分散ハッシュテーブルを用いたストレージシステムの省電力化手法

為我井 勝　長谷部 浩二　加藤 和彦

本研究では，大規模分散ストレージシステムの省電力化手法を提案する．その基本的なアイディアは，ストレージシステムを分散ハッシュテーブルにより管理し，予測される将来のアクセス頻度が類似したファイル群を同一のディスクに集約するというものである．これにより，一部のディスクにアクセスを集約させ，一定時間アクセスされないディスクを休止状態にすることで消費電力を削減する．ここでのアクセス頻度の予測は，ファイルがアップロードされてからの経過時間とアクセス回数によって行われる．これをもとに，分散ハッシュテーブルにおける ID の割り当てを，従来のハッシュ関数に加えてアップロード後の経過時間とアクセス回数により行う．以上により，アクセス頻度によるファイルの集約を，分散ハッシュテーブル上の ID 空間で自然に実現することができる．さらに，ファイルの保存場所が変化したとしても，クライアントは目的のファイルの ID をアップロード後の経過時間とアクセス回数から求めることでアクセスすることができる．本論文では，以上で述べた提案手法の詳細を説明するとともに，ディスクの使用効率を向上させる方法についても議論する．

1　はじめに

近年，クラウドコンピューティングの発展と普及に伴い，インターネット経由のオンラインストレージサービスが普及している．その例としては，Dropbox，Google Drive，Amazon S3 等が挙げられる．こうしたサービスを提供するためには，データセンター規模でサーバを稼働させる必要があり，そこでは大量の電力が消費される．文献[12] によると，米国で消費される電力のうち，データセンターが消費する電力の割合は約 1〜2%だと報告されている．そのため，データセンターの省電力化は，クラウドコンピューティングの基盤システムを構築する上で重要な課題となっている．このようなデータセンターにおいて，電力消費の割合が比較的高いものとしてストレージシステムが挙げられる．文献[7] によると，その割合は 25-35% だとも言われている．そのため，近年ストレージシステムの省電力化の研究が盛んに行われている．

以上を背景に，本研究では，大規模な分散ストレージシステムの省電力化手法を提案する．特に，ファイルが逐次追加される状況に対応し，負荷を一部のディスクに集約することを目的とする．その基本的なアイディアは，将来のアクセス頻度が類似したファイル群を同一のディスクに保存することで負荷の集約を行い，一定時間アクセスされないディスクを休止状態にするというものである．特にここでは，ディスク間でのファイルの移動により，負荷の集約を実現する．一般に，分散ストレージシステムでは，ファイルの保存場所をインデックスサーバによって管理する．しかしながら，本研究のようにファイルの移動が頻繁に起こるシステムでは，単一のインデックスサーバによって管理を行うと，そこが性能のボトルネックや単一障害点となることが考えられる．そこで本研究では，ファイルの保存場所を分散ハッシュテーブルによって管理する．分散ハッシュテーブルとは，巨大なハッシュテーブルを用いて複数のノードでデータを管理する技術である．これにより，単一のインデックスサーバによるファイルの保存場所の管理を不要にすることが

Power-Saving in Storage Systems with Distributed Hash Table.

Masaru Tamegai, Koji Hasebe, Kazuhiko Kato, 筑波大学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.

できる．以上で述べたシステムにおいて負荷の集約を実現するために，本研究では，分散ハッシュテーブル上のファイルの ID とファイルのアクセス頻度を同一視するという手法を提案する．先行研究 [8] で，ファイルの将来のアクセス頻度は，過去のアクセス回数とアップロード後の経過時間の 2 つと相関関係のあることが示されている．本提案手法でもこの関係を利用して，分散ハッシュテーブルにおける ID をファイルのアップロード後の経過時間と過去のアクセス回数により決定する．ただし，提案システムでは，クライアントはファイルへのアクセス回数を保持しているものとし，ファイルが異なるディスクに移動したとしても，クライアントは目的のファイルの ID をアップロード後の経過時間と過去のアクセス回数から求めることでアクセスすることができる．以上のアイディアに基づき，本提案手法ではクライアントの目的のファイルへのアクセスを可能にしつつ，ファイルの将来のアクセス頻度により負荷の集約を行う．

以下，第 2 章で関連研究について説明する．第 3 章では，実際のファイル共有サービスのアクセスパターンを用いて将来のアクセス頻度予測を行うために，Dailymotion と呼ばれるファイル共有サービスのアクセスパターンを分析した結果を述べる．第 4 章では本研究が提案するシステムを説明する．最後に，第 5 章でまとめと今後の課題を述べる．

