Vol.2009-ARC-183 No.24
Vol.2009-OS-111 No.24
2009/4/23

分散ストレージにおける アクセス特性を考慮したI/Oスケジューリングの提案

森田 和孝[†1]　藤田 智成[†1]　盛合 敏[†1]

ストレージの負荷分散や高性能化, 可用性向上などの観点から, PC クラスタによって仮想的なストレージシステムを提供する分散ストレージシステムが注目を浴びている. しかし既存の分散ストレージシステムは, 複数のクライアントが同一ディスクにアクセスする状況下では限られたバンド幅を共有することや多くのディスクシークが発生することが理由で性能が劣化する問題がある. そのためストレージ仮想化技術などの同一ディスク領域を複数クライアントで共有することが必要な技術を効率よく分散ストレージシステム上で実現させることが難しい. 本研究では, 各ディスクが持つ処理待ちキューのリクエストパタンから, クライアントのアクセス先ディスクを決定する手法を提案する. そして提案手法の実装を行い, 評価実験によって本手法が有効であることを示す.

A Proposal of I/O Scheduling for Distributed Storage Systems Based on Access Pattern Analysis

MORITA KAZUTAKA ,[†1] FUJITA TOMONORI [†1]
and MORIAI SATOSHI[†1]

A distributed storage system built from commodity servers is important to get more load balancing, higher performance, more availability, and so on. But existing systems do not good deal with access from multi clients, especially when they access concurrently, because of sharing narrow bandwidth and causing unnecessaly disk seeks. In a distributed storage system, it is hard to realize a technology sharing the same disk such as storage virtualization. In this paper, we propose an algorithm to decide which disk to access by considering each client's access pattern. We give the implementation and the experimental results show the effectivity of the proposed method.

1. はじめに

ストレージの負荷分散や高性能化, 可用性向上などの観点から, PC クラスタによって仮想的なストレージシステムを提供する分散ストレージシステムが注目を浴びている. しかし既存の分散ストレージシステムは, 高性能計算を利用用途として想定しているものが主流であり, 現在多くの商用共有ストレージにおいて実現されているスナップショット機能やプロビジョニング機能などのストレージ仮想化技術を実現することは想定していない. ストレージ仮想化技術は同一ディスク領域をクライアント間で共有してディスク使用効率を向上させる技術であり, 分散ストレージシステム上で同様の仕組みを実現しようとした場合, クライアント間のディスクアクセス衝突は不可避である. そのため, 複数クライアントが同一ディスクにアクセスする際には, 限られたバンド幅を共有することや多くのディスクシークが発生してしまうことなどの理由により性能が劣化する問題がある. 共有ストレージシステムのように集中型の設計をもたない分散ストレージシステムは, 他クライアントのアクセス状況を知ることができないため, 効率の良いディスクアクセス手法は課題である. 本研究では, 各ディスクが持つ処理待ちキューのリクエストパタンから, 各ディスクがデータを読み書きを行う際に要する時間を評価して, 評価値を元にアクセス先を決定する手法を提案する. さらに, 提案手法の実装を行い, 評価実験によって本手法が有効であることを示す.

本稿の構成は以下のとおりである. まず第 2 節で分散ストレージシステムにおけるディスクアクセスの課題について述べる. 続く第 3 節で第 2 節の課題に対してどのようなアプローチをとって解決するかを説明する. 第 4 節で本研究のストレージシステムのプロトタイプ実装とその性能評価を行い, 第 5 節で関連研究について述べる. 最後に第 6 節で本稿をまとめる.

2. 課題

以下, 本研究における共有ストレージとは, ストレージエリアネットワークに接続して用いる高信頼なディスクアレイ型ストレージシステムとする. また, 分散ストレージとは, コモディティな PC を複数組み合わせて構成された, 高性能化, 可用性の向上を目的とするストレージシステムとする.

