日本ソフトウェア科学会第 27 回大会 (2010 年度) 講演論文集

分散ハッシュテーブルによる省電力性を考慮したストレージシステム

丹羽 達也　長谷部 浩二　杉木 章義　加藤 和彦

近年，MAID や PDC をはじめとするストレージの省電力化の提案が数多くなされている．これらの多くはデータの管理を中央が集中して行っているが，クラウドコンピューティングに代表される大規模環境での使用を考えた場合，このような中央管理ではスケーラビリティに問題があると考えられる．そこで，本研究ではデータセンターなどでも利用可能な省電力性を考慮した分散ストレージシステムを提案する．データ管理には分散ハッシュテーブルを用いることでスケーラビリティを確保する．また，システムの省電力化は，アクセス負荷が低いときに一部の物理ノードにデータをマイグレーションし，残りのノードを休止させることで省電力化を行う．一般に負荷の変動には，時間の経過によるシステム全体のアクセス負荷の変動と，データ間でのアクセス負荷の偏りの変動があり，本研究ではこれら二つの変動を考慮した設計を行う．

1　はじめに

近年，運用コストの削減や環境保護の観点から，コンピュータシステムの省電力化に関する研究が盛んに行われている．その中でもストレージシステムが占める消費電力の割合が比較的大きいことから，MAID [1] や PDC [4] 等のストレージの省電力化の提案が数多くなされている．これらの研究は，データの管理を中央が集中して行う方式をとっているものが多い．しかしながら，クラウドコンピューティングに代表される大規模環境での使用を考えた場合，このような中央管理ではスケーラビリティに問題があると考えられる．

そこで，本研究ではデータセンターなどでも利用可能な省電力性を考慮した分散ストレージシステムの設計を行う．このようなシステムを設計する上での主な問題点として，以下の 2 つが挙げられる．

第一の問題は，データの管理方法に関するものである．一般に分散ストレージでは，どのデータ（ファイル）をどの計算機に格納するのかという問題がある．また，本研究で目的とするような大規模なストレージでは，保存されるデータ数は莫大なものとなるため，1 台の計算機がシステム全体を集中管理することは困難であると考えられる．そのため，本研究ではこれらを分散ハッシュテーブル（Distributed Hash Table, DHT）により管理する．分散ハッシュテーブルとは，ハッシュテーブルをノード間で分散管理することでスケーラビリティを確保しながら，高速な peer-to-peer 検索を可能とする技術である．

また，第二の問題は，システムの省電力化の方法に関するものである．本研究での省電力化の方法は，多くの先行研究と同様，システムのアクセス負荷の変動に注目し，アクセス負荷が低いときに一部の物理ノードにアクセスを集約することにより残りのノードを休止させるというアイディアに基づいている．このアクセス負荷の変動には，主に 1 日の人の活動に合わせたシステム全体へのアクセス負荷の変動と，新しいソフトウェアのリリースや人気のあるコンテンツの配信などによるデータ間でのアクセス数（人気度）の偏りの変動があると考えられる．そこで本研究では，

Power-Saving in DHT-based Large-Scale Storage Systems

Tatsuya Niwa　Koji Hasebe　Akiyoshi Sugiki　Kazuhiko Kato, 筑波大学大学院 システム情報工学研究科 コンピュータサイエンス専攻, Dept. of Computer Science, Graduate School of Systems and Information Engineering, University of Tsukuba.

これら二つの変動に対処しながら，常に計算機の稼働台数が最小となるようにデータの配置を最適化する方法を提案する．また，これを実現するためのアルゴリズムは，集中管理によらず各計算機によって独立に実行される．

本論文の構成は以下の通りである．第 2 章では，DHT を用いた分散ストレージシステムについて，DHT の概要，システムの構成の概要，形式的な定義の三つに分けて説明する．第 3 章では，データ拡散アルゴリズム，データ集約アルゴリズム，データ再配置アルゴリズムについて定義，アルゴリズム，例を用いて述べる．第 4 章では，シミュレーションによる評価により，アルゴリズムの効率性について評価する．第 5 章では，本研究に関連する研究について概観し，本研究の位置づけを明らかにする．第 6 章では，本研究の内容と成果を総括し，今後の課題について述べる．