2 関連研究

これまでに提案されてきたストレージシステムの省電力化手法の多くは，低負荷時にファイルを一部のディスクに集約させることで，他のディスクを省電力モードに移行させ，消費電力を削減するというものである [3]．このような省電力化の手法には，大きく分けて冗長化したデータを用いて負荷分散するものと，アクセス頻度に着目して負荷の集約を行うものの 2 つがある．冗長化したデータを用いる例としては，DIV [10] や PARAID [14] が挙げられる．DIV では，オリジナルデータが保存されたディスクとデータのレプリカが保存された 2 種類のディスクを用意する．負荷が低いときは，レプリカが保存されたディスクを省電力モードにすることで省電力化を行う．一方で，負荷が高いときは，レプリカを配置したディスクを利用して負荷分散を行う．PARAID では，RAID により構築されたストレージシステムを対象とし，そこでの未使用のスペースに着目した手法である．この手法では，データのレプリカを未使用のスペースに配置し，そのスペースを負荷の大きさにより段階的に省電力モードにしていく．

一方，アクセス頻度に着目した手法の例としては，MAID [4] や PDC [9] が挙げられる．MAID は，直前にアクセスされたデータをキャッシュディスクに保存し，常にキャッシュディスクを起動させておき，その他のデータを保存したディスクを省電力モードにする手法である．それに対し，PDC はデータをアクセス頻度順に保存することで，アクセスの少ないディスクを省電力モードに移行する手法である．

以上で挙げた手法は，比較的小規模なシステムを前提としている．しかし近年，クラウドコンピューティングの普及に伴い，大規模なストレージシステムに対する省電力化の研究も行われるようになった．そこでの中心的課題は，以上の手法で用いられてきたアイディアの拡張性を高め，大規模なシステムに適用することである．こうした研究の例として，HDFS（Hadoop Distributed File System）を対象とした手法 [6] がある．この手法は，ディスク群を hot zone と cold zone に分類して省電力化を行うものである．また同様に，クラウド上の既存のストレージシステムを対象にした手法 [5] がある．この手法では，レプリカの配置を工夫することで省電力化を行っており，さらに，省電力モードで一部のデータにアクセスできなくなる状況を考慮し，auxiliary nodes と呼ばれる特別なノードを挿入することで，全てのデータへのアクセスを可能にしている．その他にも，本研究の直接の先行研究である Okoshi らの [8] による格子状に配置したディスクでアクセス頻度ごとにファイルを集約する手法がある．Okoshi らの手法では，ファイルに対する急激なアクセス頻度の変化に対応するために，現在のアクセス頻度ではなく，将来のアクセス頻度の予測値を用いてファイルの集約を行っている．ただし，これらの手法では，ファイルが保存されている場所の管理を単一のインデックスサーバによって行っ

ている．単一のインデックスサーバによるファイルの保存場所の管理は，そこが性能のボトルネックや単一障害点となる可能性がある．

以上で述べた先行研究に対し，本研究では [8] と同様にファイルが逐次的に増加する状況に対応し，アクセス頻度の予測を行いつつ，単一のインデックスサーバによる全ファイルの保存場所の管理が不要な手法を提案する．また，[8] では Flickr のアクセスパターンのみを用いて将来のアクセス頻度予測を行っており，他のサービスに対し将来のアクセス頻度予測が適用可能かどうかは不明である．そこで，本研究では，他のサービスに対してもアクセス頻度予測が適用可能であるか否かを調べるために Dailymotion と呼ばれるファイル共有サービスのアクセスパターンの分析を行った．

3 Dailymotion における動画ファイルの分析

本研究では，Flickr 以外のファイル共有サービスにおいても将来のアクセス頻度予測が可能か否かを調べるために，Dailymotion と呼ばれる動画共有サイトのファイルのアクセスパターンを取得した [1]．Dailymotion には月間 1.28 億人の人々が訪れ，月に約 3 億人が動画を視聴している．また，1ヶ月の動画視聴回数は 25 億回以上であり，1 日に 9 千以上もの動画が投稿されている [11]．本研究では，Dailymotion の動画ファイルのアクセスパターンの収集を，Dailymotion が提供している Web API [2] を用いて行った．この API により，アップロード直後のランダムな 50,000 個の動画ファイルのアクセス回数を間 2014 年 12 月 20 日から 2015 年 2 月 22 日まで 1 時間ごとに採取した．

Okoshi らの先行研究 [8] では，実際のファイル共有サービスのアクセスパターンを用いてファイルの将来のアクセス頻度予測を行っているが，Flickr のみの写真ファイルの分析しか行っていない．そこで，本研究では，Dailymotion における動画ファイルの分析を行うとにより，アクセス頻度予測が Flickr 以外の他のサービスに対しても適応可能であることを示す．以下，3.1 節で Dailymotion のアクセスパターンを分析した結果を説明し，この結果をもとに将来のアクセス頻度を求める方法を 3.2 節で説明する．

