機構のみでは解決できなかった苛酷な CPU 演算能力の偏りケースでは，特定のノードがシステム全体のボトルネックとなっている．負荷分散器はボトルネックとなるノードを検知し，その演算負荷を他のノードに移送することにより，演算負荷を

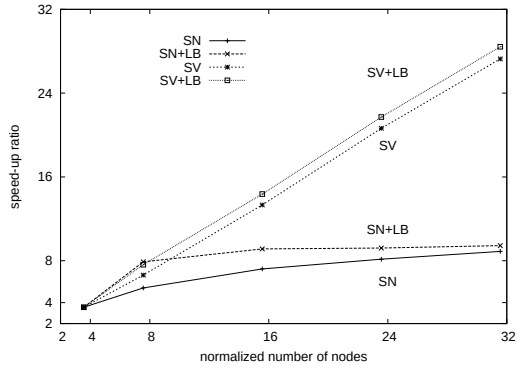


図 10: パス 2 のスピードアップ曲線
Fig. 10 Speed-up curve of pass 2

調整する．ストレージ仮想化機構上で負荷分散器を用いた SV+LB の場合，表 2 において全ての偏りの下で正規化実行時間が 100 以下となることが示す通り，あらゆる偏りを均衡化することが可能となった．

5.3.3 多ノード環境における動的負荷分散

多ノード環境における負荷分散を検証するため，先の d0 のデータ配置，c3 の CPU 配置の下でさらに Pentium III 800MHz ノードを数台追加し，パス 2 の実行時間を計測した．ノード数 4 を起点としたスピードアップ曲線を図 10 に示す．この際，横軸は 800MHz の CPU を 1 とし，総クロック数で正規化を行った．例えば，800MHz のノード 3 台に 200MHz のノードを 1 台用いる場合，正規化ノード数は 3.25 となる．図 10 からストレージ仮想化機構上では，負荷分散器の適用に係わらず SV，SV+LB ともに高いスケーラビリティが得られているが，非共有ディスク方式の SN，SN+LB では，ノード数が増加してもデータがノードに依存し，データ量の偏りの影響を受けるため，ノード数 8 以降の性能改善は見られない．

以上のことから，本論文で提案した動的負荷分散の手法は，優れた偏り均衡化能力を有していることが示された．

5.4 ストレージ仮想化機構を用いる動的資源調整

並列データマイニングアプリケーションにより，ストレージ仮想化機構を用いた動的資源調整を評価するため，表 3 に示す 32 台のノードからなるサーバプールと 4 台のディスクからなるストレージプールを設定した．トランザクションデータベースは前節と同じパラメータで作成し，データマイニングの最小支持度は 1.5%とした．

表 3: サーバプールとストレージプールの設定
Table 3 Setup of server pool and storage pool

server pool	storage pool	
node [0-31]	disk 0	disk [1-3]
Pentium III 800MHz	88,000,000 (7.3GB)	unused disks

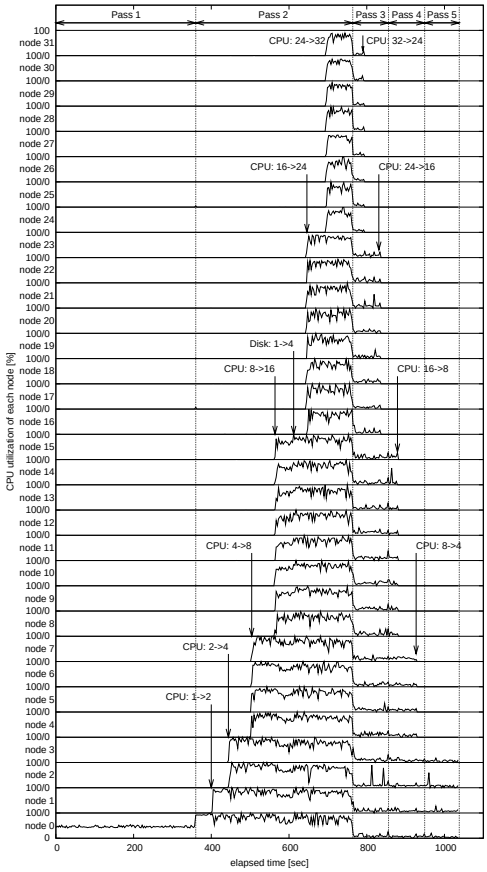
number of transactions (size of data volume)