2 DHT を用いた分散ストレージシステム

2.1 DHT の概要

以下で DHT の概要について述べる．ここでは，DHT の一種である Chord [5] を用いて説明する．

Chord の場合，2^n 個の ID（ハッシュ値）が図 1（ここでは $n = 7$）のように円状に割り当てられている．一方，計算機に対してもそれぞれ固有の ID が割り当てられ，ID 空間上に配置される．ここでは図中の白丸で示されるように，0, 30, 45, 72, 105 にノードが配置されている．データを保存するには，そのファイル名などをハッシュ関数に入力し，ID を得る．そして，その ID かそれより大きくて最も近い ID を持つノードに対してデータを保存する．(例えば，図 1 で示したノードの配置の場合，ID が 28 であるファイルは ID が 30 のノードに保存される.）またデータを読む際には，ハッシュ関数を用いてファイル名に対応する ID を得ることにより，保存されているノードを特定することができる．

DHT を基にした分散ストレージシステムの場合，ロードバランシングなどの目的で仮想ノード（すなわち，ソフトウェア的に実現された仮想的な計算機）がしばしば用いられる．(これに対して，物理的な計算機は物理ノードと呼ばれる.）本研究では，この仮想ノードをアクセス負荷の変動に合わせて移動させることで省電力を実現している．

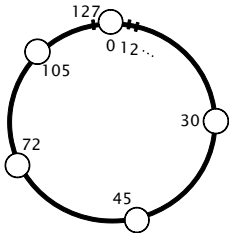

図 1 Chord

2.2 システム構成の概要

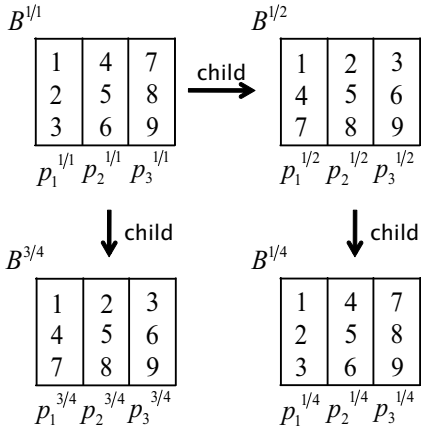

図 2 仮想ノードと物理ノードの配置

前節で説明した DHT を用いて，本研究では以下のようなストレージシステムを構成する．ここでは，図 2 で示す例を用いて説明する．この例では，議論を簡略化するために，ノードの台数をシステム全体の負荷が最小の時は 3 台の物理ノードで稼働し，最大のときは 12 台の物理ノードで稼働するとする．このとき，12 台の物理ノードを 4 つのブロックに分け，それぞれを $B^{1/1}$，$B^{1/2}$，$B^{3/4}$，$B^{4/4}$ と呼ぶ．ここで，$B^{1/2}$，$B^{3/4}$ はそれぞれ $B^{1/1}$ の子と呼ばれ，また $B^{4/4}$ は $B^{1/2}$ の子と呼ばれる．例えば $B^{1/1}$ は，$p_1^{1/1}$，$p_2^{1/1}$，$p_3^{1/1}$ の 3 つの物理ノードで構成される．これらの物理ノード上には，図で示す番号に対応して 9 つの仮想

ノードが配置される．これを $v_1^{1/1}, \ldots, v_9^{1/1}$ と呼ぶ．(したがって，例えば $p_2^{1/1}$ 上には $v_4^{1/1}$，$v_5^{1/1}$，$v_6^{1/1}$ が配置されている．)

本研究では，各物理ノードにはそれぞれキャパシティ（アクセスに対する処理能力の上限）があると仮定する．この仮定のもとで，本提案システムでは，物理ノード上の仮想ノードの負荷の合計値がキャパシティを超えない範囲で一部の物理ノードに仮想ノードを集約することで消費電力を削減する．より具体的な方法は以下の通りである．

まずはじめに，システムの負荷が低いときは $B^{1/1}$ の物理ノードのみによって仮想ノードが稼働される．次に，システムの負荷が高くなり物理ノードの負荷がキャパシティを超えたとき，保持している仮想ノード $v_i^{1/1}$ を，ID 空間が等しくなるように $v_i^{1/2}$ と $v_i^{2/2}$ の 2 つに分割し，$B^{1/2}$ の稼働中かつ低負荷な物理ノード上に $v_i^{1/2}$ をマイグレーションする．もしそのようなノードがなければ，省電力モード（あるいは休止モード）となっている一番左のノードを起動し，そこにマイグレーションする．

