平成23年度　　卒業研究論文

分散ファイルシステムCephにおけるI/O性能の検討

指導教員
松尾 啓志 教授
津邑 公暁 准教授

名古屋工業大学 工学部情報工学科
平成20年度入学　20115107番

中居 新太郎

目 次

1　はじめに

近年，ネットワークの高速化やインターネットの普及により大規模なデータ処理の需要が増加の一途を辿っており，ストレージに対する要求が高まっている．しかし，単一のサーバでストレージを構成した場合，拡張性に限界があり単一障害点になりうる恐れがある．これらの問題を解決するために分散ファイルシステムという技術が提案されている．一般的に分散ファイルシステムは複数のノードでストレージを構成し，ファイルシステムがファイルデータの配置情報や名前空間の管理等を行うために用いるファイルデータ本体以外の情報であるメタデータをストレージから分離し，単一のメタデータサーバで集中的に管理することで，ストレージへのアクセスの並行性を高めシステム全体の性能を向上させ，ノードの追加によって容易にストレージの容量を拡張することができる．また，データをレプリケーションし，物理的に距離のあるノード上にそれぞれを配置することで，ストレージの一部に障害が発生によってデータが失われることを防ぐ．

本研究では分散ファイルシステムの一種である Ceph ファイルシステムに着目した．Ceph ファイルシテムは，ストレージを数ペタバイトまで容易にスケーリングする拡張性と高い耐故障性を実現し，また様々ワークロードに適応して，動的に負荷を再分散することで常に高い性能を提供する．これらの機能を同時に実現するために動的に負荷とデータの分散を行う RADOS, 複数のメタデータサーバによる適応的なメタデータの分散管理を実現する動的サブツリーパーティショニングといった非常に興味深い概念を展開している．本稿では，Ceph ファイルシステムの様々なアクセスパターンにより I/O 性能を評価し，システム全体の性能を向上させるために I/O 操作における改良点を検討する．

第 2 章では研究背景を述べる．第 3 章で提案を述べ，第 4 章で評価について述べる．第 5 章で考察を述べ，第 6 章でまとめと今後の課題について述べる．

2　研究背景

本章では，本研究で取り扱う分散ファイルシステム Ceph について，設計の方針とアーキテクチャの構成，動作を概説する．

2.1　ファイルシステム

ファイルシステムはストレージを扱う上で重要な機能を提供する．ストレージはブロックの集合であり，ブロックアドレスでアドレッシングされる．任意のデータが配置されたブロックを判断するには，ブロックアドレスを記憶する必要がある．そこで，ファイルシステムはストレージをファイルという扱いやすい形式にして，ユーザやアプリケーションに提供する．したがって，ストレージへのアクセスはファイルシステムというレイヤを通して行われる．

ストレージの管理には，メタデータを用いる．メタデータとは，ファイルのデータ本体ではなく，データの配置情報やアクセス権限など，データ以外のファイルシステムが管理している情報のことである．ストレージ上へのアクセスにはメタデータの操作が必須であるため，効率的なメタデータ操作は，ファイルシステムの性能を向上する上で重要だと言える．

2.1.1　ローカルファイルシステム

ローカルファイルシステムとは，1 台の OS から占有されているディスクに構成するファイルシステムである．高速・信頼性・大容量なファイルシステムを実現するために様々なファイルシステムが開発されている．ファイルシステムをローカルディスクだけで構成する問題点が 2 つある．1 つ目はローカルディスク容量でファイルシステム容量が制限されてしまうことである．また，ローカルディスクはスロット数が限られるため，拡張にも制限がかかってしまう．2 つ目は複数ノードでファイル共有できないことである．これらの問題点を解決するための技術として，ネットワークファイルシステムが存在する．

2.1.2 ネットワークファイルシステム

ネットワークファイルシステムはクライアントサーバシステムであり，クライアントがネットワーク越しにサーバのローカルファイルシステムにアクセスする機構である．ファイルサーバや NAS（Network Attached Storage）といったファイル共有型のストレージシステムへのファイルアクセス手段として広く利用されている．

ネットワークファイルシステムには様々な実装があるが，一般的な共通点として，それぞれクライアントサーバ間でやりとりされるプロトコルを明確に規定しているため，そのプロトコルに準拠したクライアント，もしくはサーバであれば，動作環境に依存することなく，ファイルアクセスが可能である．このように，ネットワークシステムを用いることで，ネットさークに接続された他の計算機のファイルシステムを，ローカルファイルシステムと同じように利用することが可能となり，ローカルファイルシステムの容量とその拡張性の制限を解決する．また，複数のユーザ間でファイルを共有することも可能であり，同時にアクセスができることで利便性が大きく向上する．

2.1.3 分散ファイルシステム

ストレージに対する要求が高まり，ストレージシステムは数ペタバイト以上の容量が必要とされるようになった．また，扱うデータの量が増加していくため，それに伴ってストレージの柔軟な拡張が可能であることが望ましいと言える．しかし，代表的なネットワークファイルシステムである NFS（Network file system）や CIFS（Common Internet File System）は，基本的に単一のサーバで構成されており，拡張性に限界が生じてしまう．

この問題を解決するために，複数の計算機にファイルシステムの機能を分散させ，1つのファイルサーバとして見せるネットワークファイルシステムの一種である分散ファイルシステムが提案されている．分散ファイルシステムはストレージが不足して新たなサーバを追加する場合に，アプリケーションに対してシームレスな容量拡張が可能になる．

しかし，一般的に分散ファイルシステムはメタデータサーバを単一のノードで構成しているために，単一障害点やシステム全体のボトルネックになるという問題点があ

るなど課題も多い．

2.2 分散ファイルシステム Ceph

Ceph ファイルシステムは，非常に高い性能，信頼性，そしてスケーラビリティを目標としたオブジェクトベースの分散ファイルシステムである．スケーラビリティは，ファイルシステムにおけるストレージの容量，全体的なスループット，ディレクトリやファイルの性能等，様々な面を考慮している．また，システム上で動作するアプリケーションや扱うデータセットが変化すると，データとメタデータへのアクセスにも変化が表れることから，分散ファイルシステムのワークロードが本質的に動的であり，高い性能を維持するためには様々なワークロードに適応的に対応する必要がある．そのため，Ceph ファイルシステムは動的なワークロードに対して，適応的に負荷を再分散し，データを再配置するシステムとして設計されている．対象とするワークロードには，数万から数十万のホストが同様のディレクトリ下でファイルの読み出し，書き込み，または生成を行うような極端なケースも含まれている．

2.2.1 特徴

Ceph ファイルシステムはスケーラビリティの問題を扱うと同時に，高い性能，信頼性，そして可用性を達成するために，データとメタデータの分離，メタデータの動的な分散管理，信頼性の高い自律的な分散オブジェクトストレージという 3 つの特徴を持つ．

ストレージとメタデータの分離

Ceph ファイルシステムは，図 4 のようにデータの永久化をオブジェクトストレージディバイスが行い，メタデータの管理をメタデータサーバが集中的に行うことで，ストレージからメタデータの管理を切り離す．そして，クライアントはストレージに対して，ファイルデータの入出力（write，read）を直接要求する．このようにデータとメタデータの操作を別々のサーバで行うことで，ストレージの高いスケーラビリティと大容量のファイルデータの入出力の高速性を両立を可能にする．