並列データマイニングのアプリケーション開始時には 1 ノード (node 0) のみでプロセスが実行され，1 ディスク (disk 0) のみがトランザクションデータベースを有している．負荷分散器は有効ノードの平均 CPU 利用率を観測し，閾値との比較によりノードの投入，除去を決定する．投入，除去の閾値は 80%, 20%とした．ノード数の増減に関しては，$1 \leftrightarrow 2 \leftrightarrow 4 \leftrightarrow 8 \leftrightarrow 16 \leftrightarrow 24 \leftrightarrow 32$ と変化するように指定した．また，共有読み込みと動的デクラスタリングを用いた IO 帯域の調整を行い，ディスクの投入，除去の IO 利用率の閾値 θ_{inject},θ_{remove} は 88%，12%とした．5.1 の結果からディスク 1 台の最大 IO 帯域 T_{max} は 25MB/s としたため，これらの閾値は 22MB/s，3MB/s の IO 転送スループットに相当する．disk [1-3] を分割コピーの作成先として利用する．このため，ディスクの数は $1 \leftrightarrow 4$ と変化し得る．資源調整の判断は負荷分散器が 40 秒毎に行い，メタサーバが 20 秒毎に行うものとした．(注6)

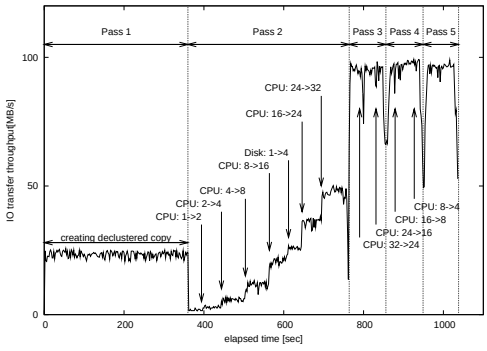
CPU 演算能力の調整及び IO 帯域の調整を行う場合の実験結果の実行トレースを図 11 に示す．図 11(a) では各ノードの CPU 利用率を，図 11(b) ではシステム全体の IO 転送スループットを示す．

パス 1 は IO バウンドであるため，ノード数の変化は見られず，動的デクラスタリングによる分割コピー作成が行われる．その後，パス 2 が開始し負荷分散器は CPU バウンドを検知し，直ちに node 1 を投入し，総計ノード数を 2 にする．この時，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ともに，負荷マイグレーションによって RECV プロセス

(注6)：投入，除去閾値の組合せ選択にあたり，CPU 演算能力に関しては (90%, 10%)，(80%, 20%)，(70%, 30%)，(60%, 40%)，IO 帯域に関しては (96%, 4%)，(88%, 12%)，(80%, 20%)，(72%, 28%) の各場合について予備実験を行った．この結果，CPU 演算能力に関しては (90%, 10%) の場合，IO 帯域に関しては (96%, 4%) の場合，それぞれ多ノード同期に伴う CPU 利用率の低下や，同一ディスクへの多ノードアクセスに伴う IO 転送スループットの低下により，ノード数 32，ディスク数 4 までの資源投入が出来なかった．一方，それ以外の場合では可能であったため，本論文の実験では，このうち最も投入閾値が高い，即ち最も除去閾値が低い組合せを採用した．



(a) CPU utilization of each node



(b) Total IO transfer throughput

図 11: 実行トレース - 動的資源調整
Fig. 11 Execution trace - dynamic resource allocation

にハッシュラインが配られ，負荷が均衡化する．さらに依然として CPU バウンドであるため，node [2-3], node [4-7]，node [8-15] が逐次的に投入され，ノード数は 2 から 16 へと段階的に増加する．ノード数が 16

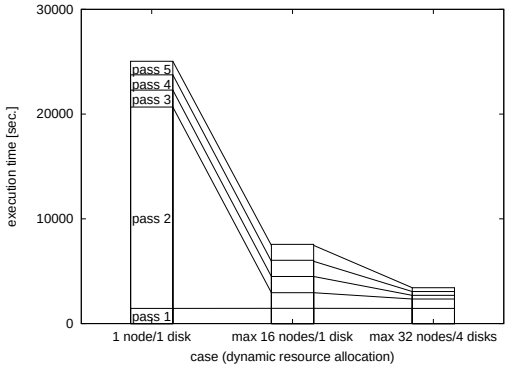


図 12: 実行時間の比較
Fig. 12 Comparison of execution time

になった際に，全ノードが CPU バウンドから IO バウンドへと移行し，ノードの追加投入は停止する．総 IO 転送スループットが約 25MB/s まで上昇し，これ以上のスループットを見込めないためである．その後，メタサーバは統計情報から disk 0 が IO バウンドしていることを検知し，分割コピー空間 (disk [1-3]) への並列 IO アクセスを活性化する．このため，IO 帯域が拡張し，ディスクのスループットが増大する．この結果，16 台のノードは再び，CPU バウンドに移行する．さらに node [16-23], node [24-32] を段階的に投入し，CPU 演算能力を増大させる．最終的に，サーバプール内 32 台全てのノードが用いられる．