†1 NTT サイバースペース研究所
NTT Cyber Space Laboratories

本節では, 現在多くの共有ストレージにおいて実現されているスナップショット機能やプロビジョニング機能などのストレージ仮想化技術を, PC クラスタによる分散ストレージシステム上で実現しようとする時に問題となるディスクアクセスの課題について着目し説明する.

2.1 アクセスパタンを考慮した効率化

ストレージ仮想化技術は実際の物理構成とは異なる仮想的なストレージをクライアントに見せる技術であり, このような技術を既存の分散ストレージにおいて実現しようとすると, ディスクアクセスの衝突が起こる. ディスクはアクセスパタンによって性能が異なるため, ディスクのアクセス状況に応じたアクセス制御をしなくては, 限られたバンド幅を共有することや多くのディスクシークが発生してしまうことなどの理由により, 入出力の並列化による負荷分散や高性能化が十分に達成できない. 例えば複数のプロセスが同時にストレージシステムにシーケンシャルアクセスした場合, それぞれのシーケンシャルリクエストをディスクがシーケンシャルアクセスとして処理することを考えなくては, アクセスが衝突してランダムアクセスになり, 性能が大幅に劣化する. 共有ストレージシステムのような集中型のアーキテクチャであれば, ストレージシステムのヘッドが全体のアクセス状況を把握してアクセスの割り当てを決定することが可能であるが, システム全体のアクセス状況のような大域的な情報を持たずに並列に入出力が行われる分散ストレージシステムにおいてはそのようなコントロールは難しい.

分散ストレージにおける既存のスケジューリングとしては, Lustre[7] や Ceph[9] など多くのクラスタファイルシステムで利用されている, ハッシュやラウンドロビンなどの確率的な負荷分散手法や, CLBS[10] と呼ばれるストレージシステムの各 PC が I/O キューに溜まっている処理待ちリクエストの量を定期的にクライアント全体にマルチキャストすることで, 処理待ちリクエストが少ないディスクへアクセスする手法がある. これらの手法は PC クラスタの各ディスクに平等にアクセスしようとするが, 各ディスクがもつキューの内容を考慮していないため, 余分なディスクシークなどが発生してしまうなどの可能性がある.

よってこの問題を解決するためにはハッシュやラウンドロビンなどの確率的な負荷分散や, 単純に処理待ちリクエストの少ない PC へリクエストを送信するのではなく, ストレージシステムの各 PC の処理待ちリクエストの内容を考慮したスケジューリングを行う必要がある.

2.2 クライアントの優先度設定

クライアント間のディスクアクセスの衝突が不可避な状況下において, クライアントの優先度設定は重要である. 多くのクライアントが同時にディスクアクセスを行うと各クライアントのディスク I/O 性能が劣化するが, そういった状況下において, クライアントごとの優先度を設定できると重要な役割をもつサーバのディスク I/O 性能を過負荷時にも落とさないことができる. 集中型アーキテクチャである共有ストレージにおいては, すべての I/O はストレージシステムのヘッダを経由して行われるため, このような帯域制御は容易に実現可能であるが, I/O がストレージの PC とクライアントで個別に行われてしまう分散ストレージにおいては, 同様の技術はそのまま実現することはできない. 現状, 分散ストレージの性能に優先度を設定するためには, クライアント間に比例関係を設定してそれにしたがったスケジューリングを行う手法[8] がある. しかし比例関係にしたがったスケジューリングでは, 常にクライアント間の性能の比例関係を保つため, ストレージシステムに余裕がある時にでも優先度の低いクライアントの性能を抑えてしまう. 実際には, はじめから帯域を制限するのではなく, 過負荷な状況においてはじめて一部のクライアントの帯域を絞った方がシステムの資源を有効活用でき, 望ましい. もし通常時においても一部のクライアントの性能を抑えることが必要であるなら, 比例関係とは関係なく, 絶対値指定で直接性能を抑える仕組みを入れれば十分である.

