広域分散ファイルシステムにおける アクセスパターンと性能を考慮したファイル配置

佐藤　仁[†1]　　松岡　聡[†1,†2]　　遠藤　敏夫[†1]

グリッドファイルシステムでの効率的な複製管理を実現するために, アクセス頻度や管理ポリシーに応じて, スループットやストレージ容量などの性能要件を満たし, かつ, 複製時間が最小になるように複製配置を決定するアルゴリズムを提案する. この複製配置問題をスループットやストレージ容量などの性能要件や複製転送のコストの最小化を目的関数とする組合せ最適化問題に帰着し, ファイルアクセスをモニタリングすることに得られた情報を利用することにより解く. 提案アルゴリズムをシミュレーションで評価した結果, 複製作成を行わない手法, アクセス時に複製をキャッシュする手法, サイト毎に複製を持つ手法などの単純な複製管理手法と比較して, ストレージ使用量を低く保ちつつ, かつ, 高いスループット性能を達成する複製配置を自動的に実現することを確認した.

Access-Pattern and Bandwidth Aware File Replication Algorithm for a Grid File System

Hitoshi Sato,[†1] Satoshi Matsuoka [†1,†2] and Toshio Endo[†1]

We propose an automated replication algorithm for a grid file system that considers file access frequency and replica maintenance policy, and that allows most of I/O accesses to be performed within given throughput and storage usage thresholds, while simultaneously minimizing replica transfer time. Our algorithm models the replication problem as a combinational optimization problem, where the constraints are derived from the given throughput and storage usage threshold, and various system parameters collected from direct file access monitoring. Our simulated-based studies suggest that the proposed algorithm can achieve higher performance than simple techniques, such as ones that always or never create replicas, while keeping storage usage very low. The results also indicate that the proposed algorithm can perform comparably with manual replica placement.

1. はじめに

近年, データインテンシブアプリケーションの実行環境としてグリッドの利用が実用的になりつつある. 広域に分散した複数のサイトの資源を集約して用いることで, 単一のサイトでは実現できない大規模な計算環境, ストレージを実現する. このような環境でのデータ共有の難しい点として, 広域に分散され配置されているデータの管理が挙げられる. 多くの場合, 異なるサイトでは異なるファイルシステムを用いており, アドホックで複雑なデータ管理をユーザ自身で行わなければならない.

このような難しさは, シングルシステムイメージを提供するグリッドファイルシステム[1),2)] を用いることで解決できる. これは, 広域に分散された資源の存在を意識することなく, 複数のアプリケーション間の連携がファイルベースで行える点などの利点があるためである. しかし, このようなファイルシステムを実際にグリッドで運用しようとした場合, 特定ノードへのアクセス集中や遠方のノードへのアクセスが発生し, アプリケーションの実行性能の低下が問題となる. このため, ファイルシステム側で必要に応じてファイルの移動や複製などのデータ管理を行うことで, ファイルアクセスのスループット性能の低下を抑える必要がある.

我々は, グリッドファイルシステムでの効率的な複製管理を実現するために, アクセス頻度に応じて, スループットやストレージ容量などの性能要件を満たし, かつ, 複製時間が最小になるように複製配置を決定するアルゴリズムを提案する. この複製配置問題をスループットやストレージ容量などの性能要件や複製転送のコストの最小化を目的関数とする組合せ最適化問題に帰着して解く.

†1 東京工業大学
Tokyo Institute of Technology
†2 国立情報学研究所
National Institute of Infomatics

提案アルゴリズムをシミュレーションで評価した結果, 複製作成を行わない手法, アクセス時に複製をキャッシュする手法, サイト毎に複製を持つ手法などの単純な複製管理手法と比較して, ストレージ使用量を低く保ちつつ, かつ, 高いスループット性能を達成する複製配置を自動的に実現することを確認した.