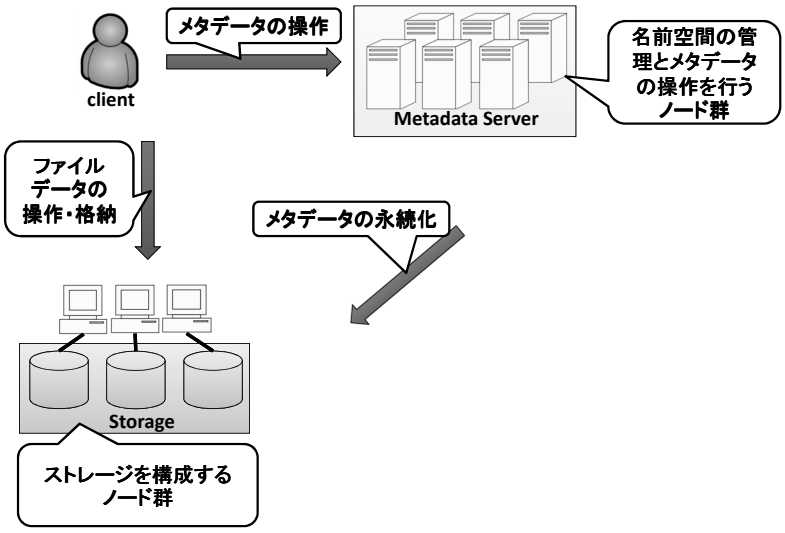

図 1: ストレージとメタデータの管理

メタデータの動的な分散管理

一般的にファイルシステムにおけるワークロードの約半分はメタデータ操作であるため，メタデータの管理の効率化はシステム全体の性能向上を図る上で需要な条件の 1 つである．Ceph ファイルシステムは，階層構造を持つ名前空間の管理を複数台のメタデータサーバに適応的に分散させる動的サブツリーパーティショニング [参] に基づいた手法を用いる．動的サブツリーパーティショニングは，名前空間をサブツリーに分割し複数台のメタデータサーバに分散させ，メタデータの局所参照性を維持することで，効率的なメタデータの更新と積極的なプリフェッチを容易にし性能の向上を可能にする．また，アクセスパターンに基づいて名前空間の管理を分散させることで，メタデータサーバクラスタ全体の負荷が均一化し，メタデータサーバのリソースを効果的に利用することを可能にする．その結果，メタデータサーバの台数によってメタデータの性能は線形にスケールする．

高信頼性の自律的な分散ストレージシステム

ペタバイト級の大規模なストレージシステムは数千ものディバイスから構成され，ディバイスの追加・離脱，障害の発生，大容量のデータの作成・移動・削除により，その様相は動的に変化する．そのため，利用可能なストレージリソースを効果的に用い，適切なレプリケーション数を保つために，複数のディバイスに分散したデータをストレージシステムの変化に合わせ再配置する必要がある．そこで Ceph ファイルシステムはストレージを構成するオブジェクトストレージディバイスに，データのマイグレーションとレプリケーション，障害の検知と回復を自律的に行わせる．同時に，オブジェクトストレージディバイスはクライアントとメタデータサーバに対して，信頼性の高いデータの分散方法を提供する．このアプローチは，Ceph ファイルシステムがより効果的に各オブジェクトストレージディバイス上のインテリジェンス（CPU，メモリ）を活用することを可能にし、線形にスケールする高い信頼性と可用性を持つオブジェクトストレージを実現する．

2.2.2 主要なコンポーネント

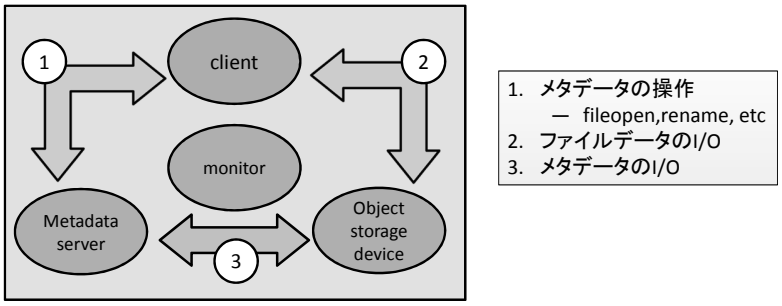

図 2: Ceph ファイルシステムのコンポーネント

Ceph ファイルシステムには図 2 に示すように 4 つ主要なコンポーネントが存在する．ユーザとなるホストやプロセスに対して，Ceph ファイルシステムに対してインターフェースを提供するクライアント，Ceph ファイルシステムのストレージを構成するオブジェクトストレージディバイス（OSD），名前空間の管理とメタデータの操作を行

うメタデータサーバ（MDS），クラスタ全体の状態を監視し，障害やノードの追加・離脱の検知を行うモニタである．次に各コンポーネントごとの詳細を説明する．

クライアント

Ceph ファイルシステムのクライアントは，アプリケーションプログラムを実行している各ホスト上で動作し，そのアプリケーションに対して Ceph ファイルシステムを利用するためのインターフェースを提供する．実装されるインターフェースは，アプリケーションプログラムやユーザが仮想ファイルシステム（VFS）を介して Ceph ファイルシステムにアクセスを行えるように，write，read，open，close などである．VFS とは，異なる実装のファイルシステムを，その構成や用途を意識することなく利用できるように仮想化する抽象化層のことである．VFS を介してアクセスすることで，分散ファイルシステムを始めとするネットワークファイルシステムを，トランスペアレントな存在として扱うことが可能になる．したがって，ユーザは大規模なストレージシステムにアクセスするとき，Ceph ファイルシステムを構成するメタデータサーバ，モニタ，オブジェクトストレージディバイスが，大規模なストレージプールに集約されていることを意識することはなく，標準ファイル入出力可能なマウントポイントが見えるだけである．また，クライアントが提供する Ceph ファイルシステムのインターフェースは POSIX に準拠している．これは，アプリケーションの要求に応じて，インターフェース自体を拡張し，コンシステンシのセマンティクスを選択的に緩和し，システム性能を向上させることに適しているためである．

オブジェクト ストレージディバイス

Ceph ファイルシステムのストレージは複数台のオブジェクトストレージディバイスに分散して存在する．オブジェクトストレージディバイスは，データオブジェクトをマッピングする対象であると同時に，オブジェクトストレージディバイスデーモンが動作するローカルのストレージに対して，データの読み書き等の命令を行うイニシエータの役割を持つインテリジェンスなディバイスである．

データとオブジェクトストレージディバイスとのマッピングにはCRUSH（Controlled Replication Under Scalable-Hashing）を用いる．CRUSHはヘテロジニアスなストレージクラスタ上に，データとその複製データであるレプリカを偏りがないよう均一に分散させるアルゴリズムである．ファイルアロケーションテーブルのようなデータの配置情報を必要としない．その代わりに，クラスタマップと呼ばれるストレージクラスタとレプリケーションの規則の記述を用いる．クラスタはストレージクラスタの構成に変化が生じる度に再編成される．そして，オブジェクトストレージディバイスはその情報を基にデータの再配置やレプリケーションを行うことで，ストレージの高い信頼性と性能を保証する．

