分散ファイルシステム GPFS を用いたスケーラブルな非同期コピーの提案

松井壮介[†]　三好浩之[†]
高井聡[†]　荒木博志[†]

分散ファイルシステムの同一ディレクトリを複数のノードが同時に更新する際，ファイルシステムの一貫性を保証するための排他制御による書き込み遅延が発生し，書き込み速度が向上しない，という問題が知られている．この問題は，複数のノードで大量のファイルを短時間に作成するスケールアウト NAS の非同期コピーにおいて，特に顕著に見られる．ディレクトリの更新などをスケーラブルに行えるよう，分散ファイルシステムを設計する方法が提案されてきたが，これらの方法は，その設計を取り入れたファイルシステムの使用を前提としていた．本稿では，使用するファイルシステムに依存せず，スケーラブルな非同期コピーを実現するために，分散ファイルシステムの排他制御による書き込み遅延とファイルサイズの関係に着目し，サイズの大きいファイル群はサイズが均等になるようにノード間でファイル転送処理の負荷を分散させ，サイズの小さいファイル群は同一ディレクトリごとに集めて同一ノードから転送する方法を提案する．本手法により，ファイル転送処理の負荷分散と書き込み遅延の解消を同時に実現できる．本手法を非同期コピーに適用することで，ノード数に応じてファイル転送速度が向上することを確認できた．

A Proposal of Scalable Asynchronous Replication on GPFS Distributed File System

Sosuke Matsui,[†] Hiroyuki Miyoshi,[†] Satoshi Takai[†] and Hiroshi Araki[†]

It is well known that write performance is not scalable if multiple nodes update the same directory in parallel on distributed file system. This is due to the fact that distributed file system forces exclusive access to the directory in order to maintain consistency. This problem appears prominently in asynchronous replication on scale-out NAS that creates a lot of files in a short period of time. There are some researches that propose scalable metadata operation on distributed file system, but these approaches force to use proposed file systems. In this paper, we propose to distribute large files among nodes based on file size, and assign small files in the same directory to the same node in order to implement scalable asynchronous replication that does not depend on file system. Our method enables workload distribution and resolves delayed write at the same time. By applying our method, we verified that the file transfer rate of asynchronous replication became scalable on our test environment.

1. はじめに

非同期コピーとは，運用中のストレージ（以下，コピー元ストレージ）のデータのコピーを，ホストからの I/O とは非同期に，遠隔地に設置したストレージ（以下，コピー先ストレージ）に定期的に作成する機能である．非同期コピーはスケールアウト Network Attached Storage (NAS) を含む多くのストレージ製品で利用可能である．

† 日本アイ・ビー・エム　ストレージ・システムズ開発
IBM Japan, Storage Systems Development

スケールアウト NAS とは複数のノードから構成されるクラスタ型のファイルサーバ専用装置である．分散ファイルシステムを搭載し，ノード間で同一ファイルシステムを共有する．ノード数に応じて，I/O 性能やストレージ容量が向上する，という特徴を持つ．非同期コピーをはじめとする，スケールアウト NAS が提供する機能も，ノード数に応じて性能が向上するように設計される必要がある．

これまで IBM Scale Out Network Attached Storage (SONAS)[7]，EMC Isilon[4]，Hitachi NAS Platform[5] などのスケールアウト NAS が発売され，いずれも非同期コピーが利用可能となっている．スケール

アウト NAS の非同期コピーでは，一般に，ファイル転送プロセスを複数のノードで並列に実行し，コピー元ストレージのファイルツリーの複製をコピー先ストレージに高速に作成する．さらに，ノードを追加することでファイル転送プロセスの数も増加し，ファイル転送速度が向上する．

分散ファイルシステムにおいて複数のノードが同一ディレクトリを同時に更新すると，ファイルシステムの一貫性を保証するための排他制御による書き込み遅延が発生し，書き込み速度が向上しない，という問題が発生する[11),12)]．この問題は，複数のノードを使って短時間に大量のファイルを作成するスケールアウト NAS の非同期コピーにおいて，特に顕著に見られる．

分散ファイルシステムに対してディレクトリ更新処理をスケーラブルに行えるように，ファイルシステムを設計する方法が考えられてきた[1),3),10)~12),15)~17)]．一方で，General Parallel File System (GPFS)[14)] や Global File System (GFS)[13)] などの分散ファイルシステムは，これらの手法が提案するように設計されておらず，依然として，複数のノードが同一ディレクトリを更新すると書き込み遅延が発生する，という問題を抱えている[11)]．そこで，どのような分散ファイルシステムに対してもスケーラブルに稼動する非同期コピーを実現するためには，ファイルシステムを設計し直すのではなく，非同期コピーに修正を加えるほうが容易である．

SONAS の非同期コピーにおいて，分散ファイルシステムの一貫性の保証に起因する書き込み遅延を解消するために，我々は，先行研究でファイルを同一ディレクトリごとに集め，同一ディレクトリ下のファイルは同一ノードが転送する，という手法を提案し，修正を加えた[18)]．しかし，この手法を適用すると，同一ディレクトリにサイズの大きいファイルが集中する場合，あるノードに転送処理の負荷が集中し，転送速度が向上しないことが判明した[19)]．

以上から，スケーラブルな非同期コピーを実現するためには，ファイル転送処理の負荷を分散すると同時に，排他制御による書き込み遅延の解消を実現する必要がある．そこで，我々は，排他制御による書き込み遅延がサイズの小さいファイルを転送する際に顕著になる，という点に着目し，サイズの大きいファイル群はサイズが均等になるようにノード間で転送処理の負荷を分散し，サイズの小さいファイル群は同一ディレクトリごとに集め，同一ディレクトリ下のファイルを同一ノードから転送する，という方法を提案する．また，転送するファイルのサイズから排他制御による書き込み遅延を算出するモデルを導入し，転送方式を切り替えるファイルサイズを判定する方法も提案する．本手法の適用により，同一ディレクトリに集中するサイズの大きいファイルの転送処理が各ノードに分散されると同時に，サイズの小さいファイルの転送時の排他制御を回避できる．さらに，本手法は非同期コピーに修正を加えるため，排他制御による書き込み遅延が発生するどのような分散ファイルシステムに対しても，スケーラブルに稼動する非同期コピーを実現することができる．分散ファイルシステムに変更を加えず，分散ファイルシステムへの書き方を工夫してスケーラブルな機能を実現している手法は，我々の知る限り存在しない．本稿で提案する手法を SONAS の非同期コピーに実装し，ノード数に応じて転送速度が向上することを確認した．なお，本稿ではコピー元ストレージとコピー先ストレージの間のネットワークに十分な帯域が確保されている環境を想定している．