2. 関 連 研 究

従来のコンピュータシステムと同様に, グリッドファイルシステムでの I/O 性能の向上のために重要となるアイデアは, “いかに必要なデータをアクセス要求元の近くに置くか” という点である. 特に, グリッドファイルシステムは広域分散環境でのデータ共有を目的として使用され, その上で実行されるアプリケーションも write once, read mostly なワークロードを持つことなどから, 特に read アクセスのスループットの向上が重要である. しかし,“どのファイルをどのノードへ複製するか” という複製作成戦略の決定は難しい問題である. これは, 複製作成戦略がファイルアクセスパターンや許容する性能要件によってファイル管理ポリシーが変わるためである. また, すでにいくつかのノードに複製が存在しているような状況においても, “どの複製を選択してアクセスするか” は難しい問題である. これは, ファイルアクセスが複製の存在するノードへのネットワークのバンド幅の性能に大きく影響を受けるためである. 例えば,NFSv4[3],AFS[4],Google FS[5],Gfarm[6] などのファイルシステムは複製作成の機能を備えるが, ネットワークトポロジやアクセスパターンを考慮していない. HDFS[7] では, ストレージノードの物理的な場所を考慮した複製配置手法を提供しているが, ストレージノードの物理的な場所の認識は手動で行わなければならず, 広域分散環境では煩雑な操作となる. Tang ら[8] では, アプリケーションのワークロードを考慮した複製作成手法について触れているが, 単一サイトに配置されたクラスタファイルシステムを対象にしている. 我々の手法は広域分散環境を主な対象としている.

グリッドファイルシステム上でのファイルキャッシュは, 一般のファイルシステムでのファイルキャッシュと同様に, ファイルが将来どの程度アクセスされるのかに応じて, 最善のキャッシュ選択を行わなければならない. しかし, グリッドファイルシステム上でのファイルキャッシュでは, キャッシュされるファイルが元々どのノードに存在しているのか, という点を考慮することが重要である. 例えば, 近くのノード上に存在するファイルのキャッシュはアプリケーションの実行時間に寄与しないためキャッシュする必要がなかったり, 頻繁にアクセスはされないが遠いノード上に存在しているファイルのキャッシュは, 将来そのファイルへ再びアクセスするとき, アクセス時間を大幅に向上改善するためキャッシュすべきだったりする. したがって, 最善のキャッシュを実現する点においても, ネットワーク性能とアクセスパターンの両方を考慮する必要がある.

データグリッドの分野では, 数多くの複製手法が提案されている[9]. 特に,Rahman[10] や Wang[11] らは, 我々の手法と同様に, ファイル複製管理やロードバランスの問題を組み合わせ最適化問題に帰着することで解いている. しかし, これらの手法はファイルシステム全体の性能の最適化や I/O 性能と複製作成のために消費されるストレージ容量のトレードオフは考慮されていない.

3. 提 案 手 法

我々の提案手法は, アクセス頻度や複製管理ポリシーに応じて,read アクセスのスループットやストレージ容量などの性能要件をみたし, かつ, 複製時間が最小になるような複製配置を決定する. これを実現するために, ファイルアクセスをモニターし, 取得したデータからノード間のリンクのスループットやファイルへのアクセス頻度を推定する. 推定されたスループットやアクセス頻度を利用して, 複製管理戦略を決定する.

3.1 アクセスパターンの収集

提案手法では, ファイルアクセスの情報を必要とする. アクセス時刻, パス, アクセス種別 (read,write など), 実際に行われた I/O 量, 実行時間, また, アクセス先のホスト名をファイルアクセス毎に取得する. 我々は, このような情報を取得するための FUSE ライブラリベースのトレーサーを開発している. 全てのファイル操作の際にライブラリ関数が呼ばれ, 上記の情報を記録し, データベースに保存する. 我々はこれらの情報をノード間のスループットの推定やノードからファイルへのアクセス頻度の推定に用いる.

3.1.1 スループットの推定

収集されたファイルアクセス情報のうちの I/O 量と実行時間の値を用いて, ノード間のスループットを推定する. 我々は, 実行時間を以下のようにモデル化した.

$$time = \frac{1}{thput} \times io\_size + e$$

ここで,$time$ は実行時間,$thput$ はノード間のスループット,io_size は I/O 量を表す. 実際の実行時間は様々な要因により変動するため,e を誤差として含むこととする. 各ノード間に対して, アクセス情報から上述のモデルへ最小二乗法を適用することにより, スループットを推定する. 最終的に, スループットを表す行列 $D_{i,j} = (d_{i,j})$ を生成する. ここで,$d_{i,j} = \frac{1}{thput}$ をノード i,j 間のスループットの逆数とする. ただし,$i = j$ のときは,$d_{i,j} = 0$ とする. この手法はノード間のリンクのスループットの時間的変動が比較的小さいという仮定を置いている.