逆にシステムの負荷が低くなったときは，分割された仮想ノードは元の位置に戻され，分割元のノードと結合される．例えば，$p_1^{1/1}$ の負荷が低くなって $v_1^{1/2}$ を結合してもキャパシティを超えない場合は，$v_1^{2/2}$ に結合して $v_1^{1/2}$ は Chord ネットワーク上から離脱する．このようにして結合を繰り返し，もし物理ノード上に稼働中の仮想ノードが無くなれば，省電力モードに移行する．

このマイグレーションは他のブロックでも同様に行われる．例えば，$B^{1/1}$ 上の全仮想ノードが $B^{1/2}$ 上に移動すると，仮想ノード $v_1^{1/2}$ と $v_1^{2/2}$ はそれぞれ $(v_1^{1/4}, v_1^{2/4})$ と $(v_1^{3/4}, v_1^{4/4})$ に分割され，$B^{3/4}$ と $B^{1/4}$ 上にマイグレーションされる．

ここで，親子関係にある 2 つのブロック間での仮想ノードの配置について言及しておく．図 2 では，あるブロックで仮想ノードが縦に整列されている場合，その子ブロックでは横に整列されている．このように，縦横に直交するように配置することで，親物理ノードは仮想ノードの移動先として子ブロックの全物理ノードを対象とすることができる．これにより，子ブロックの物理ノードは，左から順に効率的に稼働することができる．このことは，第 4 章のシミュレーション結果で示される．

2.3 システムの形式的な定義

システムの形式的な定義は以下の通りである．なお，c は 1 物理ノードあたりの仮想ノード数であり，d はシステムの負荷の増加率である．

定義 1 システムは以下の要素からなる．

- 物理ノード集合*:* $\mathcal{P} = \cup_{j=1}^{d} P^{j/d}$ ただし $P^{j/d} = \{p_1^{j/d}, \ldots, p_c^{j/d}\}$ とする*;*
- ルート仮想ノード集合*:* $V^{1/1} = \{v_1^{1/1}, \ldots, v_{c \cdot c}^{1/1}\}$*;*
- ブロック集合*:* $\mathcal{B} = \{B^{j/d} \mid j = 1, \ldots, d\}$ ただし，各 $j = 1, \ldots, d$ について $B^{j/d} = P^{j/d}$ とする．ブロック $B^{1/1}$ は ルートブロックと呼ばれる． □

ブロック $B^{2j-1/2k}$ は $B^{j/k}$ の子ブロックと呼ばれ，$B^{j/k}$ は $B^{2j-1/2k}$ の親ブロックと呼ばれる．物理ノード $p \in B$ に対し，子ブロック B に存在する物理ノードは p の子物理ノードと呼ばれる．

補足として，深さ δ の概念を定義する：ルートブロックの深さは 1 である；深さ i のブロックの子ブロックは，すべて深さ $i+1$ である．

次に，仮想ノードの分割を以下のように定義する．

定義 2 (仮想ノードの分割) 仮想ノードの分割は次の関数 $split : V \to V \times V$ により定義される．ただし，

- $split(v_i^{j/k}) = \langle v_i^{2j-1/2k}, v_i^{2j/2k} \rangle$ とする．

この関数の入力に対する出力を，子仮想ノードと呼ぶ．2 つの仮想ノードの結合は，逆の手順により行われる． □

定義 3 (負荷とキャパシティ) 各仮想ノードおよび物理ノードの負荷とキャパシティは以下のように定義される．

Load: 各仮想ノード $v \in V$ について，v の負荷は以下のように定義される．($\mathbb{Z}^*$ は正の整数を示す):

- $load : V \to \mathbb{Z}^*$.

この関数の出力は一定時間あたりの書き込み／読み込みリクエスト数を示す．この関数を拡張し，物理ノードの負荷を $load(p) = \Sigma_{v \in alloc(p)} load(v)$ と定義する．この仮想ノードの負荷は時間とともに変化するが，分割・結合しても負荷の合計値は変化しないものとする．これは以下の式によって定義される：

- $load(v) = load(v_1) + load(v_2)$
  ただし，$split(v) = \langle v_1, v_2 \rangle$ である

Capacity: 各物理ノード $p \in \mathcal{P}$ は，以下の条件で示されるような $load(p)$ の上限となるキャパシティ $cap(p)$ を持つものと仮定する：

- $load(p) \leq cap(p)$ (for $p \in \mathcal{P}$)

もし物理ノードがこの条件を満たさない場合，このノードは過負荷であると呼ぶ． □

仮想ノードと物理ノードの配置関係を関数 $alloc : P \to 2^V$ で定義する．$alloc(p) = \{v'_1, \ldots, v'_c\}$ は p が仮想ノード $v'_1, \ldots, v'_c$ を持っていることを示す．