また，個々のオブジェクトストレージ上のローカルストレージを管理するインターフェースとしてEBOFS （Extent and B-tree based Object File System）を用いる．EBOFSはFUSE（Filesystem in Userspace）によって実装されており，カーネルに実装されたファイルシステムとの機能の重複やVFSのiノードやページキャッシュと相互の影響を避けるために，直接rawブロックディバイスとやりとりを行う．

メタデータサーバ

現在，広く利用されている分散ファイルシステムは，メタデータサーバを一台の計算機で構成するものが一般的である．しかし，この形態には，2つの問題点がある．1つは，メタデータサーバが分散ファイルシステムにおける単一障害点になることである．アプリケーションやホストはファイル操作を行うとき，クライアントが提供するインターフェースを介して，メタデータサーバに対してメタデータ操作の要求を行う．メタデータサーバは，ファイルの有無，アクセス権限を確認し，クライアントがストレージディバイスにアクセスするために必要な情報を送信する．そのため，メタデータサーバに障害が発生した場合，システム全体が停止してしまう恐れがある．もう1つは，クライアントにとってメタデータサーバがボトルネックになりうることである．これは，分散ファイルシステムが数百，数千，利用目的によっては数万のクライアントが存在し，複数のクライアントが同時にメタデータサーバにメタデータ操作の要求を行う場合があり，作業負荷が集中してしまうためである．Cephファイルシステムの

メタデータサーバは，複数台の計算機でメタデータを分散管理する．また，変化するワークロードに対応して，メタデータサーバを構成するノード間でメタデータをマイグレーションすることで，サーバの一部に負荷が集中することを防ぐ動的な負荷分散を行う．

モニタ

モニタは，Ceph ファイルシステムの全体の監視を行い，障害の発生，ノードの追加・離脱等の変更を検知する．メタデータサーバとオブジェクトストレージディバイスの障害検知の方法は異なる．メタデータサーバは，一定の間隔でビーコン信号をモニタに対して送信する．モニタは，ビーコン信号を一定時間送信しなかったノードが存在した場合，そのノードに障害が発生したものと判断し，正常に動作しているメタデータサーバの他のノードに対して，状態の変化をブロードキャストする．また，メタデータサーバは，自分自身が送信したビーコン信号に応じる明確な ACK を適時に受信しなかった場合，動作を停止する．

一方，オブジェクトストレージディバイスの場合，モニタが障害を検知する前に，オブジェクトストレージディバイス間で，周期的に交換されるハートビートと呼ばれるメッセージによって，障害の有無が判断される．発生したオブジェクトストレージディバイスの障害は全て，モニタに報告される．モニタはオブジェクトストレージクラスタの構成の変化に合わせクラスタマップを更新し，各構成要素に対して最新のものを配布する．さらに，モニタは耐故障性向上のために，奇数台での冗長化を行う．したがって，オブジェクトストレージディバイスは，モニタに最新のクラスタマップを要求し，一定時間経過しても応答が得られなかった場合，異なるモニタと通信することでクラスタマップの取得を行う．その際，モニタクラスタの各ノードからの応答の一貫性を保証する必要がある．そこで，モニタクラスタが持つクラスタマップの強固な一貫性維持のために，Paxos の part-time Parliament アルゴリズムを用いる．このアルゴリズムは，分散コンピューティングシステムにおける過半数までの障害を許容しつつ，単一の結果について合意を得るものである．part-time Parliament アルゴリズムは，モニタクラスタから，クラスタマップの更新を逐次に行い，一貫性の管理を行う

リーダとなるモニタを選出する．リーダとして選出されたモニタは，各モニタによってストアされているクラスタマップのエポックを要求する．リーダは一定時間内，単一の結果，この場合は最新のクラスタマップについての合意を得るために必要な定足数のモニタからの応答を受け付ける．そして，リーダは過半数のモニタが動作中であり，最新のエポックを持っていることを保証し，動作が確認されたモニタ（アクティブモニタ）に対して短期的にリースを配布する．リースとは，クラスタマップをクライアントやオブジェクトストレージディバイスに要求されたモニタに，モニタの持つクラスタマップが最新であることを保証し，そのコピーを配布することを許可するためのものである．リースが一定時間内に更新されず無効になった場合，リーダが停止したと仮定し再選出する．また，アクティブモニタはリースに対する ACK をリーダに送信する．この ACK の送信をリーダで確認できなかった場合，そのアクティブモニタが停止したと仮定し再選出を行う．

クラスタマップの更新要求を受信したアクティブモニタは，その要求が新しいものか確かめる．更新要求が新しいものである場合，リーダに転送される．リーダは複数の更新を順序付け，エポックをインクリメントした後，Paxos の更新プロトコルを用いて，他のモニタに更新情報を配布し，同時にリースを無効化する．そして，リーダは過半数のモニタから更新情報に対する ACK を受信したことを確認し次第，新しいリースを配布する．この振る舞いは，アクティブモニタの構成に変化が生じた場合，クラスタマップの更新が部分的に実行される前に，全てのリースが無効化されることを保証する．したがって，過半数のモニタが利用可能であれば，クラスタマップの要求と更新は矛盾なく行われる．

2.2.3 CRUSH

CRUSH はストレージを構成する複数のオブジェクトストレージディバイス上に，データを分散させるアルゴリズムである．その実装はクライアントインタフェースに施されており，データとオブジェクトストレージディバイスをマッピングするために，クライアントはメタデータの一部（ファイル固有の ID，データサイズ，ストライピング情報）とクラスタマップを，そろぞれメタデータサーバとモニタから受け取り，CRUSH

によってデータとオブジェクトストレージディバイスの対応関係を算出する．そのため，ストレージクラスタの構成が変化（追加・離脱など）に伴い，ストレージに格納されているデータが偏りがないようマイグレーションが生じたとしても，クラスタマップをストレージの変化に応じて更新することで，データとオブジェクトストレージディバイスの対応関係を，CRUSH によって算出することが可能となり，データの配置情報の変更などを行う必要がない．したがって，ストレージはメタデータサーバに依存することないため，メタデータサーバの負荷を減らし，ストレージを容易にスケーリングすることが可能となる．

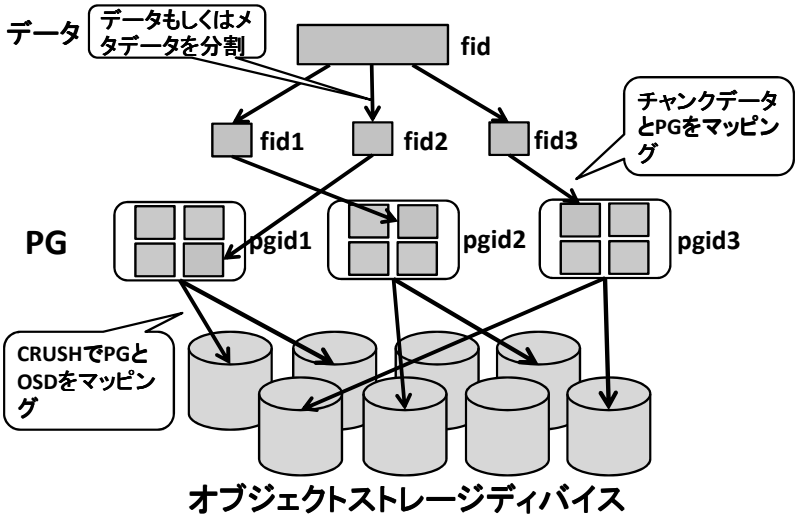

図 3: マッピングの流れ

次に図 3 クライアントがファイルデータとストレージをマッピングするまでの一連の動作を説明する．まず，データはサイズに応じて，複数のチャンクデータに分割される．チャンクデータには INO と連番が結合した名前である OID が割り当てられる．そして，プレースメントグループ（PG）と呼ばれる概念的なコンテナにマッピングされる．マッピングする PG は OID の単純なハッシュアルゴリズムによって得られた値とデータのレプリケーション数，そして PG の総合数を制御するビットマスクを用いて決定する．そして，最終的に各 PG を CRUSH によってオブジェクトストレージディバイスとマッピングすることで，データとストレージがマッピングされる．