3.1.2 アクセス頻度の推定

ノード i からファイル f に対するアクセス頻度を表すために, 将来アクセスが発生する確率を次のように定義する. ノード i からファイル f へのアクセスが発生する時間間隔がパラメタ $\lambda(>0)$ の指数分布に従うと仮定すると, 確率密度関数は,

$$f(x;\lambda) = \begin{cases} \lambda e^{-\lambda x} & (x \geq 0) \\ 0 & (x < 0) \end{cases}$$

となる. 収集されたファイルアクセス情報のうちのアクセス時刻, アクセス種別の情報より, アクセスの発生間隔が取得できるので, 最尤推定あるいはベイズ推定を行うことにより,λ を求まる. 決定されたパラメタを $\dot{\lambda}$, 最後にノード i からファイル f へのアクセスが発生した時刻を t_0, 現在の時刻を t とすると, 将来ノード i からファイル f へ発生する確率 p_i は次のようになる.

$$p_i = \int_{t-t_0}^{\infty} \dot{\lambda} e^{-\dot{\lambda}x} dx = e^{-\dot{\lambda}(t-t_0)}$$

この値 p_i をノード i からのファイル f に対するアクセス発生確率とする. すなわち, ノード i からファイル f へのアクセスの発生が現在の時刻 t に近いほど p_i の値は 1 に近づき, 遠くなるほど 0 に近づく.

3.2 複製配置の決定

推定されたノード間のスループットを用いて,read アクセスのスループットが h 以上, 複製により消費されるストレージ容量が r 以下となるような条件を満たすように複製配置を決定する. この h,r の値はファイルへのアクセス頻度や管理ポリシーに応じて, $H_{min} \leq h \leq H_{max}$,$R_{min} \leq r \leq R_{max}$ と変動することとする. ここで,H_{max},H_{min} は期待する read アクセスのスループットの最大値, 最小値を表し,R_{max},R_{min} は許容する複製により消費される容量の最大値, 最小値を表す. これらのパラメタはユーザあるいはシステムにより予め指定される. すなわち, 頻繁にアクセスされるファイルに対しては h を H_{max}, そうでないファイルに対しては h を H_{min} に近づける. また, スループット性能を重視して複製管理を行うときは,r を R_{max} に近づけ, ストレージ容量を重視して複製管理を行うときは,r を R_{min} に近づける. このような操作を行うことで,read アクセスのスループット性能と複製作成のためのストレージ容量の制約をアクセス頻度, 複製管理ポリシーに応じて変更し最適な複製配置を実現する. 我々はこの複製配置の決定を多目的整数計画問題としてモデル化した.

サイズ s のファイル f の複製の配置の決定を以下のように行う. $i \in \{1,\cdots,n\}$ を対象環境のノード名とし, ノード i 上で利用可能なストレージ容量を c_i する. また, ノード i 上に複製が存在しているかどうかを $x_i \in \{1,0\}$ で表す. $x_i = 1$ のときはノード i に複製が存在することを表し,$x_i = 0$ のときはノード i に複製が存在しないことを表す. ファイルへのアクセス頻度や複製の管理ポリシーに応じて, ノード i からの read アクセスのスループットと複製により消費されるストレージ容量の両方の制約をみたすような複製の配置 x_i を求めることである。

以下では,read アクセスのスループットに関する議論を read アクセス時間に関する問題として帰着する. 我々は,read アクセス時間を次のようにモデル化した. ただし, ここでは, 簡単のために read アクセスはファイルの全ての領域を読むと仮定する. ノード i からノード j へのアクセスを $y_{ij} \in \{0,1\}$ で表す. $y_{ij} = 1$ のときはノード i からノード j へのアクセスがあることを表し, $y_{ij} = 0$ のときはノード i からノード j へのアクセスがないことを表す. この $y_{i,j}$ を用いると, ノード i からノード j 上にへの read アクセスの時間 $v_{i,j}$ は,

$$v_{i,j} = s \cdot d_{i,j} \cdot y_{i,j}$$

と期待される. ただし,$d_{i,j}$ はノード i,j 間のスループットの逆数とする. 例えば, ノード i がノード j にアクセスしない場合,$y_{i,j} = 0$ となるのでアクセス時間は 0 になる. 一方, ノード i がノード j にアクセスする場合,$s \cdot d_{i,j}$ がアクセス時間となる. 今,read アクセスのスループットを h としてその逆数を $d(h) = \frac{1}{h}$ (ただし,$h = 0$ のとき,$d(h) = 0$) と表すと, 制約式は,

$$v_{i,j} \leq s \cdot d(h)$$

と表される. ここで,h は, アクセス頻度に応じて,$H_{min} \leq h \leq H_{max}$ と変動させるので, ノード i からファイル f へのアクセス確率 p_i を用いて,

$$h = p_i \cdot H_{max} + (1 - p_i) \cdot H_{min}$$

とする.

