スケーラブルな広域ファイルシステムの研究

研究代表者　建部 修見　筑波大学大学院・システム情報工学研究科・准教授
研究協力者　中村 昌弘　筑波大学大学院・システム情報工学研究科・M2
神林 亮　筑波大学大学院・システム情報工学研究科・M2
平賀 弘平　筑波大学大学院・システム情報工学研究科・M1
鈴木 克典　筑波大学大学院・システム情報工学研究科・M1
木村 浩希　筑波大学・第三学群情報学類・B4
小林 賢司　筑波大学・第三学群情報学類・B4
三上 俊輔　筑波大学・第三学群情報学類・B4

1　研究の概要

研究目的　増加の一途のデータの解析に対し、複数の拠点による広域分散データ解析の要求が高まっている。そのためには広域ファイルシステムが有用であるが、データ解析の規模が拡大するにつれ、ファイルを管理するメタデータのアクセス負荷の増大、広域ネットワークの遅延による応答性の低下が問題となっている。本研究では、スケーラブルな広域ファイルシステムを目指し、メタデータの広域分散管理に関する研究、ファイル複製の効率的な作成に関する研究、効率的な広域データ処理に関する研究を行う。広域分散データ解析が必要とする性能を達成するためのシステムアーキテクチャの検討、従来手法との比較によるアルゴリズムの検討を実施する。これにより、これまでデータアクセス性能の不足により実施不能であったような規模のデータ解析も可能とすることを目的としている。

本年度の研究成果の概要　本年度は、下記の課題に取り組んだ。

- 分散メタデータサーバ
- クラスタ間高速ファイル転送

分散メタデータサーバの研究として、本年度分散メタデータサーバの検討を行った。結果整合性を保つマルチマスタ構成の分散メタデータサーバのアーキテクチャ設計し、初期性能評価を行った。

クラスタ間高速ファイル転送の研究として、本年度はサイト A に保持している多数のファイルをサイト B に効率的に転送するためのスケジューリングの課題に取組、転送ファイルスケジューリングアルゴリズム、複製選択アルゴリズム、転送前動的ファイル複製作成アルゴリズムを提案し性能評価を行った。

なお、今年度は、下記の体制で研究を推進した。

2　分散メタデータサーバの研究

近年コンピュータの高性能化や価格低下により、様々な組織で PC クラスタを構築し、それらを利用して大規模な計算処理を行う事が容易となっている。それに伴い、計算処理で扱うデータ量は年々増加している。素粒子物理学、天文学、生命科学などの科学技術分野におけるデータインテンシブコンピューティング分野では、広域に散在する計算資源、記憶装置、観測装置を効率的に扱い、組織間の壁を超えて有益なデータの共有を行うグリッド関連技術の発展と標準化が進んでいる。地理的に分散した PC クラスタ間でデータを効率的に共有するためには、Gfarm[1] などの広域ファイルシステムの利用が注目されている。

ファイルシステムのメタデータへの操作は、ファイルシステム全体のワークロードの多くを占めている[3]。分散ファイルシステムでは、メタデータ操作がシステムのボトルネックにならないようにするための提案がこれまでなされているが、広域環境では Round Trip Time の長い高遅延通信路が存在するため、そのままでは適用できない。例えば、日米間の RTT は 130~250ms、日欧間は 250~300ms なので、広域ファイルシステム操作に対するリクエスト応答時間は、高遅延ネットワークを介した通信が何往復するかによって大部分が決定するからである。

我々はこれまでに、RTT の長い通信路に依存しない、Key-Value の連想配列をベースにメタデータを管理する手法について研究を行ってきた[4]。本研究では、これをベースにデータ構造を一部拡張し、実際のファイルシステムのメタデータを管理するシステム、分散メタデータ管理サーバを設計する。

2.1 設計の概要

ファイルシステムのファイルやディレクトリに関するメタデータは、inode と呼ばれる構造で管理する。ファイルシステムメタデータの操作では、複数の inode エントリをアトミックに更新する必要があり、それを可能とする仕組を提案する。サーバ間の inode の一貫性は結果整合性 (Eventual Consistency) [2] を保証する。inode への更新が起こると、他のサーバにバックグラウンドで非同期に更新が伝播し、自動的に同期する。クライアントは、拠点内のメタデータサーバに対して自由に inode の更新が可能である。メタデータサーバは、自身の持つ inode への操作が完了したら、他拠点のメタデータサーバと通信を行うことなく、直ちにクライアントに応答を返す。