3.1 Dailymotion のアクセス傾向の特徴

まず，動画ファイルの 1 時間あたりの平均のアクセス回数の推移を調べた．図 1 にその結果を示す．

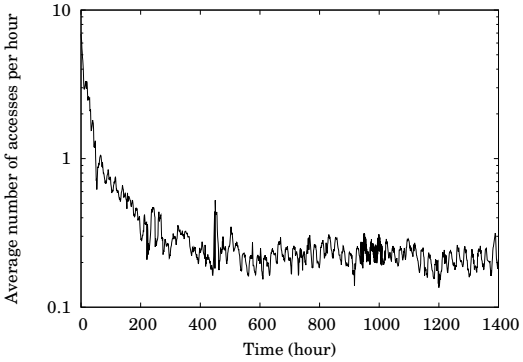

図 1　Dailymotion における平均のアクセス回数の推移

1 時間あたりの平均のアクセス回数はアップロードされて 1 時間後は 6.8 回であるのに対し，1400 時間後は 0.19 回まで減少している．また，1400 時間中，400 時間までに全アクセス数の約 55%を占めていることが分かった．これらの結果より，先行研究 [8] の Flickr のアクセスパターンと同様に，経過時間とアクセス回数は負の相関関係が存在し，特にアクセスはアップロード直後に多く行われることが分かった．

次に，図 2 にアクセス頻度の偏りを表す結果を示す．この図では各経過時間に対する累積のアクセス回数を動画ファイルのアクセス回数ごとに並び替えた結果を表している．この図 2 より，アクセスは一部のファイルに集中していることが観察できた．また，図 1 の結果と同様に，アップロード直後にアクセスの回数が多いことも観察できた．

3.2 Dailymotion におけるアクセスパターンを用いた将来のアクセス傾向予測

3.1 節の結果より，Dailymotion のアクセスパターンにおいても先行研究 [8] の Flickr のアクセスパターンと同様に，将来のアクセス頻度は，アップロードされてからの経過時間が短く，過去のアクセス回数が多

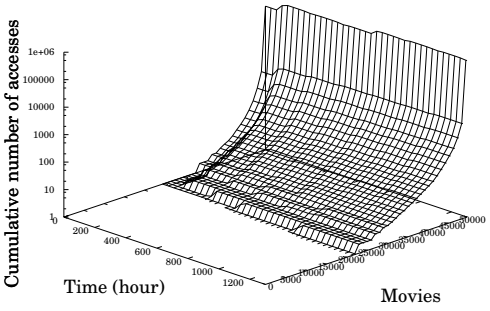

図 2　Dailymotion におけるアクセス頻度の偏り

いファイルは高いことが考えられる．そのため，将来のアクセス頻度は，過去のアクセス回数とアップロードされてからの経過時間の 2 つと相関関係のあることが考えられる．そこで，図 3 に Dailymotion における動画の経過時間とそれまでの累積のアクセス回数に対する 300 時間後のアクセス回数を表した図を示す．ここでは，x 軸に経過時間を示し，y 軸にそれまでの累積のアクセス回数を示し，z 軸に経過時間から 300 時間後までにアクセスされる回数を示している．しかし，このグラフからはデータを採取していない部分がグラフの谷として現れてしまう． そのた

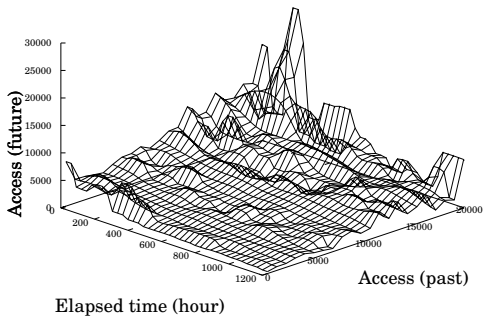

図 3　Dailymotion における将来のアクセス回数の算出

め，次に先行研究 [8] と同様に，図 3 のグラフに対し，経過時間ごとに最小二乗法と累乗関数 $z = ay^b$ により，a と b を決定し累乗近似を行い，データを採取していない部分をアクセス頻度予測に用いることができるようにしたグラフを図 4 に示す． この図を用い