定義 4 (配置) 仮想ノード集合の配置は以下のようになっている: 各 $p_i \in P^{j/k} (\in \mathcal{P})$ について，

- $alloc(p_i) = \{v_{(i-1)+1}, v_{(i-1)+2}, \ldots, v_{(i-1)+c}\}$
  ($\delta(B^{j/k})$ が奇数のとき);
- $alloc(p_i) = \{v_i, v_{c+i}, \ldots, v_{c\cdot(c-1)+i}\}$
  ($\delta(B^{j/k})$ が偶数のとき).

□

3 省電力化アルゴリズム

本手法は，2 つのアルゴリズムによって構成される．この章では，まず，日々の負荷変動に対応するためのデータ拡散・集約アルゴリズムについて説明する，このアルゴリズムは 2 つから構成されており，一方は負荷が増加するときのもので，もう一方は負荷が減少するときのものである．次に，ブロック内での負荷分散を行うための補助的なアルゴリズムであるデータ再配置アルゴリズムについて，その必要性と詳細について述べる．

3.1 データ拡散アルゴリズム

各物理ノード $p_i^{j/k}$ は一定の間隔（30 分，あるいは 1 時間ごとなど）で以下の手順を実行しており，システムの負荷が日中のアクセス変化により増加するとき，以下で定義されるデータ拡散アルゴリズムによって負荷が調整される．

定義 5 (データ拡散アルゴリズム)

1. それぞれの稼働物理ノード $p_i^{j/k}$ は自身の負荷を監視しており，もし過負荷になったら，子ブロックの物理ノードすべて (すなわち，各 $i = 1, \ldots, c$ について $p_i^{2j-1/2k} \in B^{2j-1/2k}$) に対してキャパシティおよび現在の負荷を問い合わせる．
2. 子ブロックの稼働中の物理ノードで，$p_i^{j/k}$ 上の仮想ノードを分割して片方を移動しても過負荷にならないノードが存在すれば，$p_i^{j/k}$ はその中の一つを仮想ノードのマイグレーション先として選ぶ．(仮想ノードの移動先は，すでに述べたようにあらかじめ決められている．) そうでなければ，$p_i^{j/k}$ は子ブロックの中の一番左の省電力モードとなっている物理ノードを稼働させ，そこに仮想ノードをマイグレーションする．

□

より正確には，この手続きは以下のアルゴリズム 1 によって示される．ここで，B_{cld} は p の子ブロックであり，“*queue*” はマイグレーションリクエストを保持しておくためのキューである．

ここで，前節の図 2 の例を使って説明する．いま $B^{1/1}$ の物理ノードのみが稼働しており，システム全体の負荷が上昇した状況について考える．例えば，p_1 が過負荷となったときは，まずはじめに $B^{1/2}$ の全物理ノードに対して負荷を問い合わせるが，すべて省電力モードであるため，$p_1^{1/2}$ を稼働させる．次に，v_1 を 2 つに分割して $v_1^{1/2}$ と $v_1^{2/2}$ としたうちの $v_1^{1/2}$ を $p_1^{1/2}$ 上に移動する．同様にして，p_2 と p_3 が過負荷となったときも，v_4 と v_7 を分割し，$p_1^{1/2}$ 上に移動する．結果として，$p_2^{1/2}$ と $p_2^{1/2}$ は省電力モードのままでいることが可能である．

Algorithm 1 データ拡散アルゴリズム

```
if cap(p) ≤ load(p) then
  for all p' ∈ (On(B_cld), Off(B_cld)) do
    if state(p') = 0 then
      activate p'
    end if
    if cap(p') − (load(p') + Σ_{v''∈queue(p')} load(v'')) ≥ load(child(v')) then
      queue(p') ← append(queue(p'), v')
    end if
  end for
end if
```