2.2 システム構成

各拠点にメタデータサーバを配置し、各メタデータサーバはシステムが管理する全メタデータの複製を保持する。

クライアントはメタデータ操作の為の API 呼出しを通じて、自拠点内にあるメタデータサーバに対して Read/Write リクエストを発行する。クライアントのリクエストは低遅延な LAN を介して処理される。各メタデータサーバはすべてのメタデータの複製を保持しているので、メタデータサーバはクライアントに即座に応答を返すことができる。メタデータサーバはマルチマスタ型の構成を採用し、複数クライアントからの Read/Write 操作の並列性を確保する。

2.3 サーバ間のメタデーター貫性

前節で示したようなマルチマスタ型のシステム構成の場合、サーバ間の一貫性を維持するためには、分散ロックや 2-phase commit 等を利用した複雑でオーバヘッドの大きいプロトコルの設計を行う必要がある。広域環境においては、他拠点との通信遅延が長いため、一貫性を維持するためのプロトコルは致命的な応答時間の増大に繋がってしまう。

提案システムでは、一貫性モデルとして結果整合性を保証する。結果整合性は制約の緩い一貫性で、常に最新のデータにアクセスできるとは限らないが、最終的には最新のデータにアクセスできるようになる事を保証する。結果整合性を採用した理由は、前述の通り、高遅延通信路を含む環境下で厳密な一貫性を実現すると、書き込みが集中したり、2-phase commit を実装することが避けられないからである。前者は書き込みのスケールアウトが期待できなくなる。後者は、広域通信路で行ったときのオーバヘッドが大きすぎるという問題がある。一方、広域ファイルシステムにおいては、ファイルシステムの厳密な一貫性は必ずしも要求されないことが多い。従って本システムでは、厳密な一貫性を保つことで、ファイルシステム全体の性能が低下するより、むしろパフォーマンスを出すことを優先する。もし厳密な一貫性が必要な場合は、メタデータ同期操作、ロックサービスの利用等で対応可能と考えれれる。これらの理由から、結果整合性を選択する。

メタデータサーバは、ファイルやディレクトリ等のファイルシステム上のオブジェクトに関する情報を管理する。本システムでは、これらのメタデータを inode と呼ばれるデータ構造で管理する。inode は Unix

系ファイルシステムで使われているデータ構造で、各オブジェクトに関する情報が格納される。inode エントリは、inode 番号と呼ばれるユニークな識別子で参照される。ディレクトリに関する inode エントリには、そのディレクトリに含まれるオブジェクトの inode 番号と名前のペアが含まれる。クライアントは、inode エントリをルートディレクトリから再帰的に辿ることで、目的のオブジェクトに到達する。

分散配置された各メタデータサーバは、この inode 番号とオブジェクトメタデータのペアからなる Key-Value の連想配列をそれぞれローカルに保持し、管理する。inode に対する Write/Read 操作リクエストは、すべてのメタデータサーバで並行に発行される。各々のサーバの inode エントリに対する更新は、他のサーバに非同期に伝播され、自動的に同期する。他のサーバに更新が伝播するよりも早く、他拠点のクライアントによって inode エントリに対する更新が起こった場合、inode エントリへの更新が衝突する場合がある。したがって、お互いが持っている Key-Value の連想配列を Eventually に同期し、一貫性を維持するプロトコルが必要となる。更新が衝突した inode エントリが含まれる場合は、何らかのマージルールで衝突を解決し、最終的にはすべてのメタデータサーバが同じ inode テーブルを持つ状態に収束しなければならない。衝突の解決にあたり、ディレクトリツリーの木構造が崩れて、ルート inode エントリから辿り着けない inode エントリが現れないようにする必要がある。