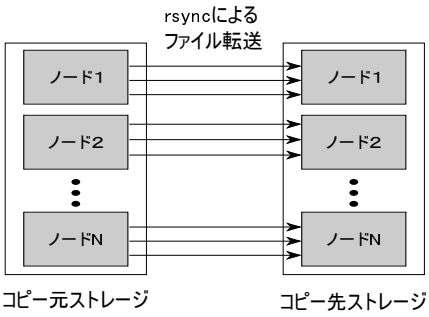

図 1 SONAS 非同期コピーの概略図
Fig. 1 Overview of SONAS asynchronous replication

本稿では，スケールアウト NAS の中でも，特に SONAS の非同期コピーをスケーラブルに実現する方法について説明するが，本手法は SONAS に限らず，分散ファイルシステムの排他制御による書き込み遅延が発生するスケールアウト NAS に対して効果がある．以下，本稿では 2 節で SONAS の非同期コピーの概要と課題を説明する．3 節で提案手法，4 節でその評価結果について述べる．5 節で関連研究について説明し，6 節で結論を述べる．

2. SONAS の非同期コピーにおける課題

2.1 SONAS とは

SONAS とは，分散ファイルシステムに GPFS を採用したクラスタ型の NAS 製品である．複数のノードで GPFS クラスタを 1 つ構成し，各ノードが同一ファイルシステムにアクセスする．ノードを追加することで，I/O の性能やストレージ容量が向上する，という特徴を持つ．

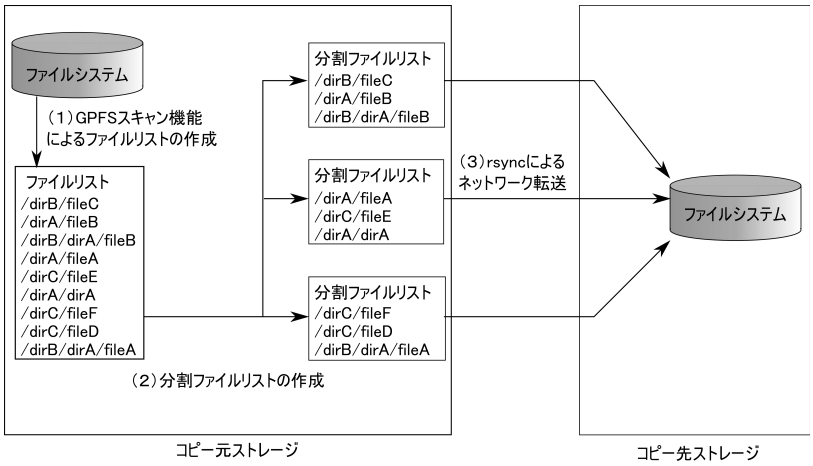

図 2 SONAS 非同期コピーの流れ
Fig. 2 Flow of SONAS asynchronous replication

SONAS の非同期コピーは，2 つのストレージ間でデータ転送を行い，コピー元ストレージのファイルツリーの複製を，コピー先ストレージにて作成する．非同期コピーはファイル単位で行われ，rsync[2] によりコピー元ストレージのファイルがコピー先ストレージへネットワーク転送される．**図 1** のように，コピー元ストレージとコピー先ストレージのノードは 1 対 1 の関係になっており，各ノードで複数の rsync プロセスが並列に実行される．非同期コピーに使用するノードの数（最小 1 台，最大 30 台）と，各ノードで実行する rsync プロセスの数（最小 1 プロセス，最大 10 プロセス）は管理者が自由に設定できる．

SONAS 非同期コピーの流れは以下の通りであり，これを模式化したものを**図 2** に示す．

(1) コピー元ストレージにおいて，前回の非同期コピーの実行時から現在までに更新があったファイルのリストを，GPFS のスキャン機能を用いて作成する．ファイル・リストには，ファイルの絶対パスが記載される．
(2) ファイル・リストを，非同期コピーに使用するノードの数（図 2 では 3 つ）で分割し，各ノードに分配する．各分割ファイル・リストに記載されるファイルの数は均等にする．
(3) 各ノードで複数の rsync プロセスを実行する場合は，分割ファイル・リストをさらに rsync プロセスの数で分割し，各 rsync プロセスが分割ファイル・リスト上のファイルを，コピー先ストレージに並列に転送する．

2.2 同一ディレクトリ更新による排他制御の影響

あるノードが GPFS にファイル等を新規作成する場合，まず，そのファイルの親ディレクトリを更新する権利を取得し，次いでそのディレクトリに対する更新処理を実行する．ディレクトリの更新処理とは，新規作成するファイルを親ディレクトリに追加する処理を指す．複数のノードが同一ディレクトリに同時にファイル等を新規作成する場合，あるノードがディレクトリを更新した後，別のノードに更新権が渡る前に，更新の内容が必ずディスクに書き込まれる[14]．このディスクへの書き込み処理により遅延が発生し，複数のノードが同一ディレクトリを更新すると，書き込み速度が劣化し，ノード数に応じて書き込み速度が向上しない，という問題（以降，ディレクトリ競合と呼ぶ）が発生してしまう[11]．SONAS の非同期コピーでは複数のノードが並列に rsync を用いてファイル転送をするため，異なるノードで実行される rsync プロセスがコピー先ストレージの同一ディレクトリ下にファイルを転送する場合，ディレクトリ競合が発生する．