3.2 データ集約アルゴリズム

負荷減少時において求められることは，仮想ノードを一部の物理ノードに集約し，より多くの物理ノードを省電力モードにすることである．このとき，単にデータ拡散アルゴリズムの逆の手順を行っても，効率良く仮想ノードを集約することはできない．これを図 3 を例に説明する．ここでは親ブロック内の 6 つの仮想ノードが分割され，子ブロックに移動しているとする．また，全ての仮想ノードの負荷が 1 であり，親ブロックの各物理ノード p_1，p_2，p_3 の空きリソース（キャパシティと現在の負荷の差）がそれぞれ 1，1，2 であるとする．このとき，p_6 上の仮想ノード $v_3^{1/2}$ と $v_6^{1/2}$ を親ブロックに移動してしまうと，p_1，p_2，p_3 の空きリソースはそれぞれ 0，0，2 となってしまうので 1 台しか省電力モードにできない．しかし，p_4 と p_5 の仮想ノードを移動すると，p_1 から p_3 の空きリソースが全て 0 となって，2 台の物理ノードを省電力モードにすることができる．

以上で述べられた最適化問題は，より形式的には以下の問題として表現することができる．

問題 1 : 与えられた 2 つの親子関係のブロック B_{prt} と B_{cld} について，$B^{1/2}$ の物理ノードの部分集合である S に含まれる全仮想ノードを，B_{prt} の仮想ノードに結合してもすべての物理ノードが過負荷にならないような集合のうち，S の要素数が最大となるものを見つける．

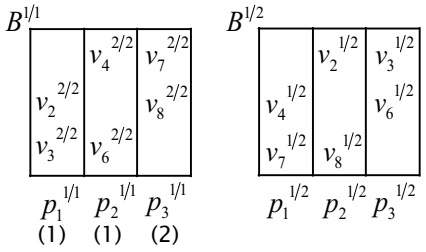

図 3　仮想ノード集約の例

この問題を解く方法として，すべての負荷情報を一部のノードに集約した上で計算することも考えられるが，スケーラビリティを確保するためには各物理ノードが協調して動作する必要がある．我々が提案するデータ集約アルゴリズムでは，各ノードの負荷情報を一定の間隔で集め，マイグレーション可能な組み合わせのうち最適解を見つけることが可能である．このアルゴリズムの定義は以下の通りである．

定義 6 (データ集約アルゴリズム) データ集約アルゴリズムは以下の 4 ステップからなる．ここで，B_{prt} と B_{cld} は親子関係のブロックである．

1. それぞれの $p'_i \in B_{cld}$ は，すべての $v' \in p'_i$ の負荷情報 $load(v')$ を，B_{prt} の全物理ノードに対して送信する．
2. 各 $p_i \in B_{prt}$ は，過負荷になることなく受け取り可能な仮想ノードを持つ B_{cld} の物理ノードの部分集合の組み合わせをすべて計算する．
3. 前のステップにより得られたすべての結果を B_{prt} の 1 台の物理ノードに集め，アルゴリズム 2 により解を見つける．ここで，$\mathcal{M}_i$ は p_i によって計算された解である．
4. 解は B_{cld} の全物理ノードに送信され, 解に従って一部の仮想ノードは B_{prt} に集約される．　□

このアルゴリズムにより，以下の命題が成立する．

命題 1 データ集約アルゴリズムによって導き出される答えは問題 1 の解である．

より理解しやすくするために, 図 3 を例にとって手順を説明する．なお，ここでは可読性のため $B^{1/1}$，$B^{1/2}$ はそれぞれ B_{prt}，B_{cld} と表記し，上つきの “1/1” は

Algorithm 2 データ集約アルゴリズム

```
function findSolution(M_1, M_2, ..., M_c)
N ← {M_1}, T ← φ
for all M_i ∈ {M_2, ..., M_c} do
  for all M ∈ M_i do
    for all M' ∈ N do
      M'' ← M ∩ M'
      if not is_include(T, M'') then
        T ← append(T, M'')
      end if
    end for
    N ← T, T ← φ
  end for
end for
return N のうち，要素数が最大となるもの

function is_include(T, M)
for all S ∈ T do
  if M ⊆ S then
    return true
  end if
end for
return false
end function
```

省略，“1/2” は “ ´ ”（ダッシュ）を用いて表記する．

手順 1: ブロック B_{cld} の全物理ノードは，自身が持つ仮想ノードの負荷情報を B_{prt} の全物理ノードに対して送信する．例えば，p'_1 は v'_4, v'_7 の現在の負荷が 1 であることを p_1, p_2, p_3 に送信する．ブロック B_{cld} のほかの物理ノード，p'_2, p'_3 に関しても同様に負荷情報を送信する．