パス 3 以降では，再び IO バウンド化しシステム全体で約 94MB/s のスループットを呈している．この時，CPU 利用率の平均は大きく低下し，20%未満となる．このため，負荷分散器は現状のノード数は必要ないと判断し，ノード数を段階的に減少させる．最終的にノード数は 4 台まで減少し，処理を終える．アプリケーションがパス 3 以降の処理においては IO 帯域を要求するため，IO 帯域の縮小は起こらなかった．全実行時間は 1036 秒であった．

動的資源調整による性能改善を実行時間により確認するため，表 3 のトランザクションデータベース長の 4 倍のデータ (約 29GB) を用意した．資源調整を行わない場合，CPU 演算能力の資源調整のみを行う場合，及び CPU 演算能力，IO 帯域双方の資源調整を行う場合について各パスの実行時間を測定した．結果を図 12 に示す．

CPU 演算能力の調整のみを行う場合，最大で 16 ノード 1 ディスクまで拡張が行われ，パス 2 が大幅に

改善しているが，パス 3 以降は改善しない．一方，動的デクラスタリングを用い IO 帯域の調整を併せて行う場合，最大で 32 ノード 4 ディスクと予め設定した資源全てを使用した．パス 2 は資源調整を行わない場合と比較し，約 22 倍に改善しており，これは，逐次的に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．パス 1 を含んだ全パスの実行時間の性能改善は約 7 倍であった．

以上のことから，ストレージ仮想化機構及び負荷分散器が協調することにより，予めサーバプール，ストレージプールとして設定した資源を適切に割り当て，CPU 演算能力と IO 帯域の調整が可能であることが示された．

6. むすび

共有ディスク方式のストレージアーキテクチャを採用する PC クラスタにおいて，共有読み込み及び動的デクラスタリングなる新たな機能を有す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と動的資源調整の方式を設計した．また，SAN 結合 PC クラスタにおいて，アプリケーション例として並列データマイニングを取り上げた実装実験により提案手法の有効性を示した．

提案した共有ディスク方式の PC クラスタにおけるストレージ仮想化機構では，共有読み込みによりノードからの要求に応じてデータを配分することが可能となる．このため，ストレージ仮想化機構はディスク間のデータ量の偏りをほぼ吸収可能であると同時に，ノード間の緩やかな演算負荷の偏りを吸収することができる．アプリケーションの負荷分散器はディスク上のデータ量を考慮する必要がなく，ノード間の演算負荷の調整のみを行うことになる．このため，ストレージ仮想化機構においては，非共有ディスク方式の負荷分散器と比較して，負荷分散器の構成を単純化することが可能となる．アプリケーションの知識を用いる負荷分散器が大きな演算負荷の偏りを除去し，一方，ストレージ仮想化機構がその誤差を自動的に吸収し，両者が相補的に機能することにより，負荷偏りに対して均衡化能力の向上を図れることを示した．

また，ストレージプールを管理するストレージ仮想化機構と，サーバプールを管理する負荷分散器が協調することにより，ユーザが予め設定したノード及びディスクを動的に割り当て，性能の改善に寄与することが可能である．従来，並列計算機における資源調整はサーバ資源の調整に主眼が置かれて来たが，動的デクラスタリングを用いることにより IO 資源を含めた調整が可能になることを示した．

今後は，本論文で提案した動的負荷分散及び動的資源調整の方式を他のアプリケーションに応用し，実験により有効性を検証する．また，複数のアプリケーションを対象とした資源調整に関して研究を進める予定である．

謝辞

本研究の一部は，文部科学省科学研究費補助金基盤研究 (S)13852015 の助成により行われた．

文　献

[1] S.R. Soltis, T.M. Ruwart and M.T. O'Keefe, "The Global File System," Proc. 15th NASA Goddard Conf. on Mass Storage Systems, pp.319-342, College Park, USA, Sept. 1996.

[2] Y. Shinkai, Y. Tsuchiya, T. Murakami and J. Williams, "HAMFS File System," Proc. 18th IEEE Symposium on Reliable Distributed Systems, pp.190-201, Lausanne, Switzerland, Oct. 1999.

[3] R.C. Burns, R.M. Rees and D.D.E. Long, "Semi-Preemptible Locks for a Distributed File System," Proc. 19th IEEE Intl. Performance, Computing and Communications Conf., pp.397-404, Phoenix, USA, Feb. 2000.

[4] Compaq Computer Corporation, "Compaq Innovation to Transform Enterprise Storage Deployment, Utilization and Management," Jun. 2000.