2.2.4 RADOS

RADOS は数ペタバイトの大規模なストレージにスケールすると同時に高い性能と信頼性を実現し，規模の拡大・縮小を繰り返すヘテロジニアスなストレージクラスタ上に，データとワークロードの均一な分散を容易にする一方で，明確に定義されたセマンティクスと強固な一貫性を持つデータの格納方法をアプリケーションに提供するデータの格納方法である．

RADOS クラスタはオブジェクトストレージディバイスの巨大な集合と，ストレージクラスタを管理するモニタの小規模な集合，そしてアプリケーションのインターフェースであるクライアントから構成されている．RADOS はスケーラビリティと効率的かつ一貫性の保たれたデータアクセスを容易にし，ストレージクラスタの変化にシームレスに適応するために，ストレージクラスタの状態を監視するモニタはストレージクラスタの状態の変化を検出すると，オブジェクトストレージディバイスとストレージクラスタを利用しているクライアントに，クラスタマップの変更を安全かつ効率的に配布する．また，レプリケーション，障害検知と障害回復の操作のインテリジェンスは，オブジェクトストレージディバイスに分散される．ストレージクラスタの一部に障害が発生した場合，モニタによって更新されたクラスタマップをオブジェクトストレージディバイスに伝播し，自律的に信頼性の向上のためにデータをレプリケーションし，リソースの効率的な利用のためにデータをマイグレーションし，データと負荷を均一に分散する．そのため，ストレージクラスタが数ペタバイトにスケールに伴って大きくなるモニタの負荷を最小限に抑えることが可能である．

2.2.5 適応的なメタデータの分散管理

GFS を始め一般的な分散ファイルシステムはメタデータを管理するメタデータサーバ（GFS の場合はマスタサーバ）は 1 台の計算機で構成されている．この形態の分散ファイルシステムには問題点が 2 つある．

1 つ目の問題点はメタデータサーバが単一障害点であることである．図??が示すように，クライアントはファイルデータへのアクセスを行う前にメタデータサーバにアクセスする．メタデータサーバは，クライアントが要求するファイルデータが存在し，

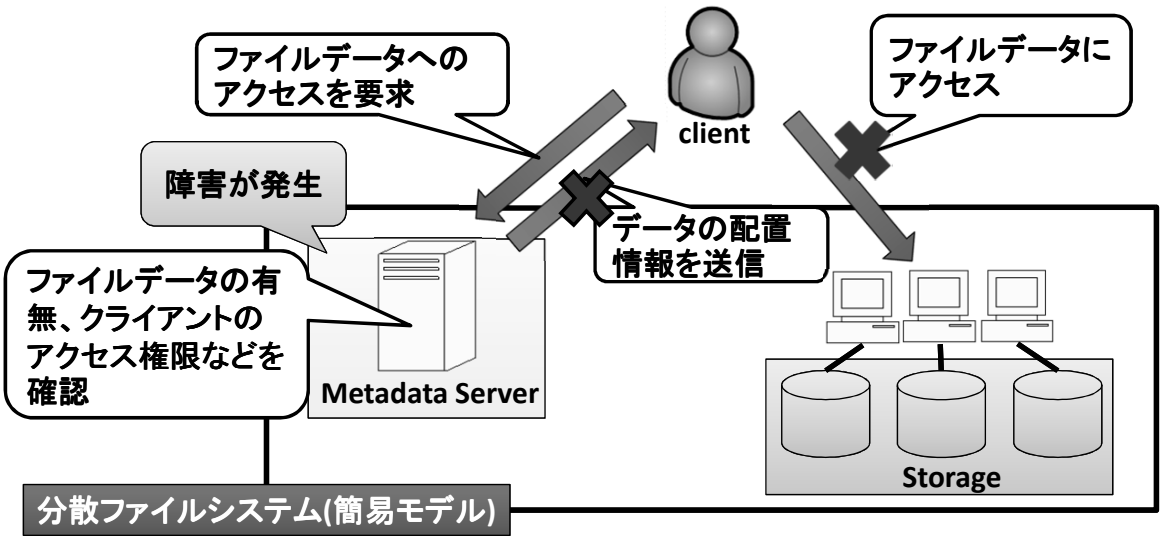

図 4: 単一障害点

またそのクライアントのファイルデータに対してアクセスする権限を持つかを確認する．ファイルの存在を確認しクライアントにアクセスする許可があれば，メタデータサーバはクライアントに対してファイルデータにアクセスするために必要な情報を送信し，クライアントはその情報からストレージ上に配置されたファイルデータにアクセスする．したがって，メタデータサーバに障害が発生した場合，システム全体が停止する．

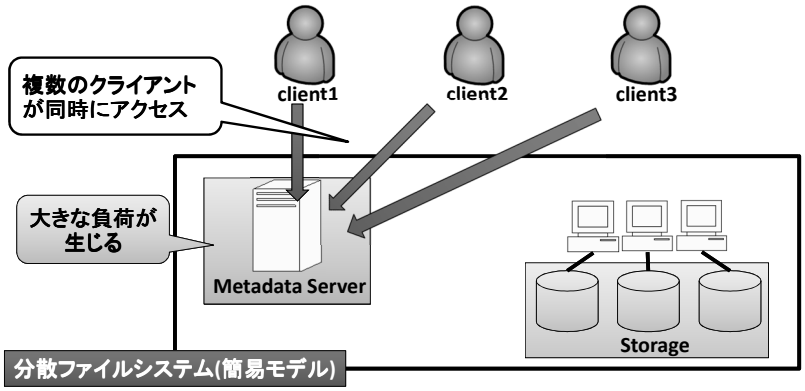

図 5: ボトルネック

2つ目の問題点はメタデータサーバがシステム全体におけるボトルネックになりうることである．図5が示すように一般的に分散ファイルシステムは複数のクライアントによって利用される．そのため，メタデータサーバにアクセスが集中し，大きな負荷が掛かりメタデータサーバはその性能を低下させ，システム全体のボトルネックになる．したがって，負荷の集中を回避しシステムの可用性を高めるためには，複数のメタデータサーバによってメタデータを分散管理する手法が必要となる．