2.4 ユニークな inode 番号の生成

ファイルシステムが新しい inode 番号の生成を必要とした時は、ユニークな番号を生成する必要がある。分散システムにおいて、あるコンポーネントがシステム全体でユニークとなるような番号を生成するには、他のコンポーネントが自分と同じ番号を生成しない事を保証すれば良い。そこで、分散システムの各コンポーネントには、起動時にあらかじめシステム全体でユニークとなる識別子（例えば FQDN 等）を割り当てる。新しい inode 番号を生成するには、自身の識別子と、コンポーネントが各々管理する非負の整数の対によって、システム全体でユニークとなるよう番号を生成する。

2.5 Vector Clock を用いた inode のバージョン管理

結果整合性を採用している提案システムでは、同時に 2 つ以上のクライアントから inode の更新操作が行われた場合、更新が衝突する可能性がある。マルチマスタ型で厳密な一貫性のための排他制御を実現するには、Write/Read 操作時に、広域に分散するメタデータサーバ間で分散ロックを取得する必要があるが、これを行わない。提案システムでは、分散ロックの代わりに Vector Clock [5] を用いる。Vector Clock は、それぞれのサーバ識別子とサーバローカルなカウンタのペアのベクトルで、半順序が定義される。メタデータサーバの inode の各エントリは、Vector Clock によってバージョニングされる。バージョニングによって、inode の更新操作の順序関係を保存し、順序が定義されなければ、更新操作により衝突を検知する。もし更新操作が衝突したら、一時的にメタデータサーバ間で inode に違いが起こるが、予め定義するマージアルゴリズムに従って、システムが自動的に衝突を解決し、いずれすべてのメタデータサーバ通知される。これにより、結果整合性を守る。

2.5.1 複数 Key のアトミックな更新操作

ファイルシステムへの操作によって、同時に複数の inode エントリが変更される例を以下に示す。

ファイルの作成

1. ユニークな inode 番号を生成し、ファイルを表す新しい inode エントリを作成
2. 親ディレクトリの inode エントリに、新しく作成した inode エントリへのリンクを追加

ファイルの移動

1. 移動先ディレクトリ inode エントリに、移動ファイルへのリンクを追加

2. 移動元ディレクトリ inode エントリから、移動ファイルへのリンクを削除

ファイルの削除

1. 親ディレクトリの inode エントリから、削除対象ファイル inode エントリへのリンクを削除
2. 削除対象ファイルの inode エントリに、削除フラグをセットする

上記の複数の inode の変更において、どちらかの変更が衝突してしまうと、ディレクトリの木構造が崩れてしまう。ディレクトリ木構造を崩さないよう inode を更新するには、inode の複数のエントリをアトミックに更新する必要がある。そこで、既存の分散 Key-Value ストアの実装 [4, 7] の、Vector Clock による単一 Key に対するオブジェクトバージョニングをベースとして、複数 Key にまたがるアトミックな更新時に、正しく更新の衝突の検知を行えるように拡張する。

基本的な方針は、アトミックな複数の変更のうち、どれか一つでも衝突していれば、全体が衝突したと検知する。inode エントリを更新する場合、どの inode 番号の、どのバージョンを更新するのかを指定する。複数の inode エントリをアトミックに更新する場合は、更新する inode エントリの inode 番号 (key) とエントリ内容 (value) のペアのリストと、その更新範囲の inode エントリの現在のバージョンをすべて反映した Vector Clock を指定する。メタデータサーバは更新リクエストを受け取ると、更新される全ての inode エントリの Vector Clock と、更新リクエストの Vector Clock を比較し、更新の衝突が起こっていないか確かめる。比較不能な Vector Clock が一組でも存在するならば、更新が衝突しているとみなし、更新範囲の inode エントリを全てブランチにする。衝突の解決は、衝突の起こった二つ以上のブランチのアトミックな更新範囲を全て読み込み、その範囲のバージョンを全て反映した Vector Clock を指定して、範囲全体を上書きするアトミック更新を行うことで解決する。