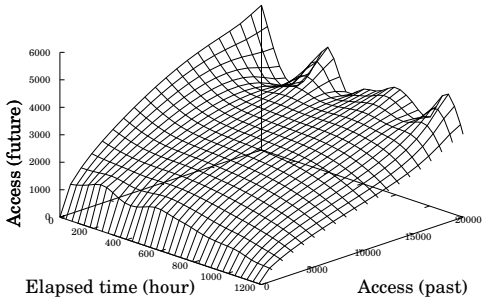

図 4　Dailymotion において累乗近似により平滑化した将来のアクセス回数

て，アップロードされてからの経過時間と累積のアクセス回数の 2 変数を引数にとり，将来のアクセス頻度を関数 $freq : F \times T \to \mathbb{N}$ として算出する．以上の分析により，先行研究 [8] で示されている将来のアクセス頻度に関する相関関係が Dailymotion においても観察することができた．そのため同様のファイル共有サービスの多くで，将来のアクセス頻度についての相関関係の見られることが予想される．したがって，本研究で採取したようなファイルの経過時間と過去のアクセス回数，将来のアクセス回数のデータを用いて，将来のアクセス頻度予測を行い，提案システムを適用することが有効だと思われる．

4　提案システム

この章では，本研究が提案するシステムについて説明する．4.1 節で提案システムの概要を説明し，4.2 節でシステムの構成を説明する．また，4.3 節では，提案システムにおける分散ハッシュテーブルの ID の定義を説明し，4.4 節では，クライアントの要求に対しシステムが行う処理を説明する．さらに，4.5 節では，提案システムが ID を更新する方法について説明し，4.6 節では，ディスクの使用率を改善するための処理の説明をする．

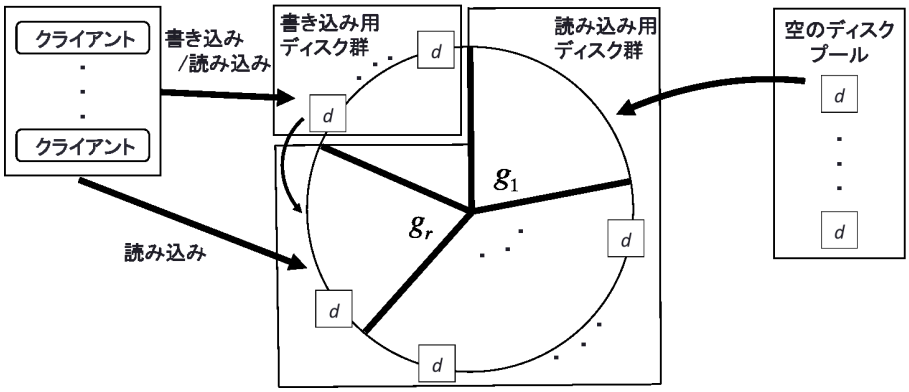

図 5　システムの構成

4.1　システムの概要

提案手法は，数千から数万台のディスクにより構成された大規模なストレージシステムを対象としている．図 5 にシステム全体の構成図を示す．提案システムは，クライアントからファイルのアップロードを受け付ける k（定数）台の書き込み用ディスク群と，クライアントから読み込みのみ受け付ける読み込み用ディスク群，さらに空のディスクプールの 3 つのグループにより構成されている．なお，本研究の分散ストレージシステムでは，分散ハッシュテーブルを用いてファイルを管理する．分散ハッシュテーブルとは，ネットワーク上の複数のノードでファイルを分散して管理する技術である．これを用いることにより，クライアントからのファイルへのアクセスを単一のインデックスサーバ無しで可能にする．本研究の提案手法では，分散ハッシュテーブルの代表的な 1 つである Chord [13] を用いる．ただし，本提案手法では，分散ハッシュテーブルにおけるファイルの ID をアクセス頻度に従い変化させる．本研究では先行研究 [8] で示された将来のアクセス頻度の性質を利用して，Chord における ID を下位の部分は従来のハッシュ関数により一意の値を割り振り，上位の部分は過去のアクセス回数とアップロード後の経過時間により変化させる．これにより，ID 空間上でアクセス頻度によるファイルの集約を行うことができる．なお，クライアントがファイルにアクセスする場合は，ファイルのハッシュ値と過去のアクセス回数，アップロード後の経過時間により ID を求めることで常にアクセスできる．また，ファイルをディスク間で移動させると使用率が低いディスクが生じてしまう可能性がある．そこで，本提案手法では，一定時間ごとに使用率が低いディスクのファイルを一部のディスクにまとめることでシステム全体としてディスクの使用率を高くする処理を行う．