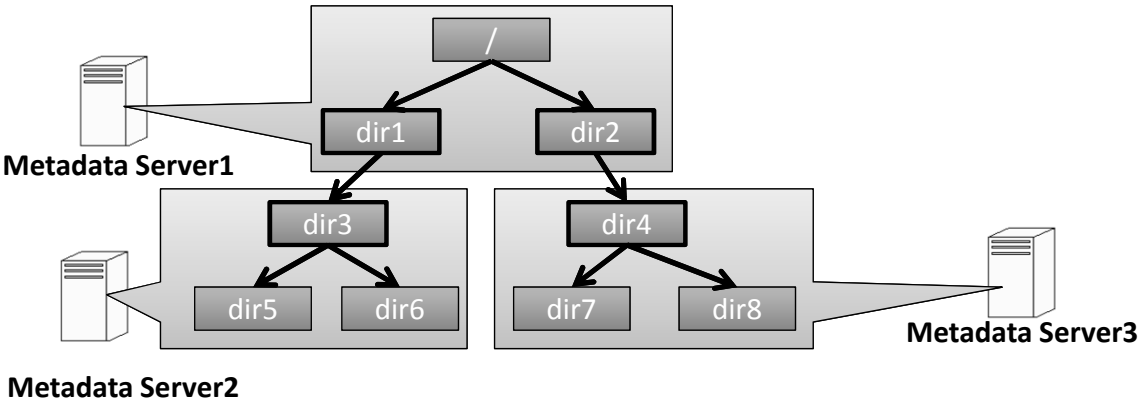

図 6: 静的サブツリーパーティショニング

クラスタファイルシステムでは，一般的に静的サブツリーパーティショニングという手法でメタデータ管理の分散を行うことで，これらの問題点に対処する．静的サブツリーパーティショニングは図6のようにファイルシステムにおける階層的な名前空間を複数のサブツリーに分割し，それらの管理を複数のメタデータサーバに分散することで負荷分散を可能にする．しかし，この手法には問題点がある．時間の経過に伴い，ファイルシステムが管理するデータ数は増減するため，システム管理者の手によって名前空間の再分割・再配置を行わなければメタデータサーバ間で管理するメタデータの数に偏りが生じてしまう．また，ファイルシステムにおけるワークロードの変化は，システム全体の負荷を大きく変動させ，メタデータサーバの一部に負荷が集中してしまう可能性がある．そこで，Cephファイルシステムはサブツリーパーティショニング，サブツリーマイグレーション，ディレクトリフラグメンテーションの3つの手法でメタデータ管理を変化するワークロードに対して適応的に分散させることで，シ

ステム全体の性能を低下しないよう維持する．

動的サブツリーマイグレーション

動的サブツリーパーティショニングは静的サブツリーパーティショニングと同様に，ファイルシステムの階層的な名前空間を複数のサブツリーに分割し複数のメタデータサーバで分散管理するだけでなく，メタデータサーバ間の負荷に偏りが生じた場合に動的にサブツリーを分割し，メタデータサーバ間でマイグレーションすることで，負荷を均一化し変化するワークロードに対応する．

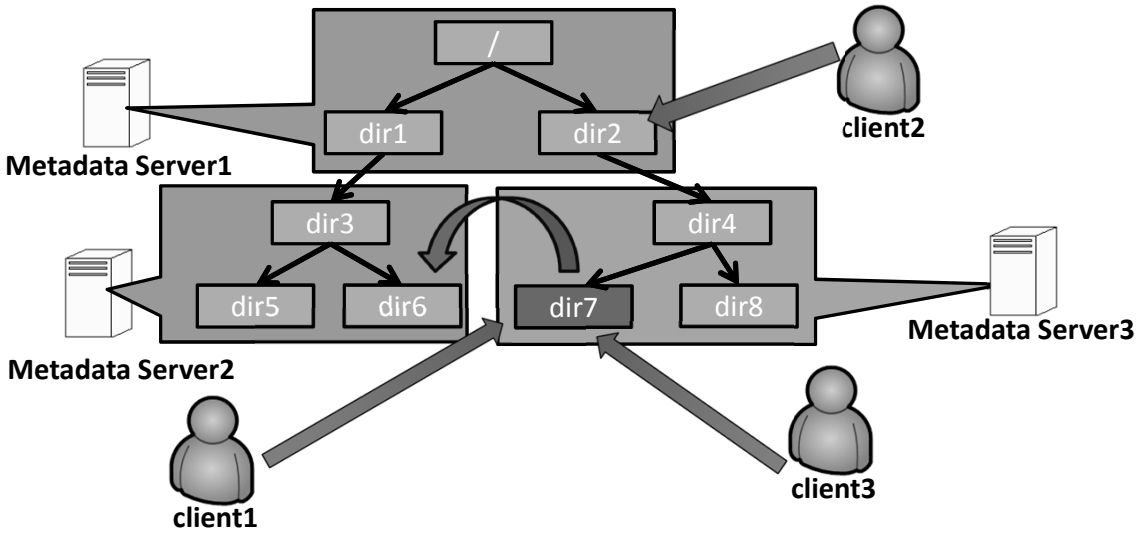

図 7: 動的サブツリーパーティショニング

図 7が示すようにメタデータサーバ間で負荷の偏りが生じるシチュエーションでは，負荷が集中するメタデータサーバとアイドル状態になるメタデータサーバが存在するため，リソースを十分に利用できず負荷の集中により性能が著しく低下してしまう．そこで，各メタデータサーバはそれぞれの負荷を検出するためにアクセス頻度を表すパラメータを保持し，メタデータサーバ間でその値を定期的に共有する．そして，負荷が高いメタデータサーバの管理する名前空間のサブツリーを分割し，負荷が均一になるように負荷が低いメタデータサーバにマイグレーションする．

ディレクトリフラグメンテーション

メタデータサーバの負荷の偏りを動的サブツリーパーティショニングを用いても均一化できないシチュエーションが存在する．あるディレクトリに対してのみ負荷が集中した場合，負荷が高いメタデータサーバから負荷が低いメタデータサーバにサブツリーのマイグレーションを行ったとしても，サブツリーのマイグレーション先のメタデータサーバに負荷が集中するため，メタデータサーバ間での負荷が均一化できない．