ディレクトリ競合により非同期コピーの性能が向上しないことを確認するために，テスト環境で検証を行った．検証のために，Linux の Kernel-based Virtual Machine （KVM）を用いて作成した 8 台の仮想マシンに GPFS をインストールし，これら全ての仮想マシンを用いて GPFS クラスタを 1 つ構成した．8 台のノードは，SONAS の基本ラックに構成可能な最大のノード数となっている[8]．次いで，仮想マシン間で共有するファイルシステムを作成した．さらに，実環境を模倣するために，仮想マシンとホスト OS との間の仮想ネットワークに 10ms の遅延を設定した．この GPFS クラスタの共有ファイルシステムに対して，rsync を用いてホスト OS からファイルをコピー

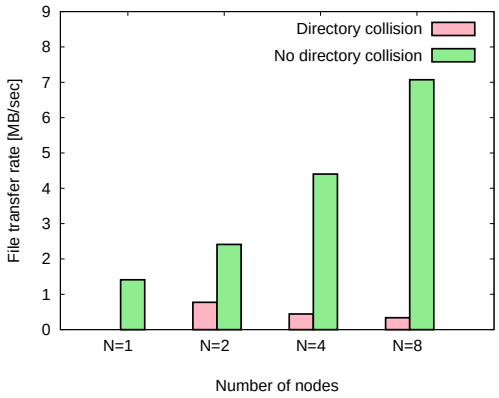

図 3 ディレクトリ競合による排他制御の影響
Fig. 3 Effect of exclusive control due to directory collision

し，転送に要する時間を測定した．検証の手順は以下の通りである．

(1) ホスト OS にディレクトリを N 個作成する．
(2) 各ディレクトリに 10kB のファイルを 1024 個作成する．
(3) ホスト OS で N 個の rsync プロセスを並列に実行させ，(2) で作成したファイルを仮想マシンが共有するファイルシステムに転送する．各プロセスは N 個の異なる仮想マシンに対してファイル転送を行う．
(4) ファイルの転送速度を測定する．

上記の手順で，$N = 2, 4, 8$ の 3 通りについて，以下の 2 つの転送方法による転送速度の比較を行った．

- ディレクトリ競合が発生するように，ファイル転送を行う．具体的には，各ディレクトリのファイルを，数が均等になるように N 個のグループに分割する．各 rsync プロセスは，同時に同一ディレクトリ内の異なるグループのファイルを転送する．転送先のファイルシステムには異なるノードが同時に同一ディレクトリを更新することになる．
- ディレクトリ競合が発生しないように，ファイル転送を行う．具体的には，各ディレクトリのファイルは，それぞれ異なる rsync プロセスが転送する．転送先のファイルシステムには異なるノードが異なるディレクトリを更新することになる．

さらに，$N = 1$ に設定し，全ファイルを 1 つのノードに転送させた場合の転送速度も測定した．この場合も，ディレクトリ競合は発生しない．測定結果を**図 3** に示す．この結果から，ディレクトリ競合が発生する場合は，ノード数に応じて転送速度が向上せず，複数のノードを使用すると転送速度が劣化することが確認できた．一方，ディレクトリ競合が発生しない場合は，ノード数の増加に伴い転送速度が向上することも確認できた．したがって，複数のノードで並列にファイル転送を行う場合，転送速度を向上させるためにはディレクトリ競合を回避する必要がある．

上記の検証を含め，以下，本稿では仮想マシン上で測定を行う．複数の仮想マシンを同一サーバ上で稼動させることにより，ハードウェア資源の競合が発生し[6]，測定結果へ影響を与えることになる．ただし，上記の検証において，ディレクトリ競合が発生する場合と発生しない場合とではファイル転送方式のみが異なり，複数の仮想マシンによるハードウェア資源の競合は，どちらの転送方式にも同等の影響を与える，と考えられる．したがって，我々は，仮想マシン上での検証でも，転送方式の違いによる転送速度の傾向の差を観測することができると考える．

2.3 ファイルサイズを考慮しない同一ノードによる同一ディレクトリの更新

我々の先行研究では，ディレクトリ競合に対応するために，ファイルを同一ディレクトリごとに集め，同一ディレクトリ下のファイルは同一ノードが転送する，という修正を SONAS の非同期コピーに加えた[18]．具体的には，図 2 のファイル・リストを分割する前に，ファイルパスでソートしてから分割することで，同一分割ファイル・リストに同一ディレクトリ下のファイルが現れやすくなるようにした．これにより，ディレクトリ競合による影響を削減することに成功したが，サイズの大きい仮想マシンのイメージファイルやデータベースファイルなどが同じディレクトリに集中する場合，新たな問題が顕在化した．文献 18) で提案した方法は，ファイルの数が均等になるようにファイル・リストを分割するため，サイズの大きいファイルがあるディレクトリに集中すると，ある分割ファイル・リストに記載されるファイルサイズの合計値が他の分割ファイル・リストと比較して，突出して大きくなることがあった．このため，ファイルツリーの構成によって，非同期コピーを実行するとサイズの大きいファイルの転送処理が特定のノードに集中し，転送速度が向上しない，という問題が発生した．

この問題を調査するために，2.2 節の実験と同じ環境を用い，以下の手順で問題を再現した．

(1) ホスト OS のあるディレクトリに 1GB のファイルを 10 個作成する．
(2) ホスト OS の別のディレクトリ下に各サブディレクトリにファイルが 6 個，ディレクトリが 5 個存在する深さが 4 のファイルツリーを作成する．各ファイルのサイズは 1MB とする．