3. 提 案 手 法

本研究の提案手法は同じデータを複数のディスクに書き込むことで耐障害性を実現しているストレージシステムを対象とする. また, データ書き込みを行うディスク領域の予約は別の機構によって実現されているとし, データの書き込み時の排他制御等については本稿で触れない.

本研究では前節で取り上げたディスクアクセスの課題を解決するために, 読み書きに要する時間を評価する関数を用意し, 評価関数の値が最も低いディスクに優先的にアクセスするというアクセス制御を提案する. 評価関数を定義するにあたって, ディスクアクセスに要する時間は読み書きを行うデータサイズとシーク回数のふたつの要素によって決定するというシンプルなモデルを考える. その上で, 各ディスクが現在読み書きしているオフセット位置と各ディスクの I/O キューの中身の情報から, 評価値を求める. クライアントはこの評価値を元に, 読み込み操作においては複数のディスクにあるデータの複製のうちどのデータにアクセスするのがよいのか, 書き込み操作でかつデータ領域の新規割り当て時においては全ストレージサーバのうちどのディスクに書き込みを行うのがよいのかを決定する. なお書き込み操作でかつデータ領域がすでに割り当てられている場合においては, 書き込むディスクの選択肢がないため本研究のスケジューリングの対象外とする.

表 1　記号の定義

記号	定義
h	現在読み書きしているオフセットの位置
R	キューにあるリクエストの数
o_r	リクエスト r がアクセスする時のオフセット
l_r	リクエスト r のサイズ
a_r	リクエスト r が単位長さ読み書きする際にかかるコストの重み係数
b	ヘッド位置をシークさせる際に必要なコストの重み係数

3.1　評価関数の設定

本研究では表 1 の記号を用いて評価関数を定義する．まず, 各記号の定義について説明する. h は現在読み書きしているオフセットの位置であり, アクセスがないと定期的に初期位置にリセットされる. R は I/O キューにあるリクエストの数である. o_r はリクエスト r が読み書きを行うディスク上のオフセット位置であり, l_r はリクエスト r のデータサイズである. a_r はリクエスト r が単位長さ読み書きする際にかかるコストの重み係数であり, 各ディスクの性能やクライアントからの物理的距離に応じた係数である. この係数を調整することでアクセスが高速なディスクを優先的に用いることが可能でなる. b はディスクがシークを行うコストの重み係数である. これらの記号定義のもと, $o_0 = h, l_0 = 0$ とすると読み書きに要する時間を評価する関数 $cost$ は

$$cost = \sum_{r=1}^{R} (a_r l_r + b \min(1, |o_r - (o_{r-1} + l_{r-1})|))$$

と定義できる. 関数 $cost$ の第 1 項は読み書きを行うデータの長さに起因する項であり, リクエスト r が要求するデータの長さに比例して読み書きに要する時間が大きくなることを表している. 第 2 項はシーク回数によって決定される項であり, シークの回数に比例して読み書きに要する時間が大きくなることを表している. クライアントは全ディスクの中でのこの計算式の値が最も低いディスクから順にアクセスするディスクを決定する. このように現在のディスクアクセスの状況を考慮したアクセス制御を行うことにより, 既存の手法では考慮していなかった, アクセスパターンによって性能が異なるということを考慮したアクセスが可能となる.

この評価関数のモデルは, 評価関数の係数を調整することでディスクアクセスの制御を柔軟に変更することが可能でなる. 例えば, このモデルはリクエストの要求元が変わる際のシークのコストを十分大きくすると, クライアント間のディスクアクセスの衝突を回避するように評価値が決定されるため, 帯域公平性によりよいものとなる. 具体的には, ペナルティ関数

$$q(x, y) = \begin{cases} 1 & \text{if } x \text{ and } y \text{ are from a same client} \\ q\ (> 0) & \text{otherwise} \end{cases}$$

を導入し, これをシークに要するコストの重みとして与える. このとき関数 $cost$ は

$$cost = \sum_{r=1}^{R} (a_r l_r + q(r, r-1) b \min(1, |o_r - (o_{r-1} + l_{r-1})|))$$