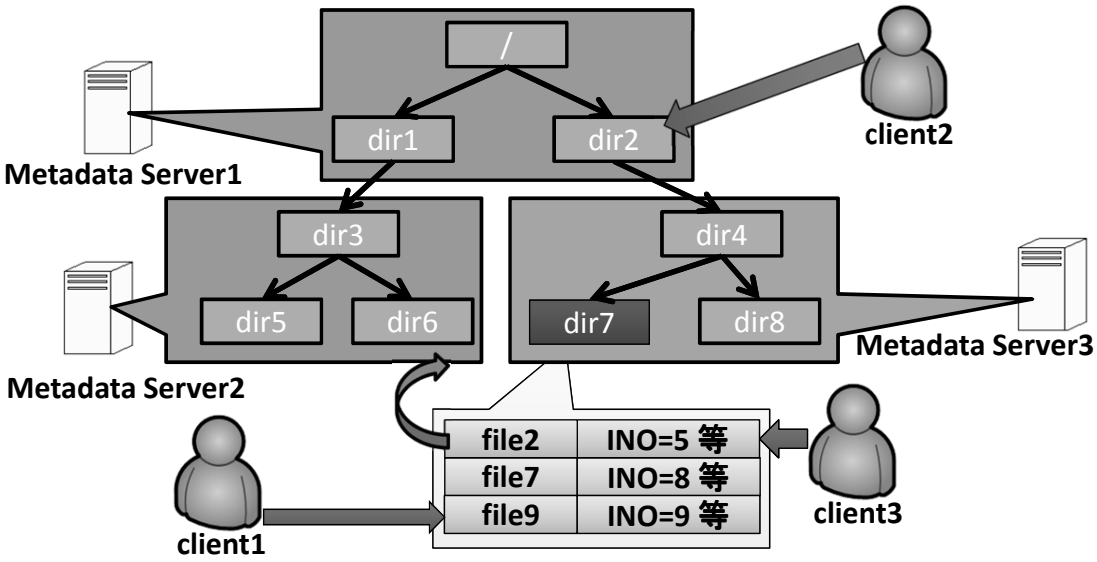

図 8: ディレクトリパーティショニング

そこで図 8 が示すようにサブツリーよりも細かい粒度での負荷均一化を行う．あるディレクトリ下のファイル，あるいはディレクトリのメタデータのアクセス頻度に偏りがある場合が多い．ディレクトリフラグメンテーションはディレクトリ単位で管理されているメタデータの組を複数の断片に分割し，その断片の一部を負荷の高いメタデータサーバから負荷の低いメタデータサーバにマイグレーションし，負荷の均一化を行う．

3 提案

Ceph ファイルシステムにおけるデータとディバイスのマッピング手法である CRUSH の改良が必要だと考えられる．CRUSH が用いるクラスタマップは，バケットと呼ばれるノードで構成される階層型のデータ構造を持つ．現在，4 種類のバケットが存在し，それぞれがさらに異なるデータ構造を持ち，バケットを構成する要素を選択する際はハッシュに基づくアルゴリズムに従う．しかし，Ceph ファイルシステムは厳密にオブジェクトストレージディバイスが管理するディスクの空き容量を管理しているわけではない．そのため，データの削除と生成が繰り返されるとデータの偏りが生じる場合が出てくる可能性がある．したがって，各ディバイスが格納するデータ容量を考慮したマッピング方法を考案する必要がある．

また，データに対するアクセスの偏りが出た場合，特定のオブジェクトストレージディバイスに負荷が集中してしまう可能性があり，システム全体の性能を著しく低下させ，さらに一部のオブジェクトストレージディバイスがアイドル状態になり，大部分の CPU，メモリ，そしてストレージディバイスといった利用可能な各種リソースが利用されない恐れがあることから，アクセス負荷を考慮したデータのマイグレーション手法を用いることで効果的なストレージの利用が見込めると考えられる．

以上の課題を Ceph ファイルシステムに適用させるには，負荷に動的に対応するバケットを提案する．従来のバケットの各要素はそれぞれ 1 つの下位バケットをポインタするが，提案するバケットでは各要素が格納数を示すパラメータとアクセス頻度を示すパラメータを持ち，バケット内でパラメータを比較し値に偏りがある場合，値が大きい要素は現在ポインタしている以外の下位バケットをポインタすることで，データと負荷を均一に保つ．

4 評価

Ceph ファイルシステムの read・write の性能の特性と，オブジェクトストレージディバイス及びメタデータサーバの台数効果の調査するために異なるワークロードでの実験を行った．

4.1 実験概要

実験の内容について説明する．Ceph ファイルシステムのクラスタ構成を変え，400 クライアントが 10MB のファイルを作成し，クライアントは自身が作成したファイルに対して write もしくは read のどちらか一方の操作を行う．read と write の操作の割合を 10：0，7：3，5：5，3：7，0：10 の計 5 パターンで混成し，同時に行う．そして，各操作のスループット（KB/sec）を算出する．また，クラスタの構成はモニタの台数が 1 台の 1 パターン，オブジェクトストレージディバイスの台数が 3 台，5 台，7 台，9 台の 4 パターン，メタデータサーバの台数が 1 台，2 台，3 台の 3 パターンの計 12 パターンで行った．なお，クライアント側でのファイルデータ，メタデータのキャッシュを避けるために，ファイル生成後，一度 Ceph ファイルシステムをアンマウントする．また，ファイルデータ及びメタデータのレプリケーションは行わないものとする．

表 1: 評価環境

server OS	Ubuntu11.04
Ceph version	v.0.37
CPU	Core i5 750 / 2.67GHz
Memory	8GB
HDD	290G
ネットワーク	Ethernet (1Gbps)

実験に使用したサーバ OS，Ceph ファイルシステムのヴァージョン，またマシンとネットワークの性能を表 1 に示す．

4.2 実験結果

以下に各実験での評価結果を示す．縦軸はスループット (KB/sec)，横軸はオブジェクトストレージの台数である．グラフはそれぞれ，メタデータサーバを構成する物理ノード数が 1 台，2 台，3 台の場合の測定結果である．

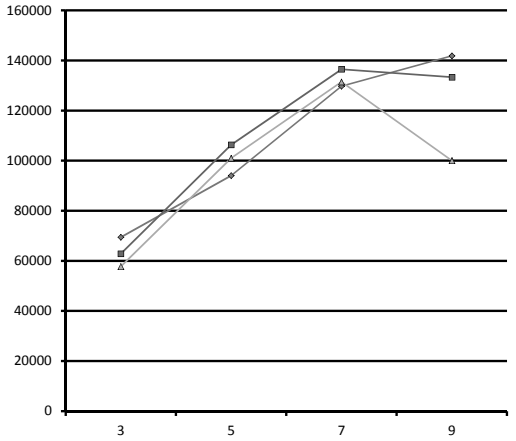

図 9: write スループット (write100% read0%)

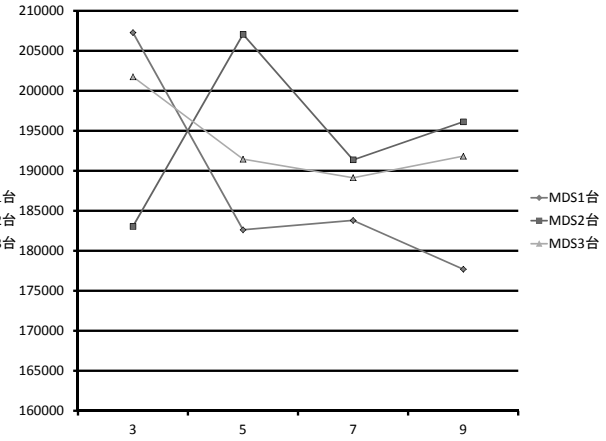

図 10: read スループット (write0% read100%)

図 9 は，400 クライアントがそれぞれ 10MB のファイルを作成した後，作成したファイルに対してシーケンシャルな write を行う．その結果，得られたスループット (KB/sec) の値をグラフで表したものである．メタデータサーバが単一の物理ノードで構成されている場合，ストレージを構成するオブジェクトストレージディバイスの台数の増加に伴い，線形にスケールしていることから，十分な台数効果を得られていることが確認できる．一方，メタデータサーバが複数台の物理ノードで構成されている場合，単一の物理ノードで構成されている場合と異なり，ストレージを構成するオブジェクトストレージディバイスの台数を 7 台から 9 台に増加すると，write 性能は向上せず低下した．これは，メタデータサーバ及びストレージを複数台のノードで構成した結果，生じるオーバヘッドが台数効果を上回ったためと考えられる．また，メタデータサーバの台数効果を確認することはできなかった．