[5] VERITAS Software Corporation, "Storage Virtualization White Paper," May 2001.

[6] K. Goda, T. Tamura, M. Oguchi and M. Kitsuregawa, "Run-time Load Balancing System on SAN-connected PC Cluster for Dynamic Injection of CPU and Disk Resource — A Case Study of Data Mining Application —," Proc. 13th Intl. Conf. on Database and Expert Systems Applications, pp.182-192, Aix-en-Provence, France, Sept. 2002

[7] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," Proc. 20th Intl. Conf. on Very Large Data Bases, pp.487-499, Santiago de Chile, Chile, Sept. 1994.

[8] M. Oguchi and M. Kitsuregawa, "Runtime Data Declustering over SAN-Connected PC Cluster System," Proc. 18th Intl. Conf. on Data Engineering, p.275, San Jose, USA, Feb. 2002.

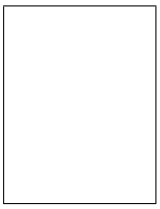
[9] M. Tamura and M. Kitsuregawa, "Dynamic Load Balancing for Parallel Association Rule Mining on Heterogeneous PC Cluster Systems," Proc. 25th Intl. Conf. on Very Large Data Bases, pp.162-173, Edinburgh, UK, Sept. 1999.

[10] T. Shintani and M. Kitsuregawa, "Hash Based Pararellel Algorithm for Mining Assocication Rules,"

Proc. 4th Intl. Conf. on Parallel and Distributed Information Systems, pp.19-30, Miami Beach, USA, Dec. 1996.

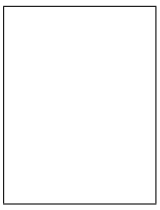
[11] 喜連川 優, "データマイニングにおける相関ルール抽出技法, " 人工知能学会誌, vol.12, no.4, pp.513-520, Jul. 1997.

（平成 x 年 xx 月 xx 日受付）



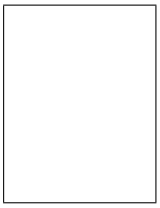
合田　和生

平 12 東大・工・電気卒．平 14 同大大学院工学系研究科電子情報工学専攻修士課程修了．現在，同大大学院情報理工学系研究科電子情報学専攻博士課程に在学中．並列データベースシステム，ストレージシステムに関する研究に従事．平 15 より日本学術振興会特別研究員．



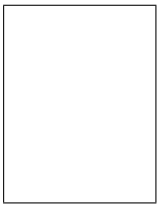
田村　孝之　（正員）

平 3 東大・工・電子卒．平 8 同大大学院工学系研究科情報工学専攻博士課程単位取得退学．平 10 博士 (工学)．NEDO(新エネルギー・産業技術総合開発機構) 最先端分野技術研究員を経て，平 10 三菱電機株式会社に入社．現在，東大生産技術研究所産学官連携研究員．並列データベース処理，Web マイニングに関する研究に従事．



小口　正人　（正員）

平 2 慶大・理工・電気卒．平 7 東大大学院工学系研究科電子工学専攻博士課程了．工博．学術情報センター中核的研究機関研究員，東大生産技術研究所特別研究員，中央大学研究開発機構助教授を経て，平 15 よりお茶の水女子大学理学部情報科学科助教授．ネットワークコンピューティング・ミドルウェアに関する研究に従事．IEEE, ACM, 電子情報通信学会，情報処理学会各会員．



喜連川　優　（正員）

昭 53 東大・工・電子卒．昭 58 同大大学院工学系研究科情報工学博士課程了．工博．同年同大生産技術研究所講師．現在，同教授．戦略情報融合国際研究センター長．データベース工学，並列処理，Ｗｅｂマイニングに関する研究に従事．情報処理学会理事，SNIA-Japan 顧問，ACM SIGMOD Japan Chapter Chair．平成 9,10 年 本学会データ工学研究専門委員会委員長．VLDB Trustee, IEEE ICDE, PAKDD, WAIM ステアリングコミティメンバ．

英文アブストラクト

PC cluster systems have been so far widely used for many applications. At present, shared nothing architecture is employed, where all the resources such as CPU, memory and storage are independent among the nodes. Recently there emerged applications which handle huge amount of data, so called data-intensive applications. For such applications, storage architecture is more influential for the performance. We constructed the PC cluster system where disks are interconnected through the recent storage area network (SAN) technology. We proposed shared file access mechanism and dynamic data declustering strategy. We implemented the experimental system and showed that the dynamic load balancing and dynamic resource allocation system built on top of the above two mechanisms work much more effectively compared with ordinary shared nothing PC cluster system.

英文キーワード

PC cluster, storage area network (SAN), storage virtualizer, dynamic load balancing, dynamic resource allocation