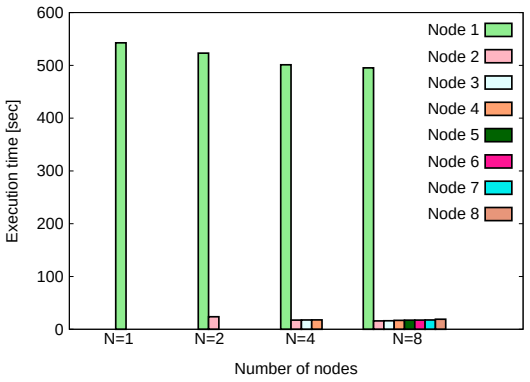

図 4 各ノードの rsync の実行時間
Fig. 4 rsync execution time on each node

(3) 全ファイルのリストを作成し，ファイルパスでソートする．
(4) ソートされたファイル・リストを N 個に分割する．
(5) ホスト OS から N 個の異なる仮想マシンに対して，N 個の rsync プロセスにより並列にファイルをコピーする．各 rsync プロセスは，(4) で作成した分割ファイル・リスト上のファイルを転送する．
(6) 全てのファイル転送プロセスの実行時間を計測する．

図 4 に，ファイル転送に使用するノードの数 N を 1，2，4，8 とした場合の，各ノードの rsync プロセスの実行時間を示す．あるディレクトリにサイズの大きいファイルが集中する場合は，特定の rsync プロセスにファイル転送処理の負荷が集中することが確認できた．また，このときに，ノードの数を増やしてもファイルの転送速度が向上しないことも確認できた．このような環境でファイル転送速度を向上させるためには，各 rsync プロセスが転送するファイルサイズを均等にする必要がある．

3. 提 案 手 法

以上から，スケーラブルな非同期コピーを実装するために，我々は以下の二つの問題を同時に解決する必要があった．

- 複数のノードがコピー先ストレージの同一ディレクトリ下にファイルを追加する際に発生する，ディレクトリ競合
- 特定の rsync プロセスにサイズの大きなファイル群の転送処理が集中する問題

上記の 2 つの問題を同時に解決するために，我々は，ディレクトリの更新処理は，追加するファイルのサイズに依存しない，という点に着目した．比較的サイズの大きいファイルを転送した場合，コピー先ストレージではディレクトリの更新処理に対してファイルデータの書き込み処理が多く発生することが予想される．すなわち，複数のノードがサイズの大きなファイルをコピー先ストレージの同一ディレクトリ下に転送した場合，ディレクトリ競合による書き込み遅延に対してファイルデータの書き込み処理に要する時間が十分に大きく，書き込み遅延は無視できると考えられる．

3.1 排他制御による書き込み遅延のモデル化

ディレクトリ競合による書き込み遅延を無視できるファイルサイズを算出するために，ファイルサイズから書き込み遅延を計算するモデルを導出した．

ディレクトリ競合が発生している環境でファイルを 1 つ作成するのに要する時間（以下，競合あり時間）を t_c，ディレクトリ競合の発生していない環境でファイルを 1 つ作成するのに要する時間（以下，競合なし時間）を t_n とする．また，競合あり時間に占める書き込み遅延の割合 P_d を，

$$P_d = \frac{t_c - t_n}{t_c} \tag{1}$$

と定義する．

次に，競合あり時間と競合なし時間を導出する．2.2 節で説明したように，ディレクトリ競合は，ディレクトリに対する更新権を保持していないノードが，そのディレクトリにファイルを追加する際に発生する．あるディレクトリに対する更新権を保持するノードはクラスタ内にただ一つ存在するため，あるノードがディレクトリにファイルを新規作成する場合，ディレクトリ競合が発生する確率は $\frac{N-1}{N}$ とおける．ただし，ノードの数を N とした．さらに，ディレクトリ競合が発生すると，あるノードによるディレクトリの更新がディスクに書き込まれることになる．以上から，ディレクトリの更新をディスクへ書き込むのに要する時間を t_{dir}，ファイルを 1 つ新規に作成する時間を t_{cr} とすると，競合あり時間と競合なし時間はそれぞれ，

$$t_c = \frac{N-1}{N} t_{dir} + t_{cr} \tag{2}$$

$$t_n = t_{cr} \tag{3}$$

とおける．式 (1)，(2)，(3) より，競合あり時間に占める書き込み遅延の割合は，

$$P_d = \frac{t_{dir}}{t_{dir} + \frac{N}{N-1} t_{cr}} \tag{4}$$

と表すことができる．

GPFS ではディスクへの最小の書き込みの単位をブ

ロックと呼び，ディレクトリを更新するとブロックサイズと等しい量のデータがディスクへ書き込まれる．よって，ブロックサイズを S_b，ディレクトリの更新によるディスクへの書き込み速度を W_d，ファイルサイズを S_f，ファイルの書き込み速度を W_f とすると，$t_{dir} = S_b/W_d$，$t_{cr} = S_f/W_f$ となる．これらを式（4）に代入すると，書き込み遅延の割合は

$$P_d = \frac{1}{1 + \frac{N}{N-1}\frac{W_d}{W_f}\frac{S_f}{S_b}} \qquad (5)$$

とおくことができる．ブロックサイズ S_b はファイルシステム作成時に決まる定数であり，ディレクトリの更新によるディスクへの書き込み速度 W_d はシステムの構成ごとにほぼ一定の値となる．また，コピー先ストレージの各ノードには十分な大きさのページキャッシュ領域が用意されており，ディレクトリ競合が発生しない場合，GPFS にファイルを新規作成してもディスクアクセスが発生しないと仮定する．例えば，SONAS の場合，各ノードに最大 144GB のメモリを搭載可能であり[9]，そのうち 3 分の 1 が GPFS のページキャッシュ領域に割り当てられる．このように仮定すると，ファイルの書き込み速度 W_f はページキャッシュへの書き込み速度となり，この値もシステムの構成ごとにほぼ一定の値となる．すなわち，ノード数 N を固定すると，ファイルサイズ S_f の増加に伴い，書き込み遅延の割合 P_d は単調に減少する．

このようにモデル化した書き込み遅延の割合 P_d の正しさを，実測値と比較することによって確かめた．

まず，2.2 節と 2.3 節で用いた環境で，実際にディレクトリ競合による書き込み遅延の割合 P_d を測定した．測定方法は以下の通りである．

（1） GPFS にディレクトリを N 個用意する．
（2） N 個のノードが，それぞれ異なるディレクトリにファイルを 1 つ作成するのに要する時間を測定する．この値が，競合なし時間 t_n となる．
（3） N 個のノードが，同一ディレクトリにファイルを 1 つ作成するのに要する時間を測定する．この値が，競合あり時間 t_c となる．