図 10 は，400 クライアントがそれぞれ 10MB のファイルを作成した後，作成したファイルに対してシーケンシャルな read を行った結果得られたスループットをグラフに表したものである．write 操作のようにストレージを構成するオブジェクトストレージディ

バイスの台数の増加に伴って，スケールはしていない．しかし，read 操作は，全てのクラスタ構成において write 操作を上回る性能を示している．これは，read 操作の大半がストレージを構成するノードのメモリにアクセスし，メモリ上にキャッシュされたファイルデータを読み出している．一方，write 操作はメモリ上のファイルデータが変更された場合，データの整合性を保つために，ディスクに格納されたファイルデータを更新する．そのため，全てのクラスタ構成で，read 操作の性能は write 操作の性能を上回る結果を示したと考えられる．

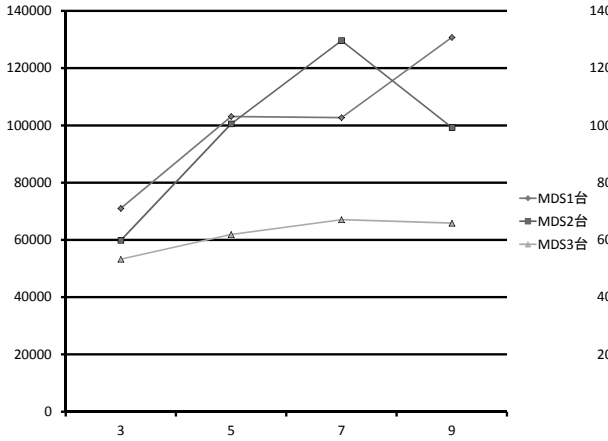

図 11: write スループット (write70% read30%)

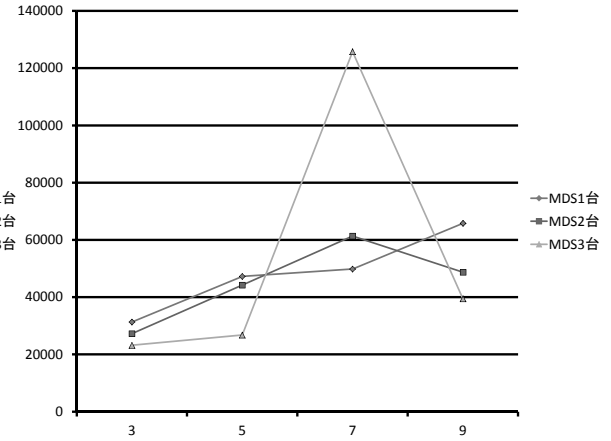

図 12: read スループット (write70% read30%)

図 11，図 12 は 280 クライアントが write 操作を，120 クライアントが read 操作を同時に行い，その結果，得られた 2 つの操作のスループットをグラフで表したものであり，図 11 が write 操作のスループットを，図 12 が read 操作のスループットを示す．write 操作，read 操作の双方が同時に行う入出力操作が一種類で行った時に比べ，性能が下回る結果となった．また，read 操作に関しては，高い台数効果を得ることはできなかった．

複数台の物理ノードでメタデータサーバが構成されている場合，オブジェクトストレージディバイス 7 台でストレージを構成した場合の write 操作に比べ，9 台で構成した場合の write 操作が示すスループットが低い結果となった．

図 13，図 14 は 200 クライアントが write 操作を，200 クライアントが read 操作を同時に行い，その結果，得られた 2 つの操作のスループットをグラフで表したものであり，

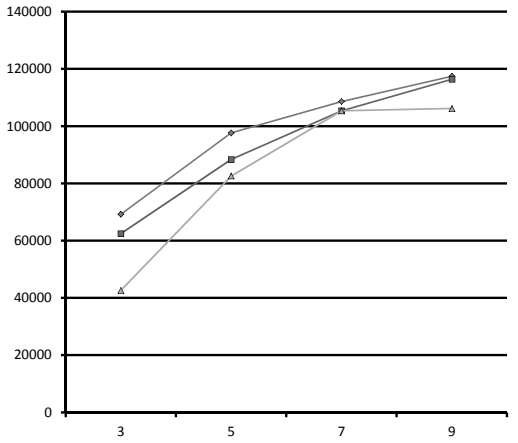

図 13: write スループット (write50% read50%)

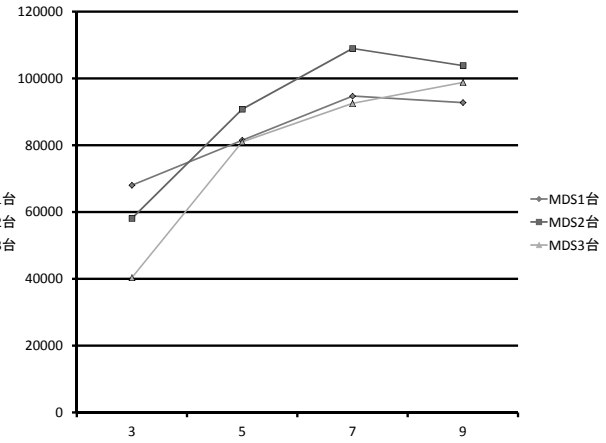

図 14: read スループット (write50% read50%)

図 13 が write 操作のスループットを，図 14 が read 操作のスループットを示す．read 操作，write 操作は共に，全てのクラスタ構成においてオブジェクトストレージディバイスの台数の増加に伴い，そのスループットが線形にスケールすることを示した．

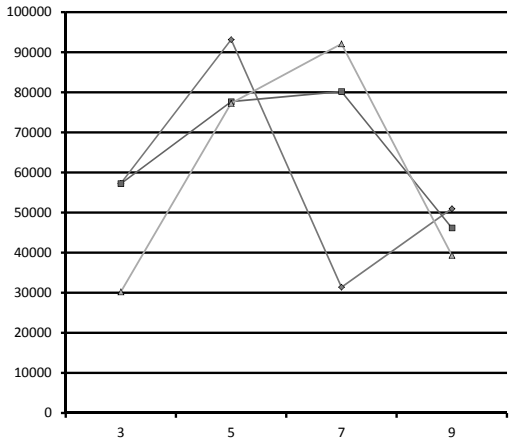

図 15: write スループット (write30% read70%)

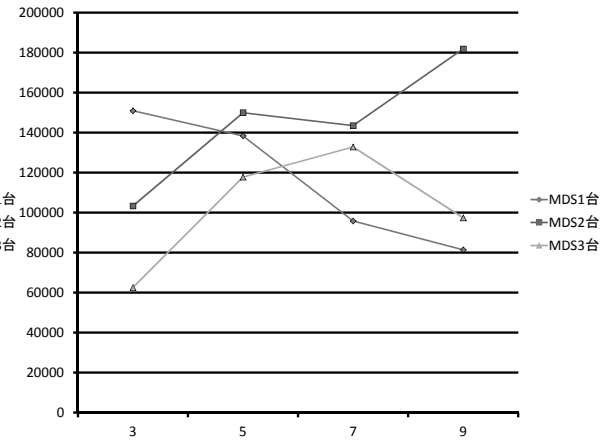

図 16: read スループット (write30% read70%)