4.2　システムの構成

提案システムにおいて，初期状態でシステムを構成しているディスクの集合を $D_{init} = \{d_1, \cdots, d_k\}$ とし，これらのディスクを初期状態の書き込み用ディスク群とする．さらに，各ディスクを Chord におけるノードとし，Successor を関数 $suc : D \times T \to D$ と定義し，Predecessor は $pre : D \times T \to D$ と定義する．また，システムの稼働中に追加される空のディスクの集合を $D_{sup} = \{d_{k+1}, \cdots\}$ とする．すなわち，全てのディスクの集合は $D = D_{init} \cup D_{sup}$ と表される．さらに，このシステムに格納されているファイルの集合を F として，$F = \{f_1, f_2, \cdots\}$ とする．システムにおける時刻は $t = 0, 1, 2, \cdots \in T$ として自然数で表し，離散的に表すものとする．次に，ファイルが属するディスクを関数 $place : F \times T \to D$ によって表す．つまり，時刻 t において，あるファイル f_i がディスク d_j に属するとき，$place(f_i, t) = d_j$ とする．また，関数 $place$ はあるディスクに属するファイル群を表すためにも用いる．すなわち，$place : D \times T \to 2^F$ でもあり，$place(d_j, t) = \{f \mid place(f, t) = d_j\}$ となる．ただし，文脈より明らかな場合は $place(f_i, t) = d_j$ を単に $f_i \in d_j$ と表す．さらに，1 つのファイルのサイ

ズを関数 $size : F \to \mathbb{N}$ で表し，ディスクに格納されているファイルの容量の合計を関数 $vol : D \times T \to \mathbb{N}$ で表す．したがって，以下のように表される．

$$vol(d_j, t) = \sum_{f \in place(d_j, t)} size(f).$$

これに対し，ディスクのファイルの許容量を関数 $cap_{vol} : D \to \mathbb{N}$ で表す．

4.3 分散ハッシュテーブルにおける ID の定義

(1) ディスクの ID の定義

分散ハッシュテーブルによって管理されるディスクの ID を関数 $ID^D : D \times T \to IDS$ により表し，ID 空間は集合 IDS で表されるものとする．関数 ID^D は以下のように定義する．

$$ID^D(d, t) = ID^D_{time}(d, t) : ID^D_{unique}(d, t).$$

ここでは，x と y が整数で y の桁数が Y のとき，$x : y$ は $x \cdot 10Y + y$ を表すものとする．ID は $M = M_{time} + M_{unique}$ ビットで表され，ID のうち上位 M_{time} ビットがディスクの場合は関数 $ID^D_{time} : D \times T \to \mathbb{N}$ により表される．それに対し，下位 M_{unique} ビットがディスクの場合は関数 $ID^D_{unique} : D \times T \to \mathbb{N}$ により表される．なお，ID 空間のうち $2^{M_{time}} - 1 : 0$ から $2^{M_{time}} - 1 : 2^{M_{unique}} - 1$ までが書き込む用ディスク群が管理される範囲であり，$0 : 0$ から $2^{M_{time}} - 2 : 2^{M_{unique}} - 1$ までが読み込み用ディスク群が管理される範囲とする．次に，関数 ID^D を構成する関数 ID^D_{time} と ID^D_{unique} についてそれぞれ説明する．まず，関数 ID^D_{time} を次の式で定義する．

$$ID^D_{time}(d, t) = \begin{cases} 2^{M_{time}} - 1 \\ \quad (if\, t < time_{add}(d)) \\ (2^{M_{time}} - 1) - (r_i - time_{add}(d)) \\ \quad (if\, t \geq time_{add}(d)). \end{cases}$$

提案システムでは，読み込み用ディスク群におけるディスクとファイルの ID は一定時間 c（定数）ごとに変化し，$r_1 < \cdots < r_i \leq t < r_{i+1}$ とし，r_i のときに変化する．すなわち，$r_i + c = r_{i+1}$ となる．以下の本論文では，時刻 t において直前に更新された時刻を r_i と表すものとする．関数 ID^D_{time} では，ディスクが書き込み用ディスク群にある場合は常に $2^{M_{time}} - 1$ であり，ディスクが読み込み用ディスク群に移動したとき $(2^{M_{time}} - 1) - (r_i - time_{add}(d))$ に変化する．この式における関数 $time_{add} : D \to T$ は，ディスクが書き込み用ディスク群から読み込み用ディスク群に挿入された時刻を表している．そのため，$t < time_{add}(d)$ のときは，時刻 t においてディスク d が書き込み用ディスク群に存在することを表し，$t \geq time_{add}(d)$ のときは読み込み用ディスク群に存在することを表す．

次に，関数 ID^D_{unique} について説明する．関数 ID^D_{unique} を次の式で定義する．

$$ID^D_{unique}(d, t) = \begin{cases} w\_index(d, t) \cdot \frac{2^{M_{unique}} - 1}{k} \\ \quad (if\, t < time_{add}(d)) \\ hash^D(d) \\ \quad (if\, t \geq time_{add}(d)). \end{cases}$$