また, 作成される複製の数は,x_i がノード i 上での複製の存在を表すので, $\sum_{i=1}^{n} x_i$ と表される. 従って, 複製作成のために許容するストレージの容量の制約は, ストレージ容量を r とし, 複製の存在を考慮することで,

$$\frac{R_{min}}{s} + 1 \leq \sum_{i=1}^{n} x_i \leq \frac{R_{max}}{s} + 1$$

と表される.

今, 求める問題は予め指定されたパラメタ H_{max}, H_{min},R_{max},R_{min} が与えられたときに上記の制約をみたすような複製配置 x_i を求める問題として, 次のように記述できる.

Minimize

$$\max_{i,j=1,\ldots,n} v_{i,j} \tag{1}$$

$$\sum_{i=1}^{n} x_i \tag{2}$$

Subject to

$$x_i \in \{0,1\} \tag{3}$$
$$y_{i,j} \in \{0,1\} \tag{4}$$
$$v_{i,j} = s \cdot d_{i,j} \cdot y_{i,j} \leq s \cdot d(h) \tag{5}$$
$$h = p_i \cdot H_{max} + (1-p_i) \cdot H_{min} \tag{6}$$
$$\frac{R_{min}}{s} + 1 \leq \sum_{i=1}^{n} x_i \leq \frac{R_{max}}{s} + 1 \tag{7}$$
$$\sum_{j=1}^{n} y_{i,j} = 1 \tag{8}$$
$$x_j = 1, \text{if } \sum_{i=1}^{n} y_{i,j} > 0 \tag{9}$$
$$s \cdot x_i \leq c_i \tag{10}$$

目的関数は,(1) ノード i,j 間の最大 read アクセス時間を最小化し,(2) 複製数を最小化することを表している. 前述の制約式 (3),(4),(5), (6),(7) の他に, 更に 3 つの制約式を加える. まず,(8) は各ノードは必ず複製を持つ他のノードにアクセスするということを表す.(9) は, 任意のノードからのアクセスがノード j にあれば, ノード j は必ず複製を持つということを表す. また,(10) は複製のサイズがノードのストレージの容量を超えないということを表す.

今,H_{max},H_{min},R_{max},R_{min} が与えられているとする. (6) よりファイルのアクセス頻度に応じて $\dot{h}$ と定まるとする. ここで, 複製管理において read アクセスのスループットを重視するか, ストレージ容量を重視するか, を決定するパラメタ $\alpha(0 \leq \alpha \leq 1)$ を導入し,$\dot{r}$ を以下のように表す.

$$\dot{r} = \alpha \cdot R_{max} + (1-\alpha) \cdot R_{min}$$

すなわち, スループットを重視する場合, 複製数の制約 $\dot{r}$ を α を 1 に近づけることで R_{max} に近づけ, ストレージ容量を重視する場合,α を 1 に近づけることで $\dot{r}$ を R_{min} に近づける. このとき, 上記の問題は, 以下の目的関数

$$\max\left\{ \max_{i,j=1,\cdots n} v_{i,j} - s \cdot d(\dot{h}), \sum_{i=1}^{n} x_i - (\frac{\dot{r}}{s} + 1) \right\}$$

を制約条件 (3),(4),(8),(9),(10) の元で最小化する問題に帰着できる. すなわち, 定められたポリシー H_{max},H_{min},R_{max},R_{min},α を満たしつつ, スループットを最大化, ストレージ容量を最小化する複製配置を実現する. また,H_{max},H_{min} が与えられてストレージ容量に制約がない場合, (1) を考慮せず,(7) を $\sum_{i=1}^{n} x_i \geq 1$ と変えた制約条件の元での (2) 複製数の最小化問題と帰着し, R_{max},R_{min} が与えられてスループットに制約がない場合, (2),(5) を考慮しない制約条件の元での (1)read アクセス時間の最小化問題と帰着することで, 様々な複製配置のポリシーに柔軟に対応することができる.