手順 2: p_1, p_2, p_3 が受け取った情報から，例えば p_2 は p'_1 の v'_4，p'_3 の v'_6 のどちらか一方を受け取り可能だと判断できる．補足として，仮に B_{prt} の p_2 以外の全物理ノードがすべての分割された仮想ノードを受け取り可能で，p_2 が v'_4 を受け取るとしたら，集合 $\{p'_1, p'_2\}$ の物理ノードは省電力モードになることができる．(逆に，この仮定の下では p_1, p_2 はこれ以上の仮想ノードを受け取れず，v'_3, v'_6 は移動できないので p'_3 は省電力モードになれない．）言い換えれば，この集合は p_2 が考えられうるもっとも省電力にできる物理ノードの組み合わせでもある．同様にして，p_2 が $v'6$ を受け取った場合の集合は $\{p'_2, p'_3\}$ である．次のステップで使われるアルゴリズム 2 では，p_2 が算出した結果を集合 $\mathcal{M}_2$ と記述する．

このステップで算出された集合は以下の通りとなる．

- $\mathcal{M}_1 = \{\{p'_1, p'_2\}, \{p'_1, p'_3\}\}$,
- $\mathcal{M}_2 = \{\{p'_1, p'_2\}, \{p'_2, p'_3\}\}$,
- $\mathcal{M}_3 = \{\{p'_1, p'_2, p'_3\}\}$.

手順 3: p_1, p_2, p_3 によって導き出された全結果 $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3$ は，B_{prt} の 1 台の物理ノード (ここでは p_1 とする) に集められる．p_1 はアルゴリズム 2 を用いて休止可能な物理ノード数が最大となる解を求める．このアルゴリズムは，直感的には各 $\mathcal{M}_i (i = 1, 2, 3)$ から要素を一つ選び出し，共通要素（M''）を計算している．

最終的に導き出された B_{cld} の物理ノードの部分集合は，B_{prt} がそれらの仮想ノードを結合することで省電力モードになることができるノードの集合である．例として，$\mathcal{M}_i$ の最初の要素が選択された場合を考えてみる．

- $S_1 = \{p'_1, p'_2\}$,
- $S_2 = \{p'_1, p'_2\}$,
- $S_3 = \{p'_1, p'_2, p'_3\}$.

このとき，$S_1 \cap S_2$ によって求められた物理ノード集合は，もし p_1 と p_2 がそれらの仮想ノードを可能な限り受け取り，B_{prt} の他の物理ノードも可能な限り受け取った場合に省電力モードとなることができることを示す．これ以降についても同様で，最終的に $\bigcap_{i=1}^{3} S_i$ によって求められた解は，それらが持つ仮想ノードをすべて結合したら省電力モードになることができる物理ノードの集合である．このアルゴリズムはこのような共通要素の組み合わせを可能な限り行い，要素数が最も大きくなる解を選ぶ．補足として，計算途中で得られる M'' にすでに同じ解があった場合は計算を省略することができる．これにより計算量

を押さえることが可能である．

手順 4: アルゴリズム 2 によって求められた解は，B_{cld} の全物理ノードに通知される．そして，それに該当する物理ノードは全仮想ノードを親物理ノードに対して移動する．

最後に，ストレージが十分大きくて冗長性を持たせられる場合に，データの移動コストを抑える方法について説明する．すでに述べたように，我々の手法では仮想ノードの配置はあらかじめ決められている．このため，集約アルゴリズムによって分割された仮想ノードが元の場所へ移動する際に，移動元に仮想ノードのデータを残しておくことで，次回以降，仮想ノードを移動するには前回からの差分更新のみで済む．

3.3 データ再配置アルゴリズム

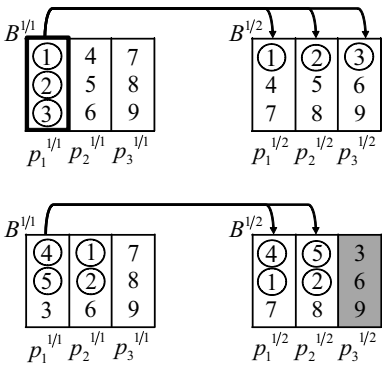

図 4 仮想ノード集約の例

前節では，我々のアルゴリズムがどのように負荷の変動に応じて仮想ノードを移動し，省電力化するかについて述べた．しかし，効率的に省電力化し続けるにはブロック内でロードバランシングを行う必要がある．これについて図 4 の簡単な例を用いて説明する．図中の上のブロック $B^{1/1}$ は $B^{1/2}$ の親ブロックであり，3 つの物理ノードのキャパシティはすべて 6 であるとする．ここで，データの人気度の変化により $v_i^{1/1} \in p_1^{1/1}$ (すべての $i = 1, 2, 3$ について) の負荷が 3 になり，他の仮想ノードは 1 であるとする．よって，$p_1^{1/1}$ は $B^{1/1}$ の他の物理ノードに比べ，非常に負荷が高い状態となる．このような場合，この物理ノードは他に比べ，早い段階で仮想ノードを移動する必要があり，その結果，$B^{1/2}$ の全物理ノードは低負荷状態で稼働しなければならない．