関数 $w\_index : D \times T \to \mathbb{N}$ は時刻 t において，ディスク d の ID^D_{unique} の書き込み用ディスク群における順序を表す関数であり，$0 < w\_index(d, t) \leq k$ となる．関数 $hash^D : D \to \mathbb{N}$ はディスクに対し，ディスクの IP アドレス等から一意に決まる値をハッシュ関数により割り当てることを表している．

(2) ファイルの ID の定義

次に，分散ハッシュテーブルによって管理されるファイルの ID を表す関数 $ID^F : F \times T \to IDS$ について説明する．関数 ID^F を以下の式で定義する．

$$ID^F(f, t) = ID^F_{time}(f, t) : hash^F(f).$$

まず，ID のうち上位 M_{time} ビットを表す関数 $ID^F_{time} : F \times T \to \mathbb{N}$ について説明する．ファイルが保存されているディスクが書き込み用ディスク群に存在するとき，ID^F_{time} は $2^{M_{time}} - 1$ となる．それに対し読み込み用ディスク群に存在するときは以下の式となる．

$$ID^F_{time}(f,t) = \begin{cases} ID_{freq}(f, t_{r_{i-1}}) - c \\ \quad (if\, group(ID_{freq}(f, t_{r_{i-1}}) \\ \qquad = group(ID_{freq}(f, t_{r_i})) \\ ID_{freq}(f, t_{r_i}) \\ \quad (otherwise). \end{cases}$$

本研究の提案手法では，読み込み用ディスク群において，ファイルの ID が変化したとき，アクセス頻度が同様なディスク間でのファイルの移動を無くすために，ID 空間を n 段階のグループに分割し，同一グループ内でファイルが他のディスクへ移動しないようにファイルの ID を定義した．そのため，n 段階のグループの集合を $G = \{g_1, \ldots g_n\}$ とし，それぞれのグループにおける ID の範囲の上限を関数 $area : G \to 2^{IDS}$ で表し，以下のように定義する．

$$area(g_i) = \{x | (i-1) \cdot \frac{2^{M_{time}}-2}{n} < x \leq i \cdot \frac{2^{M_{time}}-2}{n}\}.$$

それに対し，$ID^F_{time}(f,t)$ で用いられる関数 $group : IDS \to G$ は，関数 $area$ を使って定義する部分関数であり，ID を引数にとりその ID が所属するグループを返すものとする．さらに，関数 $ID_{freq} : F \times T \to \mathbb{N}$ は読み込み用ディスク群に存在するディスクに保存されているファイルの ID^F_{time} を求めるときや更新するときに用いる関数であり，関数 ID_{freq} の式は以下の通りである．

$$ID_{freq}(f,t) = (2^{M_{time}} - 1) - (t - time_{up}(f)) + a \cdot freq(f,t).$$

a は $0 < a$ の定数であり，関数 $time_{up} : F \to T$ でファイルがディスクにクライアントからアップロードされた時刻を表している．また，関数 $freq : F \times T \to \mathbb{N}$ は時刻 t におけるファイル f のアクセス頻度を表す関数である．関数 $freq$ では，先行研究 [8] における Flickr のアクセスパターンを用いた将来のアクセス頻度予測や，本研究で分析した Dailymotion のアクセスパターンを用いた将来のアクセス頻度予測を用いる．最後にファイルの関数 $hash^F$ について説明する．ファイルの ID の下位 M_{unique} ビットは関数 $hash^F : F \to \mathbb{N}$ により，ファイルの属性情報等から一意に決まる値を割り当てるものとする．

4.4 クライアントの要求とシステムの処理

この節では，提案システムにおけるクライアントによるファイルのアップロード方法とクライアントがファイルへのアクセス方法を説明する．

(1) クライアントがファイルをアップロードする方法

まず，提案システムにおけるクライアントのファイルのアップロード方法を説明する．まず，クライアントは，ファイルに関数 $hash^F$ により ID の下位 M_{unique} ビットを割り当てる．それに対し，上位 M_{time} ビットの値は $2^{M_{time}} - 1$ とする．以上の ID に従い対応する書き込み用ディスク群のディスクにアップロードする．