となる. また, 優先度が高いクライアントから低いクライアントにディスクアクセス元が変わるときのペナルティ関数は大きく, 優先度が低いクライアントから高いクライアントにディスクアクセス元が変わるときのペナルティ関数は小さくすることで, アクセスが過多な状況においても優先度が高いクライアントに優先的にストレージシステムを使用させることができる. これによってクライアントの優先度設定を行うことが可能になる. このように, 評価関数の係数を調整してクライアント間のディスクアクセスの衝突を避けることで帯域の制御を行うことにより, 既存の分散ストレージ上における帯域制御技術や優先度設定では生じていたアクセスの衝突による性能の劣化を回避することができる.

3.2　アルゴリズム

評価式 $cost$ には, ディスクの I/O キューに関する情報などが含まれているため, そのまま適用すると計算には全ディスクのキューの中身を把握する必要があるように見える. しかし評価式 $cost$ は, 現在の評価値を $cost'$, キューの最終リクエストがアクセスした後のオフセット位置を h' として,

$$cost = \sum_{r=1}^{R} (a_r l_r + b \min(1, |o_r - (o_{r-1} + l_{r-1})|))$$
$$cost' + a_r l_r + b \min(1, |o_r - h'|)$$

であるので, 現在の評価値とキューの最終リクエストがアクセスした後のオフセット位置がわかれば計算可能であり, 実際に把握しなければいけないデータの量は多くない. そのため本研究では, 現在の評価値とキューの最終リクエストがアクセスした後のオフセット位置を定期的にマルチキャストすることによって情報を共有する.

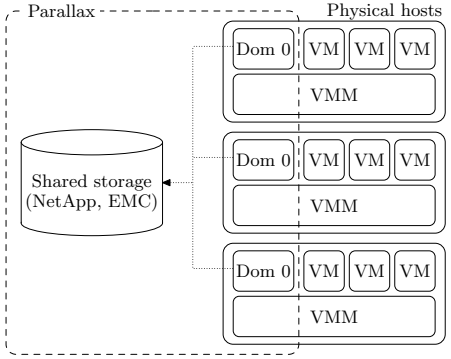

図 1　Parallax

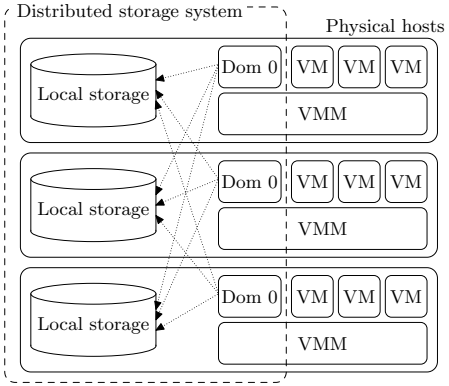

図 2　プロトタイプの構成

4. 実装と評価

本節では提案手法の有効性を確認するためのプロトタイプ実装とその評価を示す.

本プロトタイプの実装は既存のオープンソースソフトウェアである Parallax[5] を拡張する形で行った. Parallax は Xen[2] 用に実装された仮想マシン用のストレージであり, スナップショットからの分岐やプロビジョニング機能などのストレージ仮想化技術が実現されているストレージシステムである. しかし Parallax は図 1 にあるように, 共有ディスクの存在を前提としているため, 図 2 のように分散化する必要がある. しかし単純に分散化しては 2 節で述べたディスクアクセスの問題が生じる. そのため Parallax に 3 節で述べた手法を導入する.