2.5.2 衝突解決サーバの選択と、inode の衝突解決アルゴリズム

更新の衝突が起こった場合、システムによる自動的な inode の衝突解決を行うが、更新の衝突を検知した際、各々のサーバが衝突解決を行うと、衝突解決のための更新がまた衝突してしまう問題がある。そのため、衝突解決を自動で行う担当サーバ (衝突解決サーバ) を一台選択する。衝突解決者の選出には、システム管理者がマニュアルオペレーションで設定する方法や、paxos アルゴリズム [6] の利用が考えられる。

inode の衝突解決アルゴリズムは従来研究で提案されている。ファイルの更新が衝突した場合は、ファイルを別の名前にリネームし、同じディレクトリに保存することで解決する。更新の衝突により親ディレクトリが失われてしまった場合には、親ディレクトリを復活させて解決する方法と、親ディレクトリを失ったエントリを (/orphan 等の) 特別なディレクトリに移動する方法が考えられる。衝突した複数の更新操作は、衝突解決サーバに到着した順番で直列化され、順番にマージ処理を行う。解決方法に複数選択肢がある場合は、ユーザが予め衝突解決ポリシーを設定する。

2.6 おわりに

本研究では、広域ファイルシステムのための分散メタデータサーバの設計について述べた。マルチマスタ型の分散システムで、一貫性は結果整合性を選択し、高遅延通信路の影響を受けないファイルシステムの inode 更新処理方法を提案した。複数の inode エントリをアトミックに更新する場合でも、更新の衝突検知と解決操作を行うプロトコルの設計を行った。今後の課題としては、システムを実装し、広域分散環境においてリクエスト処理性能を評価する事である。

3 クラスタ間高速ファイル転送の研究

ネットワークの広帯域化に伴い広域環境における大規模データ共有が現実のものとなってきた。一方で、大規模データ処理のために PC クラスタのそれぞれのローカルディスクが用いられるようになった。その結果、多数のローカルディスク間の効率的な並列ファイル転送が必要となっている。本研究では、並列ファ

イル転送を効率的に行うためのスケジューリング手法を提案する。ファイル複製の適切な選択、送信元クラスタ内での動的な複製作成により効率的な並列ファイル転送を可能とする。提案手法を実際の広域ネットワーク環境で評価した結果、1 ノードに偏って格納されている 50 GB のデータを 17 分 21 秒で転送することができた。これは同一の環境で 1 ノード対 1 ノードでファイル転送を行った場合の 86.1%の転送時間である。このとき、平均スループット 49.0 MB/sec、最高スループット 380 MB/sec の性能であった。

3.1 問題設定

本研究ではクラスタ内の複数のディスクに格納されるファイル群を効率的に他クラスタに転送するための機構について考察する。すべてのノードがそれぞれローカルなディスクを持ち、ファイルはそれらのディスクに格納されるようなクラスタ環境を想定する。ネットワーク環境として、クラスタ間の全体の転送バンド幅は、クラスタ間の特定のノード同士の転送バンド幅よりも大きいとする。また、クラスタ内のノード間のバンド幅はクラスタ間の特定ノード同士のバンド幅よりも大きいと仮定する。さらに，次のような状況を仮定する。

ディスクのシーク頻発による速度低下 あるハードディスクに対し同時に複数ファイルを読み書きを行った場合、シークが頻発しディスクアクセス時間が極端に長くなる。この問題はファイル転送性能に対し大きな影響を及ぼすにも関わらず、適切なスケジューリングによって回避可能な問題であり、ファイル転送スケジューリングを行う上で第一に守るべき制限として設定する。

ファイル配置の不均一性 特定のディスクに対し偏ってファイルが格納されている可能性がある。このとき、ファイル群転送において一部のディスクからの読み出し速度が全体の律速となると考えられ対策が必要となる。本手法では、転送時に動的にファイル複製を作成することでこの問題を解決する。

ファイル複製の存在 あるファイルに対し複製が複数のディスクに存在する場合、適切なファイル複製を選択することにより効率的なファイル転送が可能である。

3.2 並列ファイル転送スケジューリング

前節で述べた想定環境において各ノードのディスク上のファイル群を効率的に転送するために解決すべき問題としては以下の項目を挙げることができる。

ファイルの転送順序、転送タイミングの決定 輻輳の発生による転送速度の低下を回避するために各ノードが転送を開始するタイミングをスケジューリングしなければならない。

同時転送ノード数とデータ転送量の制御 同時転送ノード数と各ノードからの転送量を制御することで、ネットワークに流れるデータ量を制限し、輻輳の発生を抑制する。

ファイル複製の選択 適切なファイル複製の選択を行うことで特定のノードに負荷が偏らないよう考慮する。

動的なファイル複製作成のスケジューリング 全体としての転送時間を短縮するようにクラスタ内のファイル複製作成と外部への転送のスケジューリングを行わなければならない。