(2) クライアントがファイルにアクセスする方法

次に，提案システムにおいて，クライアントが目的のファイルにアクセスする方法を説明する．クライアントが時刻 t において，目的のファイル f にアクセスするものとし，f をアップロードした際に $hash^F(f)$ の値を保持しており，f に対してアクセスするのは 1 人のクライアントのみであるため，f のアクセス回数とアップロードしてからの経過時間の情報も保持している．なお，f に対し複数のクライアントからアクセスする場合は，ID を問い合わせるノードをシステムの中に設置し，ここにファイルのアクセス回数を記録し，クライアントはここからアクセス回数とアップロード後の経過時間を求める．ただし，クライアントにとって，f が書き込み用ディスク群に存在している場合，いつ f が読み込み用ディスク群に移動するか分からない．そのため，f が読み込み用ディスク群に保存されているかどうか不明な場合と明確な場合でアクセスする方法が異なるため，それぞれ説明する．

- Case 1： クライアントにとって f が書き込み用ディスク群か読み込み用ディスク群に保存されているかどうか不明なとき
  1. $ID^F_{time}(f,t) = 2^{M_{time}} - 1$ とし，書き込み用ディスク群に保存されているものとしてアクセスする．
  2. 書き込み用ディスク群に保存されている場合はそのままアクセスする．それに対し，書

き込み用ディスク群には保存されていない場合は，目的のファイルは読み込み用ディスク群に保存されていることを記録し，読み込み用ディスク群に保存されていることが明確な場合の手続きを行う．

- Case 2： クライアントにとって f が読み込み用ディスク群に保存されていることが明確なとき
  1. f が現在所属する ID のグループに加わった時刻 t' を求める．具体的には $ID_{freq}(f, r_i)$，$ID_{freq}(f, r_{i-1})$，$\cdots$ を順に求めていき，f が初めて異なるグループとなるときの 1 つ前の時刻が t' となる．
  2. 1. で求めた t' を用いて，ID の上位 M_{time} ビットの値を以下の式の値として，f にアクセスする．

$$(2^{M_{time}}-1)-(r_i-time_{up}(f))+a\cdot freq(f,t').$$

ただし，読み込み用ディスク群において，4.6 節で説明するファイルを一部のディスクにまとめる作業により，f が ID に対応するディスクに保存されていない場合がある．その場合，そのディスクが所持する移動先リストを参照し，クライアントは実際に保存されているディスクにホップする．

4.5 ディスクとファイルの ID の更新方法

この節では，アクセス頻度が類似したファイルを同一のディスクに集約するためのディスクとファイルの ID の更新方法を説明する．まず，時刻 t において，書き込み用ディスク群のディスク d が，所持するファイルのデータ量が許容量を超えたとき， すなわち，$vol(d,t) \geq cap_{vol}(d)$ となったとき，d を書き込み用ディスク群から読み込み用ディスク群に移動する．その際に，d の ID は関数 ID^D に従い変更され，d に保存されたファイルの ID も ID^F に従い変更する．一方，読み込み用ディスク群の全てのディスクは一定時間 c ごとに ID^D_{time} を更新する．さらに，ディスクの ID^D_{time} を更新する際にディスクが所持する各ファイルの ID^F_{time} も更新する．更新する方法は，ファイル f の時刻 r_i において，ID を更新するとしたとき，$ID_{freq}(f, r_i)$ を調べる．この値が $ID^F_{time}(f, r_{i-1})$ の値と比較して同じグループであれば，$freq(f, r_i)$ の値を考慮せず更新する．すなわち，$ID^F_{time}(f, r_{i-1})$ から c だけ小さくする．異なるグループであれば，$ID_{freq}(f, r_i)$ の値に更新し，その値に従いディスク間の移動が必要な場合は対応するディスクに移動する．これらの操作により，アクセス頻度が類似したファイルが集約され，それに伴い，アクセスが少ないディスクを省電力モードに移行させる．

4.6 ディスクの使用効率を改善するための処理

読み込み用ディスク群では，ディスク間でファイルが移動した結果，使用率の低いディスクが生じることがある．そこで，使用率が一定量以下のディスクのファイルを定期的に集約し，ディスクの台数を出来る限り小さくする処理を行う．具体的には，グループごとに全てのディスクの使用率を調べ，グループの中で使用率が一定量以下のディスクを列挙し，それらの中で ID が高いディスクにファイルを集約する．ただし，移動したファイルの ID を担当し，本来そのファイルが保存されるディスクに，移動したファイルと移動先のディスクの組を記録し，クライアントからのアクセスも可能にする．