この問題を解決するために，本研究では，比較的長期的な期間で（例えば，1 週間）ブロック内のロードバランシングを行うためのデータ再配置アルゴリズムを提案する．例えば，図 4 の場合，$p^{1/1}$ は不均一であることを検知し，いくつかの仮想ノード（この場合は $v_1^{1/1}$ と $v_2^{1/1}$）を同じブロック内の負荷の低い物理ノード上の同一行の仮想ノード（$v_4^{1/1}$ と $v_5^{1/1}$）と入れ替える．このとき，他のブロックについても同様に仮想ノードを入れ替える．(なお，異なる物理ノード間で交換する場合はデータの移動が発生するが，同一物理ノード上で位置が入れ替わる場合は発生しない．) この場合では，$p_3^{1/2}$ が稼働するのを遅延させれるため，$B^{1/2}$ の稼働ノード数を減らすことができる．

しかし，より現実的な状況を考えたとき，1 つのブロックのみで最適化すると他のブロックの結果と衝突する可能性がある．この問題を解決するために，すべての負荷情報を 1 つの物理ノード上に集め，全体で均衡がとれるように最適化する必要がある．本論文ではこの問題までは取り扱っておらず，今後の課題の一つである．なお，第 4 章では衝突が一切無い状況を想定してシミュレーションを行っている．

4 シミュレーションによる評価

本研究では，システムへのアクセス頻度が変化する状況におけるアルゴリズムの効率性をシミュレーションによって評価した．具体的には，稼働物理ノードの平均負荷を評価した．この平均負荷が高いほど，アクセス負荷を少数のノードに集約できているといえる．また，上記の結果に加え，同じ設定の基でのシステムの稼働ノード数に関してもシミュレーションを行い，データ再配置アルゴリズムの実行の有無を比較することで，このアルゴリズムの有効性を示す．

以下にシミュレーションでの想定環境を示す．

- 物理ノード数：160 台
- 最大負荷増加倍率：4 倍

- 各物理ノードにおける仮想ノード数：20 台
- 全仮想ノード数：400
- 各物理ノードのキャパシティ：100

ここで，2 つのグループの仮想ノードを考える．1 つ目のグループは 100 個の仮想ノードからなり，初期負荷は 5 である．2 つ目のグループは 300 個の仮想ノードで，初期負荷は $5 \cdot \alpha$ である．また，時刻 $t = 0, 1, \ldots, 23$ を考え，$t = 0$ を初期時刻とし，$t = 11$ まで負荷が初期負荷の 0.34 倍ずつ増加し，その後 $t = 23$ まで初期負荷の 0.34 倍ずつ減少するものとする．

$\alpha = 1.2, 1.5, 2.0$ としてシミュレーションを行ったときの，稼働物理ノードの平均負荷を図 5 に，稼働物理ノード台数を図 6 に示す．データ再配置アルゴリズムを適用 (図中の凡例では with long-term が該当) すると，平均負荷が上昇し，稼働台数が減少していることから省電力性が向上しているのが分かる．また，稼働物理ノードの平均負荷は常に 80 – 98% であり，2 つの異なる負荷値を持った仮想ノードがある環境においても本アルゴリズムが効果的に動作することが確認できた．