3.3 問題のモデル化

クラスタ間のネットワーク転送を安定に高速に行うため、広域転送に関してはトラフィックシェーピングを行うことが有効である。そのため、モデル化に際し、単一ファイルのクラスタ間の転送は固定バンド幅 c で行うこととし、このバンド幅 c の接続を「コネクション」とよぶ。クラスタ間のネットワークバンド幅を十二分に利用するためには、クラスタ間のバンド幅分、コネクションを複数同時に利用すればよい。このコネクションの集合を C で表す。

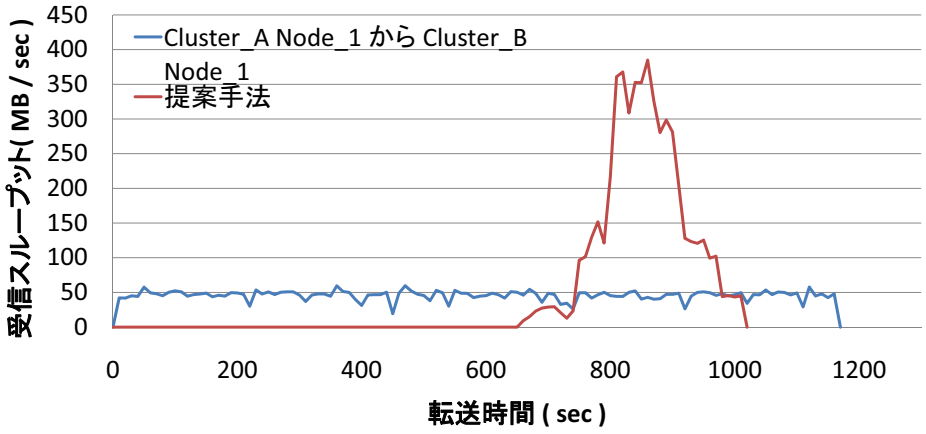

図 1: 1 対 1 転送及び提案手法のクラスタ B における受信スループット

ノードの集合を $N = \{N_1, N_2, \cdots, N_n\}$、ファイルの集合を $F = \{F_1, F_2, \cdots, F_m\}$ としたとき、解くべき問題はファイル複製の選択決定であるノード集合 N に対するファイル集合 F の組み合わせ問題と、ファイル転送順序のスケジューリングであるファイル集合 F とコネクション集合 C の割り当て問題となる。

ここで、転送処理に関して複数のファイルを一括して一つのデータとして扱っても問題はない。また、同様に一つのファイルを分割し、別々のデータとして転送しても問題はない。この特徴を利用するとすべてのノードが複数のファイルを持つ状態を、すべてのノードが一つの転送データを持つ状態であると見ることができる。また、全てのノードはファイル転送のタスクに関して互いに影響を与えず、独立であるといえる。これらの特徴からファイル集合 F とコネクション集合 C の割り当て問題は、転送に関して独立であるノードの集合 N とコネクション集合 C の独立タスクのスケジューリング問題と言い換えることができる。

なお、提案スケジューリングの詳細については著書、論文 2 を参照されたい。

3.4 評価

現在、システムとしては提案段階であるため、アルゴリズムについて C++言語で実装し、分散ファイルシステム Gfarm のファイル転送機能を用いて実環境上で評価実験を行った。

実験には産業技術総合研究所の PC クラスタ A の 7 ノードと筑波大学の PC クラスタ B の 7 ノードを用いた。クラスタ A, B のすべてのノードが Gigabit Ethernet で接続され、クラスタ間は 10 Gigabit Ethernet で接続されている。

このクラスタ A, B の間でファイル転送を行い提案手法を評価する。すべてのデータはクラスタ A の 1 ノードのローカルディスクに格納され、クラスタ A から B へファイル転送を行う。転送するデータサイズは 50 GB とする。