それぞれのケースで 1 ノードあたり合計 250MB となるように複数のファイルを作成し，作成に要した時間をファイルの数で割ることで，ファイル 1 つあたりの測定時間を求めた．$N = 2, 4, 8$ の場合に，作成するファイルのサイズを 128kB，256kB，512kB，1MB，2MB，4MB，8MB，10MB，20MB と変えて，測定値から式（1）を用いて書き込み遅延の割合 P_d を算出した．

次いで，この実験環境における，式（5）の定数を測定した．まず，GPFS のあるディレクトリに空のファイルを作り，更新内容をディスクへ書き込むプログラムを作成し，実行時間を測定した．ブロックサイズ $S_b = 256$kB をプログラムの実行時間で割り，$W_d = 86.74$MB/s を得た．次いで，Linux の dd コマンドを用いて，ファイルの書き込み速度を測定したところ，$W_f = 583.4$MB/s であった．

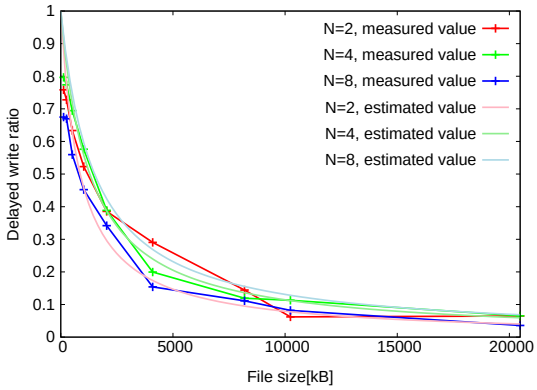

図 5 ファイルサイズに対する書き込み遅延の割合の，実測値とモデル値の比較
Fig. 5 Comparison between actual measured value and estimated value of overhead with respect to file size

図 5 に，式（5）から算出した予測値 P_d と，実測値から求めた P_d を示す．実測値はモデルによる予測値と同様にファイルサイズ S_f の増加に伴い単調に減少し，予測値から大きく外れてしまうことがない，ということを確認できた．また，一定以上のファイルサイズでは P_d の値，すなわち転送速度に占める書き込み遅延の割合の変化は小さく，ほぼ一定となることも確認できた．そこで，仮に書き込み遅延の割合が 5% 以下の場合に遅延が無視できるとすると，この実験環境では，ノード数 N の場合に $P_d = 0.05$ となるファイルサイズ S_T^N は，式（5）から $S_T^2 = 16.3$MB，$S_T^4 = 24.5$MB，$S_T^8 = 28.6$MB であった．

3.2 提案手法の流れ

提案手法による非同期コピーの処理の流れは以下のようになり，これを模式化したものを**図 6** に示す．非同期コピーに使用するノード数 N は，事前に設定されているとする．

（1） コピー元ストレージにおいて，GPFS のスキャン機能を用い，ファイル・リストを作成する．このとき，ファイル・リストには，ファイルの絶対パスとファイル・サイズが記載される．
（2） 排他制御による書き込み遅延が無視できる，遅延の割合 P_d の値を事前に設定しておく．コピー

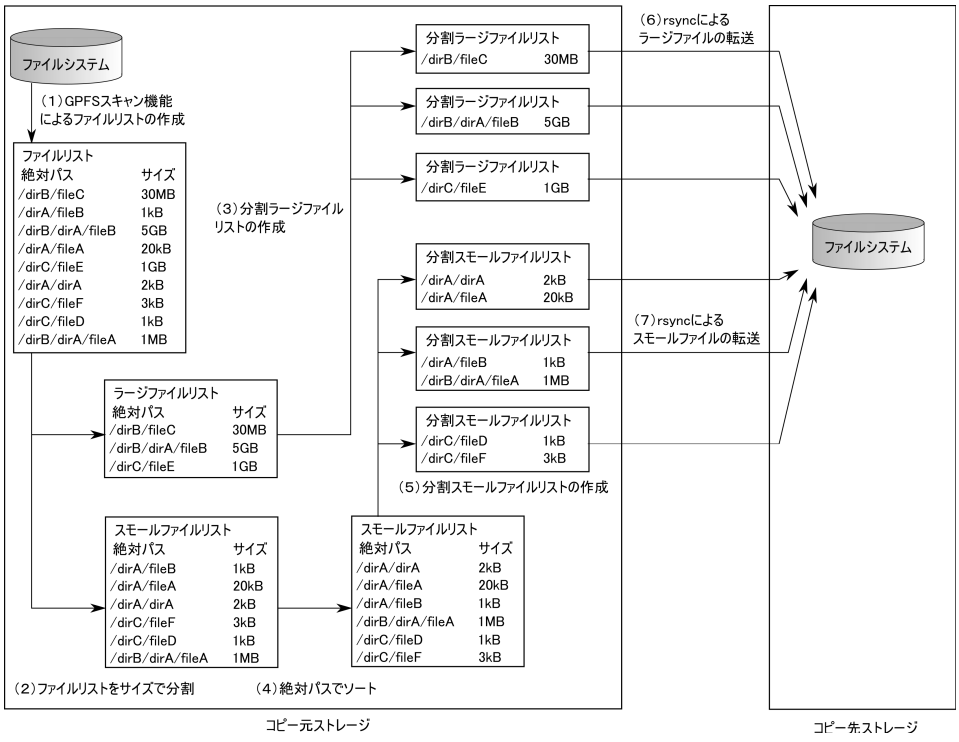

図 6 提案手法の流れ
Fig. 6 Flow of our proposed method

先ストレージにおいて，ブロックサイズ S_b，ファイルの書き込み速度 W_f，およびディレクトリの更新によるディスクへの書き込み速度 W_d を測定し，ディレクトリ競合による書き込み遅延を無視できるファイルサイズの閾値 S_T^N を計算する．作成されたファイル・リストを一行ずつ読み，サイズがある閾値 S_T^N 以上のものをラージ・ファイル・リスト，閾値 S_T^N 以下のものをスモール・ファイル・リストに記載する．