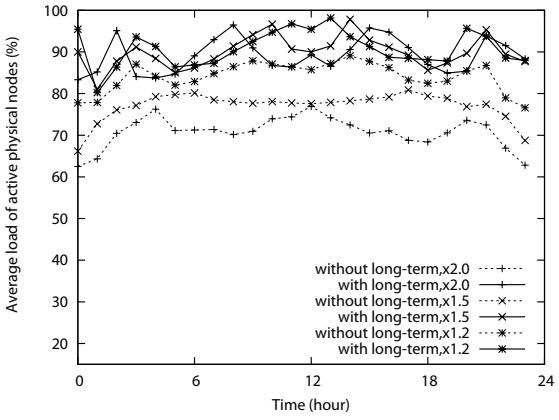

図 5　稼働物理ノードの平均負荷

5　関連研究

これまでに提案されてきたストレージの省電力化に関する研究に共通するアプローチは，データの配置

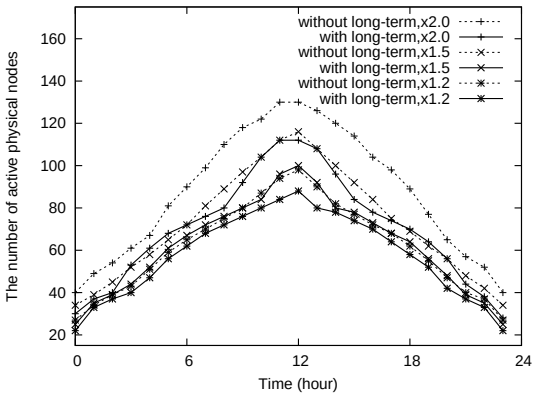

図 6　稼働物理ノード数

を最適化することでアクセス負荷を一部の計算機に集約するというものである．これにより，ストレージ全体へのアクセスが少ない間，一部の計算機を休止させることにより省電力化を実現している．こうした研究の代表例として MAID [1] や PDC [4] などが挙げられる．これらはデータの人気度に着目しながら，一部のディスクに人気度の高いデータを集約することで電力の削減を行うという試みである．

これらの研究では，データアクセスの管理を一部の計算機が集中して行う方法に基づいている．そのため，これらの研究で対象となるストレージは比較的少数のディスクから構成されるものが多い．それに対し，Harnik ら [3] のような最近の研究では，クラウド環境などでの利用を想定した大規模分散ストレージを対象とした省電力手法が提案されている．この手法は，複製を効果的に配置することにより，すべてのデータにアクセス可能な状態を保ちつつ，稼働しているディスクの数を抑えるというものである．また，本研究で基にしている DHT によるストレージへの適用についても議論している．本研究との違いは，Harnik らが複製を用いるのに対し，本研究では効果的なアルゴリズムにより仮想ノードを移動することで，負荷の集約を実現している．

また一方で，負荷を調整するという点において本研究に関連の深いものとして，Surana らによる DHT のロードバランシングに関する研究 [2] が挙げられる．この研究では，仮想ノードを移動し，物理ノード間で

負荷を均一にすることでロードバランシングを実現している．一方，本研究では，仮想ノードの移動により負荷を偏らせることで，省電力を行っている．

6 まとめと今後の課題

本論文では，分散ストレージシステムを省電力化するための手法を提案した．本手法では，データの管理を DHT によって行い，また，ロードの変化パターンに着目して 2 種類のアルゴリズムを用いることで消費電力を最適化する方法を提案した．その上で，シミュレーションにより効果的に省電力化できることを示した．

今後の課題として，杉木らによって研究されているクラウド基盤ソフトウェア Kumoi [6] 上で現在実装途中である本システムを完成させ，実機による実験評価を行うことを目指している．

謝辞 本研究は，総務省戦略的情報通信研究開発推進制度 (SCOPE)ICT イノベーション促進型研究開発の支援を受けて行われた．

参 考 文 献

[1] Colarelli, D. and Grunwald, D.: Massive arrays of idle disks for storage archives, In *Proc. ACM/IEEE Conf. on Supercomputing*, 2002, pp.1-11.

[2] Surana, S., Godfrey, B., Lakshminarayanan, K., Karp, R. and Stoica, I.: Load balancing in dynamic structured peer-to-peer systems, *Perform. Eval.*, 2006, pp.217–240.

[3] Harnik. D., Naor, D. and Segall, I.: Low power mode in cloud storage systems. *Parallel and Distributed Processing Symposium, International*, 2009, pp.1-8.

[4] Pinheiro, E. and Bianchini, R.: Energy conservation techniques for disk array-based servers, In *Proc. Int'l Conf. on Supercomputing*, 2004, pp.68-78.

[5] Stoica, I., Morris, R., Karger, D., Kaashoek, M., F. and Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications, In *Proc. ACM SIGCOMM*, 2001, pp.149-160.

[6] 杉木章義, 加藤 和彦: Kumoi：クラウドコンピューティング研究・開発のためのシェル環境の構築, 情報処理学会第 21 回コンピュータシステムシンポジウム, 2009.

[7] 丹羽達也, 長谷部浩二, 杉木章義, 加藤和彦: Power-aware Chord：省電力性を考慮した分散ハッシュテーブル, 日本ソフトウェア科学会全国大会予稿集, 2009.