この最適化問題を解くことにより,$i \in \{1, \cdots, n\}$ に対して x_i の値を決定し, この結果に応じてファイル f の複製配置を行う.

3.3 複製転送の決定

今, ファイル f の複製配置を決定したので, どのように効率的に現在の複製配置から新しい複製配置を実現するかを決定する. つまり, 現在の複製配置から新しい複製配置への複製の総転送時間の最小化を試みる. $\bar{x_i}$ を現在の複製配置とし,x_i を新しい複製配置とする. ノード i からの総転送時間 w_i は,

$$w_i = \sum_{j=0}^{n} s \cdot d_{i,j} \cdot z_{i,j}$$

と表される. ここで,$z_{i,j} \in \{1,0\}$ は, ノード i がノード j へ複製を転送するかどうかを表す. つまり,$z_{i,j} = 1$ のとき, ノード i がノード j に複製を転送し,$z_{i,j} = 0$ のとき, 転送しないことを表す.

今, 複製の総転送時間 $\max_{i=1,\cdots n} w_i$ を最小化したいので, 複製転送の決定は次のような整数計画問題として定式化できる.

Minimize

$$\max_{i=1,\ldots,n} w_i \tag{11}$$

Subject to

$$\bar{x_i}, x_i \in \{0,1\} \tag{12}$$
$$z_{i,j} \in \{0,1\} \tag{13}$$
$$\sum_{i=1}^{n} z_{i,j} \in \{0,1\} \tag{14}$$
$$w_i = \sum_{j=1}^{n} s \cdot d_{i,j} \cdot z_{i,j} \tag{15}$$
$$\bar{x_i} = 1, \text{if } \sum_{j=1}^{n} z_{i,j} > 0 \tag{16}$$
$$x_j = \sum_{i=1}^{n} z_{i,j} \tag{17}$$

目的関数 (11) はノード間での最大総転送時間を最小化する. 前述の制約式 (12), (13), (15) の他に, 更に 3 つの制約式を加える. まず,(14) は, 同じファイルの複製を同じノードに作成することを避けることを表す. (16) は, ノード i からの複製の転送が発生していたら, そのノード i には予め複製が存在していることを表す. 最後に,(17) は, ノード j への転送が発生していたら, そのノード j には必ず複製が存在することを表す.

この最適化問題を解くことにより,$i, j \in \{1, \cdots, n\}$ に対して, $z_{i,j}$ の値を決定し, この結果に応じて複製の転送を行う.

4. 実　　験

提案手法の有効性を検証するためにシミュレーションによる評価を行った. 図 1 にシミュレーションでのグリッド構成を示す. この環境で, ファイルの集合に対してアクセスするジョブがバースト的に発生する

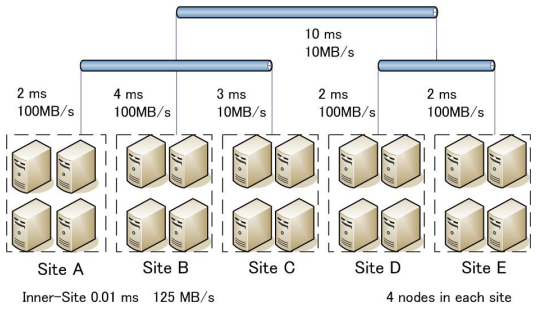

図 1 シミュレーションの構成

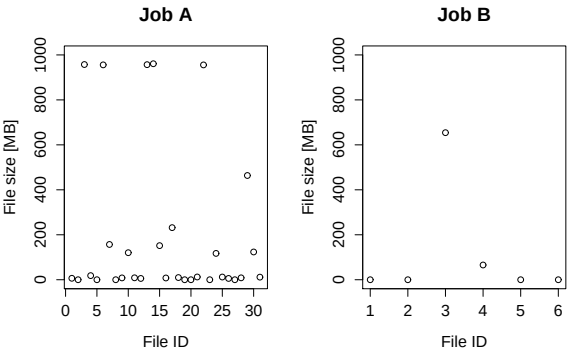

図 2 ファイルサイズの分布

表 1 ジョブの総実行時間とストレージ使用量

	Elapsed time (h)	Storage usage (GB)
no rep	1900	6.82
on demand	7.20	134
static	5.40	34.1
proposal	14.4	17.1