(3) ラージ・ファイル・リストに記載されたファイルを上から順に，各 rsync プロセスに配るようにして分割ラージ・ファイル・リストに記載する．分割ラージ・ファイル・リストに記載されたファイル・サイズの合計がある閾値以上になった場合，新しい分割ラージ・ファイル・リストを作成し，そこに記載する．(図 6 の場合は 30MB, 5GB, および 1GB のファイルの転送を異なる rsync プロセスが担当する．SONAS の非同期コピーはファイル単位であるため，ファイルを分割して rsync プロセス間で負荷を分散することはできない．)

(4) スモール・ファイル・リストを絶対パスでソートする．

(5) ソートされたスモール・ファイル・リストを，起動する rsync プロセスの数で分割する．

(6) 各 rsync プロセスが，分割ラージ・ファイル・リストに記載されたファイルを順にネットワーク転送する．

(7) 各 rsync プロセスが，分割スモール・ファイル・リストに記載されたファイルを順にネットワーク転送する．

(3) で分割ラージ・ファイル・リストのサイズに上限を設けることで，rsync プロセスの負荷を制限することができる．また，分割スモール・ファイル・リストには，同一ディレクトリ下のファイル群が記載される．以上により，サイズの大きいファイルについては負荷分散が，サイズの小さいファイルについてはディレクトリ競合の解消が実現される．

4. 評　　価

4.1 提案手法の効果

提案手法により，ファイル転送速度がノード数の増加に伴い改善する度合いを測定した．まず，サイズの大きいファイルが同一ディレクトリに集中する，という環境を再現するために，GPFS の同一ディレクトリに 1GB のファイルを 16 個作成した．次に，同一ディレ

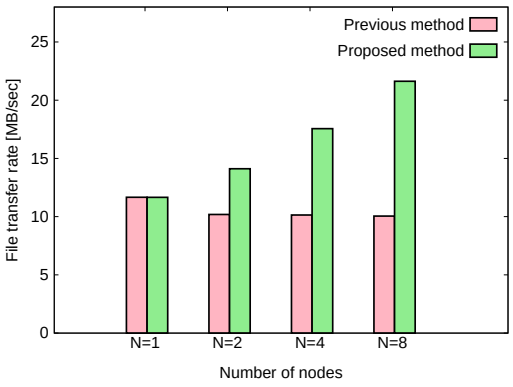

図 7 $P(90, 5, 3, 2, 0)$ の場合の，ファイル転送速度の比較
Fig. 7 Comparison of file transfer throughput for $P(90, 5, 3, 2, 0)$

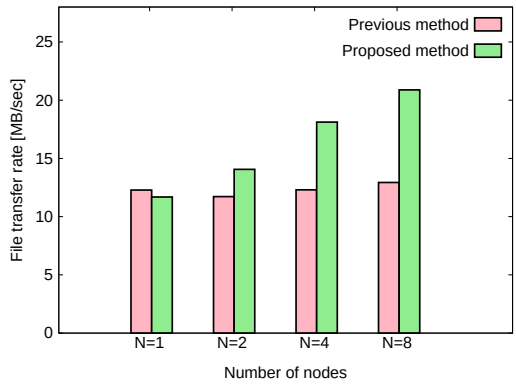

図 8 $P(75, 10, 10, 5, 0)$ の場合の，ファイル転送速度の比較
Fig. 8 Comparison of file transfer throughput for $P(75, 10, 10, 5, 0)$

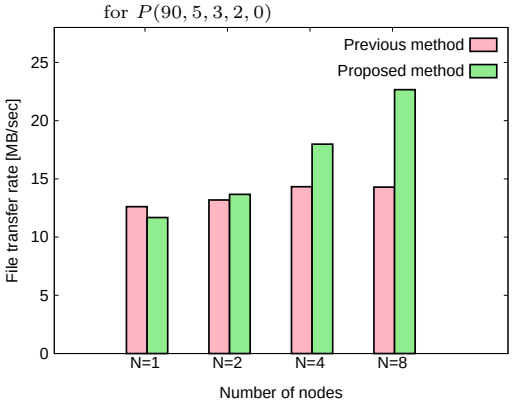

図 9 $P(55, 20, 10, 10, 5)$ の場合の，ファイル転送速度の比較
Fig. 9 Comparison of file transfer throughput for $P(55, 20, 10, 10, 5)$

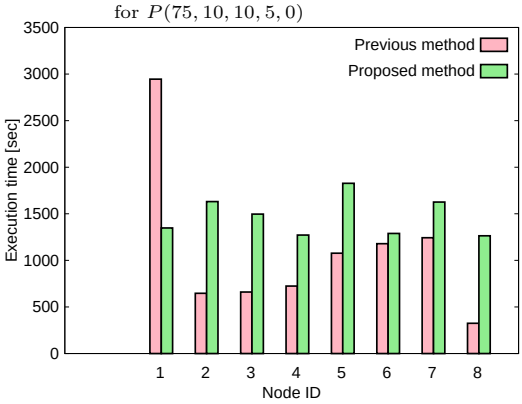

図 10 $P(75, 10, 10, 5, 0)$ の場合の，各ノードの rsync 実行時間
Fig. 10 rsync exection time on each node for $P(75, 10, 10, 5, 0)$

クトリに集中しないファイルツリーを生成するために，1kB-1MB のサイズのファイルを A% ，1MB-10MB のファイルを B% ，10MB-100MB のファイルを C% ，100MB-1GB のファイルを D% ，1GB のファイルを E% の確率で作成するプログラム $P(A, B, C, D, E)$ を用意した．様々なファイルツリー構成で提案手法を評価するために，このプログラムを用いて，A-E の値を変えて，各サブディレクトリにファイルが 4 つ，ディレクトリが 3 つ存在する深さが 4 のファイルツリーを作成した．ファイルサイズの上限を 1GB としたのは，実験環境の容量の制限のためである．2.2 節で用いた実験環境に SONAS の非同期コピープログラムをインストールし，ノードの数 N を変えて，本稿の提案手法と従来手法[18] のファイル転送速度を比較した．比較対象の従来手法として我々の先行研究を選んだ理由は，他のスケールアウト NAS の非同期コピーのファイル転送方式が明らかとなっていないためである．ファイル転送は，10ms の遅延を設定した仮想ネットワークを経由して，同じ GPFS クラスタのあるファイルシステムから別のファイルシステムに対して行った．