図 15，図 16 は 120 クライアントが write 操作を，280 クライアントが read 操作を同時に行い，その結果，得られた 2 つの操作のスループットをグラフで表したものであり，図 15 が write 操作のスループットを，図 16 が read 操作のスループットを示す．read 操作の性能は図 12，図 14 と比較して全体的に，高い性能を示している．

図 17，図 18 は，図 9，図 10 の実験において，クライアントが動作するホストから，メタデータサーバを構成する物理ノードに対して送信されるパケットを，それぞれカ

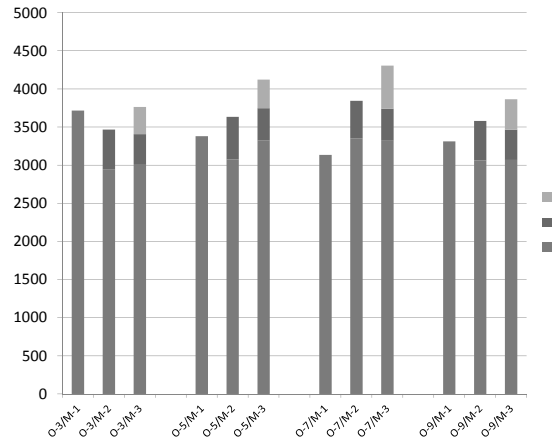

図 17: write におけるパケット受信量

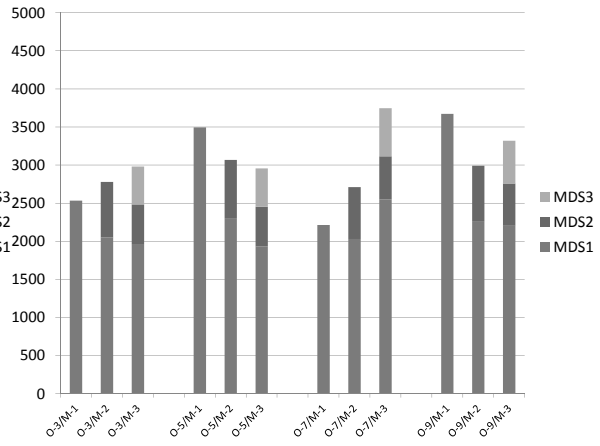

図 18: read におけるパケット受信量

ウントし，グラフとして示したものである．なお，パケットキャプチャには Wireshark を用いており，フィルタリングの条件はパケットの送信元 IP アドレスをクライアントホストのものとし，受信先 IP アドレスをメタデータサーバを構成する物理ノード，それぞれの IP アドレスとする．また，パケットキャプチャはクライアントが Ceph ファイルシステムをマウントし，write 操作もしくは read 操作を行い，アンマウントするまでの期間行う．グラフは，測定した際のストレージを構成するオブジェクトストレージデバイスの台数によって，4 つのグループ (左から 3 台，5 台，7 台，9 台) に分ける．また，グループ内のグラフは，いづれも左からメタデータサーバが 1 台，2 台，3 台で構成された時のものである．

read 操作は write 操作に比べ，クライアントからメタデータサーバに対して送信されるパケット数が全体的に少ない．また，メタデータサーバが複数台で構成されている場合，メタデータサーバが受信するパケット数に偏りがある．

5 考察

性能評価で示したように，Ceph ファイルシステムはストレージを構成するオブジェクトストレージディバイスの台数の増加に伴い，write 操作のスループットが線形にスケールしていることから，十分な台数効果が望めると考えられる．read 操作は線形にスケールしていない場合があるが，write 操作と同等の割合で混成した場合 (400 クライアント中，200 クライアントが write 操作，残り 200 クライアントが read 操作を行った場合)，線形にスケールしていることから，ストレージに高い負荷が掛かる状況下では，台数効果を得られることが予測される．一方で，メタデータサーバを構成する物理ノードの台数が複数台である場合，通信オーバヘッド等により，性能の低下が見られる．効果的なパケット転送方法を考慮していく必要があるだろう．

Wireshark によって，クライアントがメタデータサーバに対して，送信するパケット量を測定した．その結果，複数の物理ノードでメタデータサーバが構成されている場合，クライアントのアクセスが分散していることが分かる．本研究で行った実験では，ディレクトリフラグメンテーションによるメタデータサーバのアクセス負荷の分散が十分な効果を得られていない．ディレクトリフラグメンテーションは，1 つのディレクトリに対するアクセス頻度が高い場合か，そのディレクトリ直下で管理されるファイル (もしくはディレクトリ) に関するメタデータ数が増加し，ディレクトリが頂戴なものとなった場合に行われる処理である．すなわち，動的サブツリーパーティショニングのように負荷状況によらず，名前空間を分割するわけではない (動的サブツリーパーティショニングは，負荷によらず名前空間を複数のサブツリーに分割し，メタデータサーバを構成する複数のノードに分散させる)．ディレクトリフラグメンテーションを行う条件やメタデータをマッピングするディレクトリの断片の選択方法 (調査段階ではファイル (もしくはディレクトリ) 名から得られるハッシュ値を用いている) を改良する必要がある．

6　まとめ

本論文では，分散ファイルシステムの一種である Ceph ファイルシステムに注目し，その各種機能について調査を行った．また，複数のクラスタ構成において，400 クライアントがそれぞれ，write 操作もしくは read 操作を同時に行い，その割合を変更することで様々なシチュエーションの下での I/O 性能 (スループット) の測定を実施し，Ceph ファイルシスにおける I/O 性能について検討した．

さらに，Wireshark を用いて，クライアントからメタデータサーバに対して送信されるパケットの個数をカウントし，クライアントのメタデータサーバに対するアクセスの分散状況を調査した．その結果，クライアントのアクセスに偏りがあり，ディレクトリフラグメンテーションの効果を得られる条件やディレクトリの断片とメタデータのマッピング方法の見直しが課題として考えられる．

本論文内では，Ceph ファイルシステムの I/O 性能の評価を行い，その特徴を調査したが，今後は障害検知や障害復旧，ノードの追加・離脱等，I/O 性能に限らず，分散ファイルシステムとしての機能の調査を行い改善点を検討していく必要があるだろう．

謝辞

本研究のために，多大な御尽力を頂き，御指導を賜った名古屋工業大学の松尾啓志教授，津邑公暁准教授に深く感謝致します．

また，本研究の際に多くの助言，協力をして頂い松井俊浩准教授，齋藤彰一准教授，松尾・津邑研究室並びに齋藤研究室，松井研究室の皆様に深く感謝致します．

参考文献

[1] Sage A. Weil. *Ceph: Reliable, Scalable, and High-Performance Distributed Storage.* PhD thesis, University of Califonia, SantaCruz, December 2007.

[2] Sage Weil, Scott A. Brandt, Ethan L. MIller, and Carlos Maltzahn. Crush: Controlled, scalable, decentralized placement of replicated data, November 2006.

[3] Sage Weil, Scott A. Brandtand Ethan L. MIller, and Carlos Maltzahn. Rados: A fast, scalable, and reliable storage service for petabyte-scale storage clusters, November 2007.