状況を再現した．シミュレーションでは, バイオインフォマティクスの分野で広く利用されている相同性検索ツール BLAST[12] を対象アプリケーションとした．図 2 は,BLAST より収集した 2 つのジョブ A,B でのアクセスされるファイルのサイズの分布である．x 軸はファイルの ID を表し,y 軸はサイズを表す．ファイルの集合に対して,ID の小さい順に順次アクセスするジョブをワークロードとして与えた．これは, データインテンシブアプリケーションでよくみられる典型的なワークロードである[13]．この設定で, 以下のように,2 つのフェーズ (1 フェーズ 6 時間) からなるワークロードを与えた．

フェーズ 1 サイト A,B,C の各ノードがジョブ A を 10 回投入し, サイト D,E の各ノードがジョブ B を 10 回投入する．

フェーズ 2 サイト A,B の各ノードがジョブ B を 10 回投入し, サイト C,D,E の各ノードがジョブ A を 10 回投入する．

シミュレーションでは, 全てのファイルは最初にサイト C の 1 ノードに全てのファイルが置かれている設定とした．このワークロードを異なる 3 つのポリシーと提案手法を比較した．

- レプリケーションを行わない (*no rep*). 各ノードがファイルを持つ遠方のノードへアクセスする．
- 最初にアクセスした際にオンデマンドにアクセス要求元のノードへ複製する (*on demand*).
- 一定数の複製を作成する (*static*). シミュレーションでは, 開始から終了時各サイトの 1 つのノードが 1 つだけ複製を持つ．

提案手法 (*proposal*) では 1 時間に 1 回複製配置の再構成を行った．また, パラメタは,$H_{max} = 125, H_{min} = 1, R_{max} = 10, R_{min} = 0$, $\alpha = 0.5$ と設定した．ただし,$H_{max,min}$ の単位は MB/s で,$R_{max,min}$ の単位は GB とし, α は複製管理ポリシーを決めるパラメタで $0 \leq \alpha \leq 1$ ある．

表 1 は各複製管理手法でのジョブの総実行時間と複製により消費されたストレージの使用量を比較したものである．実行時間とストレージ使用量の間にトレードオフの関係があることがわかる．“no rep” 手法は最も実行時間が長くなっているが, ストレージ使用量は最も低くなっている．一方で,”on demand” 手法は最も実行時間が短くなっているが, ストレージ使用量は最も高くなっている．提案手法 (proposal) は,”no rep” 手法と比較して 132 倍の実行性能の向上を示し,”on demand” 手法と比較してストレージの使用量を 7.84%に抑えた．また,”static” 手法に匹敵する実行性能とストレージ使用量を自動的に実現した．この”static” 手法は, 実際のグリッド環境では, 幾つか問題がある．サイト毎にアドホックな複製管理をしなくてはならなかったり, サイトに存在するノード数の違いやネットワーク構成の違いによって, アクセス集中や遠方のアクセスなどが発生してしまい,”no rep” 手法と同様に大幅な性能低下が発生してしまうためである．

図 3, 図 4 は, それぞれ, シミュレーションにおける実行が終了していない累積ジョブ数の変化, ストレージ使用量 [GB] の変化を表したものである．図中,x 軸はシミュレーションの経過時間 (sec) を表す．我々の手法は, アクセス頻度や複製管理ポリシーに応じて, 適切な複製数と自動的に配置場所を自動的に決定していることが確認できる．特に, ジョブが終了し, ファイルへのアクセスがなくなった後も, 複製管理を継続し, 管理ポリシーに応じた最適な複製数を保持し続け, 不必要な複製の保持を避ける．