結果を，**図 7**，**図 8**，および**図 9** に示す．いずれのファイルツリー構成でも，従来手法はノードの数を増やしても転送速度がまったく向上しないか，向上の変化が緩やかなのに対し，提案手法は同一ディレクトリに保存された 1GB のファイル群の転送処理をノード間で分散したため，ノードの数を増やすことによって転送速度も向上した．特に，$P(90, 5, 3, 2, 0)$ のファイルツリーを転送した場合，非同期コピーに使用するノードの数を 1 つから 8 つに増やすことで，85% の改善効果を得た．

また，3 つのファイルツリー構成の中で，転送速度の改善率が平均的であった $P(75, 10, 10, 5, 0)$ の構成で，ノードを 8 つ使って非同期コピーを実行した場合の，各ノードの rsync プロセスの実行時間を計測したところ，**図 10** のようになった．提案手法を適用する

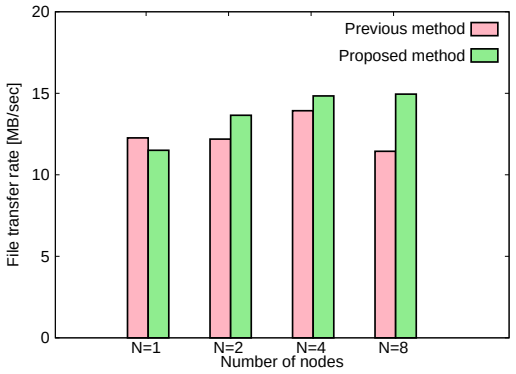

図 11 サイズの大きいファイルが同一ディレクトリに集中せず，かつ $P(75, 10, 10, 5, 0)$ の場合の，ファイル転送時間の比較
Fig. 11 Comparison of file transfer throughput for $P(75, 10, 10, 5, 0)$ and when large files do not exist in the same directory

ことで，あるノードに集中していた転送処理の負荷が分散されたことがわかった．

4.2 サイズの大きいファイルが同一ディレクトリに集中しない場合

提案手法による改善が期待できないケースを明らかにするために，4.1 節で作成した $P(90, 5, 3, 2, 0)$ のファイルツリーから 1GB のファイルを 16 個含んだディレクトリを削除し，非同期コピーを実行してファイルの転送速度を計測した．結果は**図 11** のようになった．サイズの大きいファイルがあるディレクトリに集中しないため，提案手法と従来手法に大きな差が見られなかった．ただし，同じ種類のファイルを同一ディレクトに保存するファイルツリー構成は一般的であるため，仮想マシンのイメージファイルやデータベースファイルなど，サイズの大きいファイルが同一ディレクトリに集中する可能性が高く，提案手法による改善の効果が期待できる．

5. 関 連 研 究

分散ファイルシステムでは，ファイルを複数のブロックに分割し，複数のノードに分散させて並列に読み書きを行うため，サイズの大きなファイルの操作がスケーラブルに行える，という特徴を持つ．これに対し，サイズの小さいファイルの読み書きやディレクトリ更新などのメタデータ操作は分散の効果が現れず，スケーラブルに行えない，という課題がある．これまでも，分散ファイルシステムに対するメタデータ操作方法などの設計を見直し，サイズに依存せずスケーラブルにファイル操作を行う方法が提案されてきた．

一つ目は，分散ファイルシステムのメタデータ処理を効率化する方法である．Carns らは，Parallel Virtual File System (PVFS) のメタデータ操作の内部で行われる手続きを効率化し，サイズの大きいファイルの書き込みや読み込み速度に影響を与えることなく，サイズの小さなファイルの処理速度の改善に成功した[1]．

二つ目は，サイズの小さい複数のファイルをサイズの大きい一つのファイルに変換して書き込むことで，書き込みバイト数あたりのメタデータ操作の回数を減らす方法である．Dong らは，Hadoop Distributed File System(HDFS) へ上記の変更を加え，サイズの小さいファイル操作をスケーラブルに行う方法を提案した[3]．また，この他にも，いくつかの同様な手法が提案されてきた[10),17)]．

三つ目は，書き込み時に分散ファイルシステムの一貫性を保証せず，読み込み時に保証する方法である．Patil らは，ディレクトリを複数のブロックに分割し，複数のノードが並列に同一ディレクトリを更新する際，各ノードが異なるブロックを独立に更新し，ディレクトリの読み込みの際に整合性を取る手法を提案した[11),12)]．

四つ目は，複数のノードで同一オブジェクトに対するメタデータ操作を衝突させない方法である．Weil らは，ファイルシステムを複数のサブツリーに分割し，同一ノードが同一サブツリーに対するメタデータ操作を行う方法を提案した[15)]．

これらの手法はいずれも，新たな分散ファイルシステムの設計方法を提案しているため，ファイルシステムを入れ替えた場合，その都度これらの設計を取り入れる必要がある．どのようなファイルシステムに対してもスケーラブルなファイル操作を行うためには，ファイル操作を行う側に修正を加えるほうが容易である．

我々の先行研究では，GPFS を導入したシステムでスケーラブルに稼動する非同期コピーを実現するために，ファイルを同一ディレクトリごとに集め，同一ディレクトリ下のファイルは同一ノードから転送する修正を非同期コピーに実装した[18)]．しかし，この方法はサイズの大きいファイルが同一ディレクトリに集中する場合，ノードの数を増やしても転送速度が向上しないという問題があった．別の先行研究[19)] では，本稿と同様にファイルサイズの大小で非同期コピーのファイル転送方式を切り替える方法を提案したが，ファイルサイズの閾値を経験的に決めており，ノードの数などの構成が異なる環境には適用できない可能性があった．また，非同期コピーのファイル転送速度の改善に注力しており，ノード数に応じて転送速度が向上するか検証は行われていなかった．