3.4.1 1 対 1 のファイル転送と提案手法の比較

提案手法と 1 対 1 のファイル転送を比較評価する。1 対 1 転送にはクラスタ A の 1 ノードからクラスタ B の 1 ノードへ、50GB 分のファイルを転送した。結果を図 1 に示す。1 対 1 の転送には 19 分 46 秒かかり、平均スループットは 42.2 MB/sec であった。一方、提案手法は 17 分 21 秒かかり、平均スループットは 49.0 MB/sec であった。以上の結果から、今回の実験ではクラスタ間でデータを転送する目的においては提案手法を用いることで 1 対 1 転送の 86.1%の時間で全データを転送することができたといえる。

3.5 まとめ

本研究では広帯域なネットワークで接続されるクラスタ間において、その広帯域ネットワークを有効に活用し高速にファイル転送を行う手法について提案し、評価した。提案手法は各ノードが並列にファイル転送を行うことを基本とし、さらに各ノードからの転送量制御、転送順序制御、動的な複製作成などにより効率的に転送を行う。

評価は分散ファイルシステム Gfarm のファイル転送機能を用い、実際に 2 クラスタ間でファイル転送実験を行った。結果として、50 GB のデータを 17 分 21 秒で転送し、これは 1 ノード対 1 ノードの場合の 86.1%の時間であった。また、最もディスク I/O の遅いノードが 54.8 MB/sec であるという環境のもと、平均スループット 49.0 MB/sec、最高スループット 380 MB/sec の性能を得た。

4 今後の展望

本研究では、ここで報告したように 2 つの課題について研究を行った。今後の展望は下記のとおりである。

- 分散メタデータサーバの研究では、今後実装をすすめ、広域分散ファイルシステムに組み込んでいきたい。このとき、広域ファイルシステムのメタデータ操作、ファイル操作に関してほぼローカルの共有ファイルシステムと変わらない性能が期待される一方で、バンド幅をスケールさせることが可能となる。今後、実装、評価をすすめ、実際の広域のデータインテンシブコンピューティングにおいて評価をしていきたい。
- クラスタ間高速ファイル転送の研究では、提案アルゴリズムを誰でも利用可能なかたちで提供し、複数拠点間でのファイル共有を促進していきたい。

参考文献

[1] *Gfarm Grid File System*. http://sourceforge.net/projects/gfarm/

[2] Vogels, W.: “Eventually Consistent - Revisited”, http://www.allthingsdistributed.com/2008/12/eventually_consistent.html

[3] Roselli, D., Lorch, R. J., and Anderson, T. E.: “A Comparison of File System Workloads”, Procs 2000 USENIX Annual Technical Conference, pp. 41–54, 2000.

[4] 平賀弘平, 建部修見: “広域ファイルシステムにおける分散メタデータサーバの検討”, 情報処理学会研究報告, 2009-HPC-119, pp. 139–144, 2009.

[5] Lamport, L.: “Time, clocks, and the ordering of events in a distributed system”, Commun. ACM, Vol. 21, No. 7, pp. 558–565, 1978.

[6] Lamport, L.: “Paxos Made Simple”, ACM SIGACT News, Vol. 32, No. 4, pp. 18–25, 2001.

[7] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall P., and Vogels W.: “Dynamo: amazon’s highly available key-value store”, SIGOPS Oper. Syst. Rev., Vol. 41, No. 6, pp. 205–220, 2007.

研究成果リスト

著書、論文

1. 平賀弘平、建部修見: “広域ファイルシステムのための分散メタデータサーバの設計”, 情報処理学会研究報告, 2009-HPC-121(32), pp. 1–7, 2009.

2. 鈴木克典、建部修見: “クラスタ間高速ファイル転送方式の提案と評価”, 情報処理学会研究報告, 2009-HPC-121(31), pp. 1–8, 2009.

3. 木村浩希、建部修見: “広域分散ファイルシステム Gfarm の MPI-IO の実装”, 情報処理学会研究報告, 2010-HPC-124, 2010 (to appear).

4. 小林賢司、建部修見: “クラウドコンピューティングにおける高性能ファイルシステムの検討”, 情報処理学会全国大会, 2010 (to appear).

5. 三上俊輔、建部修見: “MapReduce におけるファイルシステムの性能評価”, 情報処理学会全国大会, 2010 (to appear).

公開ソフトウェア

1. 建部修見: Gfarm ファイルシステム version 2.3.0

   http://sourceforge.net/projects/gfarm/

   大規模分散データ解析を支援する広域分散ファイルシステム.