5. おわりに

グリッドファイルシステムでの効率的な複製管理を実現するために, アクセス頻度に応じて, スループットやストレージ容量などの性能要件を満たし, かつ, 複製

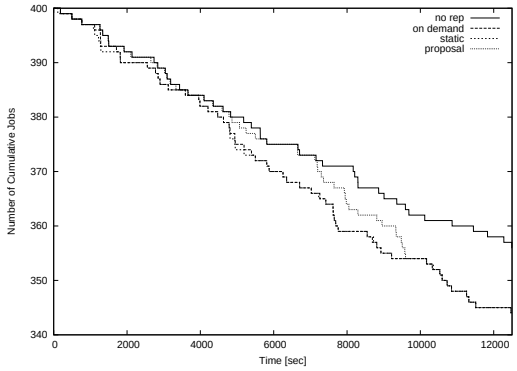

図 3 累積ジョブ数の変化

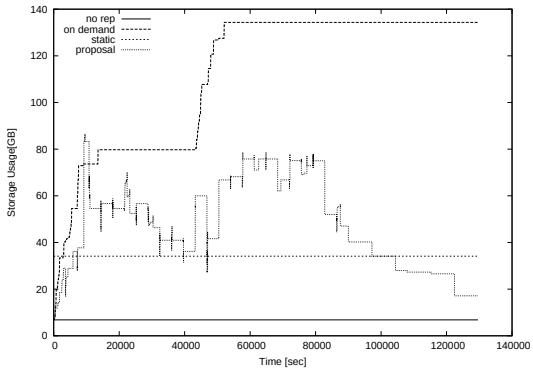

図 4 ストレージの使用量の変化

時間が最小になるように複製配置を決定するアルゴリズムを提案した. 提案アルゴリズムをシミュレーションで評価した結果, 複製作成を行わない手法, アクセス時に複製をキャッシュする手法, サイト毎に複製を持つ手法などの単純な複製管理手法と比較して, ストレージ使用量を低く保ちつつ, かつ, 高いスループット性能を達成する複製配置を自動的に実現することを確認した.

この提案手法はプロトタイプとして Gfarm[1] に実装している. これを, 大規模な実行環境及びタスクセットへ対応し, シミュレーションと実環境で比較評価していくことで, 各種パラメタの適切な設定を探るのが今後の課題である.

謝辞 本研究の一部は科学研究費補助金特定領域研究 (18049028) と JSPS GCOE プログラム「計算世界観の深化と展開」の補助による.

参 考 文 献

1) Gfarm, `http://datafarm.apgrid.org`.
2) XtreemFS, `http://www.xtreemfs.com`.
3) Callaghan, B., Robinson, D., Thurlow, R., Beame, C., Eisler, M. and Noveck, D.: Network File System (NFS) version 4 Protocol (2003), RFC 3530.
4) Howard, J. H., Kazar, M. L., Menees, S. G., Nichols, D. A., Satyanarayanan, M., Sidebotham, R. N. and West, M. J.: Scale and Performance in a Distributed File System, *ACM Transactions on Computer Systems (TOCS)*, Vol.6, No.1, pp. 51–81 (1988).
5) Ghemawat, S., Gobioff, H. and Leung, S.-T.: The Google File System, in *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pp. 96–108, Bolton Landing, NewYork (2003).
6) Tatebe, O., Morita, Y., Matsuoka, S., Soda, N. and Sekiguchi, S.: Grid Datafarm Architecture for Petascale Data Intensive Computing, in *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2002)*, pp. 102 – 110 (2002).
7) Hadoop, `http://hadoop.apache.org/`.
8) Tang, H., Gulbeden, A., Zhou, J., Strathearn, W., Yang, T. and Chu, L.: A Self-Organizing Storage Cluster for Parallel Data-Intensive Applications, in *In Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, p.52 (2004).
9) Venugopal, S., Buyya, R. and Ramamohanarao, K.: A taxonomy of Data Grids for distributed data sharing, management, and processing, *ACM Computing Surveys*, Vol. 38, No.1, p.3 (2006).
10) Rashedur M. Rahman, K. B. and Alhajj, R.: Study of Different Replica Placement and Maintenance Strategies in Data Grid, in *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid* (2007).
11) Wang, Y. and Kaeli, D.: Load Balancing using Grid-based Peer-to-Peer Parallel I/O, in *In Proceedings of IEEE International Conference on Cluster Computing (Cluster 2005)*, pp. 1 – 10 (2005).
12) NCBI BLAST, `http://www.ncbi.nlm.nih.gov/BLAST`.
13) Li, H. and Wolters, L.: Towards A Better Understanding of Workload Dynamics on Data-Intensive Clusters and Grids, in *Proceedings of 21th International Parallel and Distributed Processing Symposium*, pp. 1 – 10, Long Beach, California, USA (2007).