4.1 プロトタイプの構成

図 3 に本プロトタイプの実装のシステム構成図を示す. このシステムにおいて, 仮想マシンはクラスタストレージを Xen の仮想ブロックデバイスとして認識する. 仮想ブロックデバイスに対するディスク I/O の要求はカーネル空間を通って, Xen 上の特権ドメインである domain 0 の tapdisk デーモンに送られる. 仮想マシンからきたディスク I/O の要求を送信するディスクを決定するために, tapdisk デーモンはマルチキャストされた各ディスクの現在の評価値などの情報を元に, データ読み込みの場合はいくつかある複製のうちからどれにアクセスすることがよいか, データ領域の新規割り当ての場合は全ディスクのうちどれにアクセスすることがよいかを求める. 得られた結果を元に, tapdisk デーモンはディスク I/O を行う.

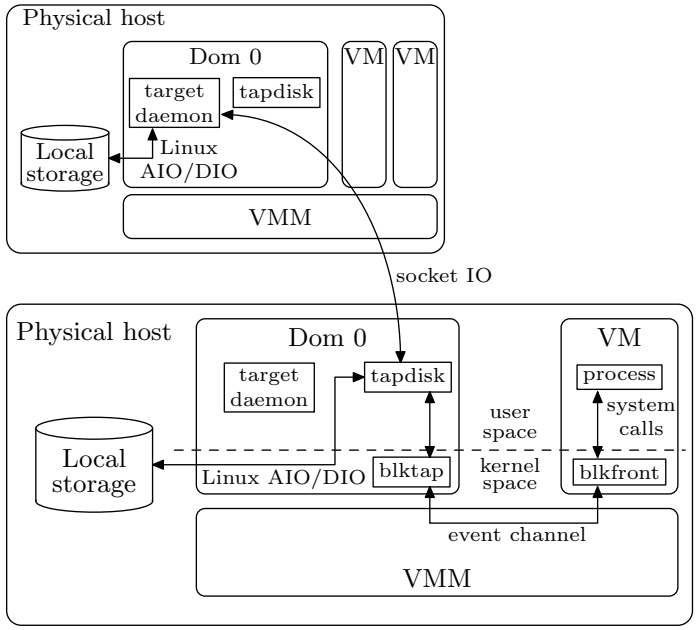

図 3　システム構成図

アクセス先が他サーバのディスクの場合はそのサーバのターゲットデーモンに対してリクエストを転送し, 目的のディスクにアクセスする. そのためローカルのディスクには高速にアクセスできることから, 評価関数のローカルディスクに対する係数 a は他のディスクのものよりも小さくする. ディスク I/O の処理が終了したら, 再びカーネル空間を通って仮想マシンに入出力終了を伝える. 現在の実装では単純化のため write は複製が全てディスクに書き込まれるまで仮想マシンへ処理の終了を伝えない. しかし, 実際には複製のうちどれかひとつがディスクに書き込まれた時点で処理を返すことも可能である. このような一般的な場合のアクセスのコントロールについては今後の課題である.

4.2 評　　価

本節では実装したストレージシステムの基本性能について測定を行い, 性能評価を行った結果を示す.

評価を行った環境はクラスタストレージの台数が 4 台であり, クラスタストレージの各マ

シン上で仮想マシンをクライアントとして動作させた. 各マシンの性能は, CPU が Intel Core 2 Quad 2.40GHz, メモリが 2 MB で, それぞれ 1 TB の SATA 接続のハードディスクをひとつずつ持っている. 各クライアントの仮想マシンイメージはひとつのイメージのスナップショットから分岐したものを用いた. またデータの冗長度は 2 とした. 比較対象の分散スケジュール手法として, 本プロトタイプからキュー情報をマルチキャストする機能をのぞき, ラウンドロビンによってアクセスディスクを決定するようにしたものと, CLBS[10] と呼ばれる処理待ちキューの量のみをマルチキャストする手法を実装したものを用いた. それぞれのクライアントにはベンチマークツール disktest を実行させた.