逆に，読み込み用ディスク群において，ディスク間のファイルの移動により，データの許容量を超えてしまうディスクが生じることも起こりうる．その場合は，許容量を超えてしまったディスクが担当する ID 空間を半分に分割するように ID を設定した新たなディスクを挿入し，許容量を超えたディスクに保存されたファイルを新たなディスクに移動し，負荷分散を行う．具体的には，時刻 t のとき，読み込み用ディスク群のディスク d_i において，ファイルの移動により，

$$vol(d_i,t) \geq cap_{vol}(d_i)$$

となるとき，空のディスクプールから新しいディスクの追加を行うことで，負荷分散を行う．そこで，追加するディスクを d_j とする．d_i と $pre(d_i,t)$ における関数 ID^D の値を用いて，d_j の ID^D の値を以下のように設定する．

$$ID^D(d_j,t) = ID^D(pre(d_i,t),t) + \frac{ID^D(d_i,t) - ID^D(pre(d_i,t),t)}{2}.$$

この ID をもとに Chord の join のアルゴリズムを用いて，ディスク d_j を追加することで，d_i のファイルを分散させる．

5 まとめと今後の課題

本研究では，分散ハッシュテーブルを用いた大規模ストレージシステムにおける省電力化手法を提案した．先行研究では，ファイルの保存場所の管理を単一のインデックスサーバにより行っていたが，本研究では，分散ハッシュテーブルによりファイルを管理する手法を提案した．これにより，本提案手法は先行研究と比較して拡張性が高い手法であると言える．さらに，先行研究では Flickr のアクセスパターンのみを用いてファイルの将来のアクセス頻度予測を行っていたので，他のサービスに対する有用性が不明であった．本研究では，Dailymotion におけるアクセスパターンの分析も行い，その結果，Flickr と同様の相関関係が観察された．このことは，本提案手法が多くのファイル共有サービスに対して有用であることを示唆している．

今後は，引き続き Dailymotion の動画ファイルのアクセス回数のトレースを行い，長い時間のアクセスパターンの分析を行う予定である．また，シミュレーションにより，提案手法の省電力性や応答性，ファイルの移動回数，さらにディスクの使用率を調べ評価する予定である．また，シミュレーション以外にも提案システムを実装し，より現実的な設定のもとでの提案手法の有用性を示す予定である．

参 考 文 献

[1] Dailymotion about. `http://www.dailymotion.com/jp/about`.

[2] Dailymotion API. `https://api.dailymotion.com/videos`.

[3] T. Bostoen, S. Mullender, and Y. Berbers. Power-reduction techniques for data-center storage systems. *ACM Comput. Surv.*, Vol. 45, No. 3, pp. 33:1–33:38, July 2013.

[4] D. Colarelli and D. Grunwald. Massive arrays of idle disks for storage archives. In *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*, SC '02, pp. 1–11, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[5] D. Harnik, D. Naor, and I. Segall. Low power mode in cloud storage systems. *Parallel and Distributed Processing Symposium, International*, Vol. 0, pp. 1–8, 2009.

[6] R. T. Kaushik and M. Bhandarkar. Greenhdfs: Towards an energy-conserving, storage-efficient, hybrid hadoop compute cluster. In *Proceedings of the 2010 International Conference on Power Aware Computing and Systems*, HotPower'10, pp. 1–9, Berkeley, CA, USA, 2010. USENIX Association.

[7] J. Kim and D. Rotem. Energy proportionality for disk storage using replication. In *Proceedings of the 14th International Conference on Extending Database Technology*, EDBT/ICDT '11, pp. 81–92, New York, NY, USA, 2011. ACM.

[8] J. Okoshi, K. Hasebe, and K. Kato. Power-saving in storage systems for internet hosting services with data access prediction. IGCC 2013, pp. 1–10, 2013.

[9] E. Pinheiro and R. Bianchini. Energy conservation techniques for disk array-based servers. In *Proceedings of the 18th Annual International Conference on Supercomputing*, ICS '04, pp. 68–78, New York, NY, USA, 2004. ACM.

[10] E. Pinheiro, R. Bianchini, and C. Dubnicki. Exploiting redundancy to conserve energy in storage systems. *SIGMETRICS Perform. Eval. Rev.*, Vol. 34, No. 1, pp. 15–26, June 2006.

[11] J. Richard. French video site takes on youtube with local offering, November 16 2006. `http://www.thetimes.co.uk/tto/news/world/article1982385.ece`.

[12] P. Sehgal, V. Tarasov, and E. Zadok. Optimizing energy and performance for server-class file system workloads. *Trans. Storage*, Vol. 6, No. 3, pp. 10:1–10:31, September 2010.

[13] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '01, pp. 149–160, New York, NY, USA, 2001. ACM.

[14] C. Weddle, M. Oldham, J. Qian, A. A. Wang, P. Reiher, and G. Kuenning. Paraid: A gear-shifting power-aware raid. *Trans. Storage*, Vol. 3, No. 3, October 2007.