我々は，分散ファイルシステムへのファイルの書き方を工夫することによって，どのような分散ファイルシステムに対しても，スケーラブルに稼動する非同期コピーを実現できた．また，我々の先行研究[19]と異なり，ファイル転送速度に占める書き込み遅延の割合をモデル化し，転送方式を切り替えるファイルサイズを算出する方法を提案した．さらに，ノード数に応じてファイル転送速度が向上することも検証した．これまで，分散ファイルシステムの設計方法の改善によりスケーラブルなメタデータ操作を実現する方法は数多く提案されているが，分散ファイルシステムへの書き方を工夫したスケーラブルな機能を提案している事例は，我々の知る限り存在しない．

6. 結　　論

本稿では，スケーラブルに稼動する非同期コピーを実現するために，サイズの大きいファイル群とサイズの小さいファイル群の転送方式を切り替える方法を提案した．8 台の仮想マシンから構成した GPFS クラスタによる実験から，提案手法は従来手法と比較して，ノードを追加することにより転送速度が改善することを確認できた．あるファイルツリー構成では提案手法による改善の効果は見られなかったが，同じ種類のファイルを同一ディレクトリに保存するファイルツリー構成は一般的であるため，仮想マシンのイメージファイルやデータベースファイルなどのサイズの大きいファイルがあるディレクトリに集中する可能性が高く，提案手法による改善の効果が期待できる．また，本手法は GPFS に限らず，ディレクトリ競合による書き込み遅延が発生する分散ファイルシステム対して効果がある．

本稿では，ファイルの転送速度に占める書き込み遅延の割合が 5% 以下の場合，書き込み遅延の影響が無視できる，と仮定してファイルサイズの閾値を算出し，検証を行った．今後は，非同期コピーの全処理を効率化するために，ファイル転送処理，およびその前処理を合わせて最適化するファイルサイズの閾値を算出する方法について検討予定である．

参 考 文 献

1) Carns, P., Lang, S., Ross, R., Vilayannur, M., Kunkel, J. and Ludwig, T.: Small-File Access in Parallel File Systems, *Proceedings of the 23rd International Symposium on Parallel&Distributed Processing*, IEEE Computer Society, pp. 1–11 (2009).
2) Davidson, W.: rsync. `http://rsync.samba.org/`.
3) Dong, B., Qiu, J., Zheng, Q., Zhong, X., Li, J. and Li, Y.: A Novel Approach to Improving the Efficiency of Storing and Accessing Small Files on Hadoop: a Case Study by PowerPoint Files, *Proceedings of the 7th International Conference on Services Computing*, IEEE Computer Society, pp. 65–72 (2010).
4) EMC: Isilon Scale-out Network Attached Storage (NAS) for Big Data. `http://www.emc.com/domains/isilon/index.htm`.
5) Hitachi: HDS: Hitachi Network Attached Storage (HNAS) Platform — NAS System Solutions. `http://www.hds.com/products/file-and-content/network-attached-storage/`.
6) Hwang, J., Zeng, S., Wu, F. and Wood, T.: A Component-Based Performance Comparison of Four Hypervisors, *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management*, IEEE Computer Society, pp. 269–276 (2013).
7) IBM: SONAS Concepts, Architecture, and Planning Guide (2012). `http://www.redbooks.ibm.com/redbooks/pdfs/sg247963.pdf`.
8) IBM: SONAS Information Center (2013). `http://pic.dhe.ibm.com/infocenter/sonasic/sonas1ic/topic/com.ibm.sonas.doc/pln_f2c_featurecode_racks_rxa.html`.
9) IBM: SONAS Information Center (2013). `http://pic.dhe.ibm.com/infocenter/sonasic/sonas1ic/topic/com.ibm.sonas.doc/pln_f2c_featurecode_nodes_si2.html`.
10) Liu, X., Han, J., Zhong, Y., Han, C. and He, X.: Implementing WebGIS on Hadoop: A Case Study of Improving Small File I/O Performance on HDFS, *Proceedings of the International Conference on Cluster Computing and Workshops*, IEEE Computer Society, pp. 1–8 (2009).
11) Patil, S. and Gibson, G.: GIGA+ : Scalable Directories for Shared File System, Technical report, Parallel Data Laboratory, Carnegie Mellon University (2008).
12) Patil, S. and Gibson, G.: Scale and Concurrency of GIGA+: File System Directories with Millions of Files, *Proceedings of the 9th USENIX Conference on File and Storage Technologies*, USENIX Association, pp. 177–190 (2011).
13) RedHat: Red Hat Global File System. `http://www.redhat.com/whitepapers/rha/gfs/GFS_INS0032US.pdf`.
14) Schmuck, F. and Haskin, R.: GPFS: A Shared-Disk File System for Large Computing Clus-

ters, *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, USENIX Association (2002).
15) Weil, S. A., Brandt, S. A., Miller, E. L., Long, D. D. E. and Maltzahn, C.: Ceph: A Scalable, High-Performance Distributed File System, *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, USENIX Association, pp. 307–320 (2006).
16) Xing, J., Xiong, J., Sun, N. and Ma, J.: Adaptive and Scalable Metadata Management to Support a Trillion Files, *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, ACM Press (2009).
17) Zhang, Y. and Liu, D.: Improving the Efficiency of Storing for Small Files in HDFS, *Proceedings of the International Conference on Computer Science and Service System*, IEEE Computer Society, pp. 2239–2242 (2012).
18) 三好浩之, 萩原克彦, 松井壮介, 岩崎礼江: SONAS 非同期コピーのパフォーマンス改善, *PROVISION*, No. 70, pp. 90–96 (2011).
19) 松井壮介, 三好浩之, 高井聡, 荒木博志: rsync プロセスの負荷分散による SONAS 非同期コピーのパフォーマンス改善, *PROVISION*, No. 77, pp. 73–79 (2013).