4.3 アクセスパタンを考慮した効率化

様々なバッファサイズのディスクアクセスを組み合わせて行うことで, キューの内容を評価することの有効性を評価する. まず, 4 台のクライアントをふたつのグループに分け, 一方のグループには大きなバッファサイズでのディスクアクセスを, もう一方のグループには小さなバッファサイズでのディスクアクセスを行わせる. 大きなバッファサイズのディスクアクセスとしては, サイズ 1 MB のランダム読み込みを, 小さなバッファサイズのディスクアクセスとしては, サイズ 4 KB のランダム読み込みを行う. また, それぞれのディスクアクセスを行う台数の割合を変化させることで, 様々な入出力パタンを観察する. 大きなバッファサイズのクライアントのスループットの平均値, 小さなバッファサイズのクライアントのスループットの平均値についての結果を図 4 と図 5 に示す. 図中の横軸は, バッファサイズ 1 MB でアクセスをしたクライアント数とバッファサイズ 4 KB でアクセスをしたクライアント数を'/' で区切って表示している. 図 4, 図 5 の結果から, 提案手法によってキューの処理待ちリクエストパタンを考慮したアクセス決定の効果が現れていることがわかる.

4.4 クライアントの優先度設定

ストレージの基本性能を調べるにあたって, クライアントの優先度設定の観点から性能評価を行った. これらを調べるため, クライアントの数を変化させてベンチマークを行い, その影響を調べた. その際, ひとつのクライアントに対して他のクライアントより優先度を高くする設定を行い, 全クライアントの平均の性能を見ることでストレージシステム全体の性能を, 性能が最大のクライアントの性能を見ることで優先度設定の有効性を確認する. 各クライアントにはディスクアクセスが衝突したときに最も性能が劣化するように, シーケンシャルな読み込みアクセスを実行させ, クライアント台数に対するスループットの変化を調べた. 各手法に対する, クライアントのスループットの平均値と最大値の結果を図 6 に示す. ラウンドロビンによるアクセス先の決定は, キューの情報をマルチキャストする必要がないた

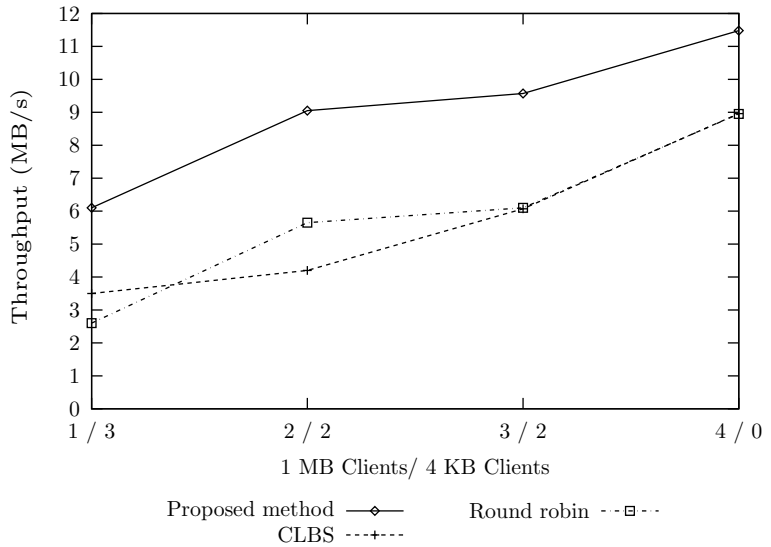

図 4 アクセスパターンを組合せた時のバッファサイズ 1 MB のクライアントのスループット

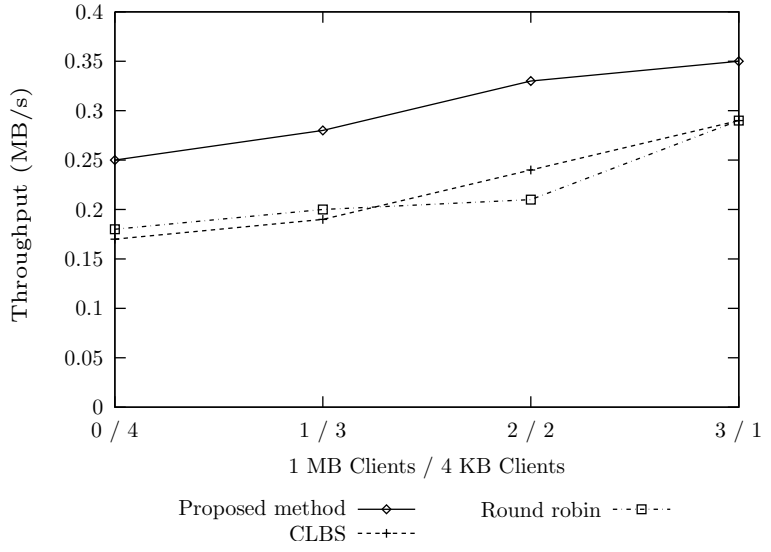

図 5 アクセスパターンを組合せた時のバッファサイズ 4 KB のクライアントのスループット

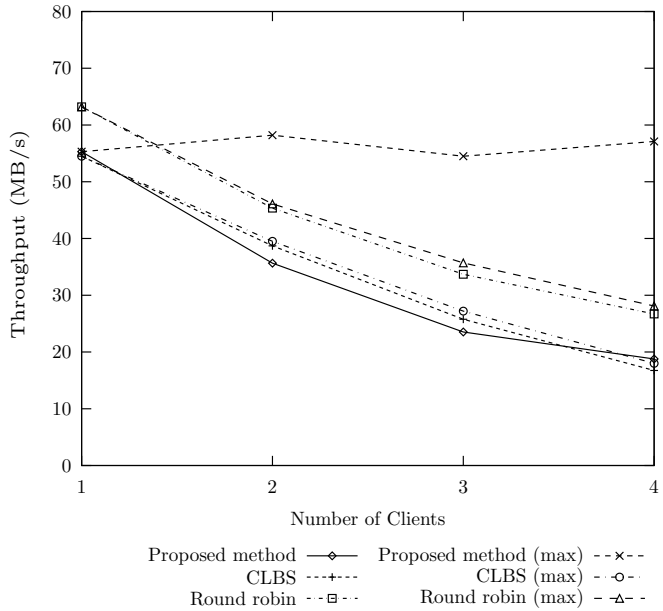

図 6 クライアント台数に対する性能の評価

め, 提案手法や CLBS より平均の性能では上回っているが, クライアント台数が増えるにしたがって性能が下がっている. それに対し, 提案手法は優先度を高く設定した一台に関しては, クライアント台数が増えてもストレージシステムを一台で占有していた時と同程度の性能を出すことができていることがグラフよりわかる.

5. 関 連 研 究

PC クラスタを用いたストレージシステムとしては, GlusterFS[1] や FAB[6], Lustre[7], Ceph[9] などがある. これらの分散ファイルシステムはそれぞれ多種多様であるが, どれもハイパフォーマンスコンピューティング向けの用途を想定しており, どのシステムもデータの配置先の決定に関してはハッシュ関数などのランダムな値やラウンドロビンによって決定される. そのため, 本研究で取り上げたようなクライアント間のディスクアクセスの衝突問題やクライアントの優先度設定を行う問題はこれらのシステムでは考慮していない. また, データサーバを等質に扱い, 各データサーバにかける CPU, ディスクなどの負荷が均等になるように設計されているため, ヘテロジニアスな環境においてはそのまま使用することはできない. 本研究の手法は各ディスクごとに目的関数に重みをつけることで, ヘテロジニアスな環境においても適用可能である.

分散ストレージにおけるスケジューリングの研究として, CLBS という手法がある. これは各ディスクにある処理待ちリクエストの量をブロードキャストすることでクライアントがアクセス先のディスクを決定するというものである. しかしこの手法では各ディスクにどのようなリクエストが処理待ちになっているかを知ることはできず, 本稿で述べたような処理待ちのリクエストパタンを考慮したスケジューリングはできない.

分散ストレージの帯域制御を行う研究として, ディスクアクセスリクエストのキューを制御することによって, 各クライアントの性能に比例的な関係をもたせる技術[3),4),8)] がある. これによって, 各クライアントの帯域制御を行うことができる. しかしこれらの手法はストレージシステムに余裕がある場合にでも優先度の低いクライアントの帯域を絞るため, ストレージ性能の使用効率が低い. 一方, 本研究の手法はアクセスするディスクを分けるため, ストレージの負荷が低いときにはその性能を十分に発揮できる.

6. お わ り に

本研究では, サーバ仮想化環境のためのクラスタストレージにおける課題として, ディスクアクセスの問題をとりあげた. ディスクアクセスの問題については, ディスクアクセスの衝突問題, クライアントの優先度を設定する問題について述べ, 評価関数を設定することで解決できることを示した. この評価関数を用いることによってディスクのアクセス状況に応じたアクセス制御を行い, 入出力の並列化による負荷分散や高性能化が向上した. 例えば複数のプロセスが同時にストレージシステムにシーケンシャルアクセスを行った場合においても, アクセスが衝突してランダムアクセスにならずに, それぞれのシーケンシャルアクセスを残すことで性能の低下を防ぐことができるようになった. また, 評価関数に重みをつけることで帯域の比例的共有によりよいアクセス割り当てを行えるようになった. そして最新の評価値のみをマルチキャストすることで現実的に実用可能なアルゴリズムを示し, 実際に動作するシステムとして実装を行った.

今後の予定としてまず実運用においても本手法が有効であるかを確認するために, より一般的な負荷パタンでの性能評価を行う. また, 本研究ではクラスタストレージに必要な耐障害性などの信頼面については触れなかったが, 本手法が高信頼なシステムを実現するための技術とどのように組み合わせていけるかを検討することも今後の課題である.

参 考 文 献

1) GlusterFS. http://www.gluster.org/docs/index.php/GlusterFS.
2) P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A.Warfield. Xen and the art of virtualization. In *Proceedings of the nineteenth ACM symposium on Operating systems principles (SOSP '03)*, pp. 164–177, 2003.
3) P.Goyal, H.M. Vin, and H.Cheng. Start-time fair queuing: A scheduling algorithm for integrated services packet switching networks. In *Proceedings of ACM SIGCOMM '96*, pp. 157–168, 1996.
4) W.Jin, J.S. Chase, and J.Kaur. Interposed proportional sharing for a storage service utility. In *Proceedings of International Conference on Measurements and Modeling of Computer Systems (SIGMETRICS '04)*, pp. 37–48, 2004.
5) D. T. Meyer, G. Aggarwal, B. Cully, G. Lefebvre, M. hael J. Feeley, N. C. Hutchinson, and A.Warfield. Parallax: virtual disks for virtual machines. In *Proceedings of EuroSys 2008*, pp. 41–54, 2008.
6) Y. Saito, S. Frølund, A. Veitch, A. Merchant, and S. Spence. Fab: building distributed enterprise disk arrays from commodity components. In *Proceedings of the 11th international conference on Architectural support for programming languages and operating systems (ASPLOS-XI)*, pp. 48–58, 2004.
7) P.Schwan. Lustre: Building a file system for 1000-node clusters. In *Proceedings of the 2003 Linux Symposium*, 2003.
8) Y.Wang and A.Merchant. Proportional-share scheduling for distributed storage systems. In *Proceedings of the 5th USENIX conference on File and Storage Technologies (FAST '07)*, pp. 47–60, 2007.
9) S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn. Ceph: a scalable, high-performance distributed file system. In *Proceedings of the 7th conference on USENIX Symposium on Operating Systems Design and Implementation*, pp. 22–22, 2006.
10) Z.Xu, Y.Zhu, R.Min, and Y.Hu. Achieving better load balance in distributed storage system. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, pp. 1314–1322. CSREA Press, 2002.