情報処理学会論文誌 Vol. 52 No. 2 488–506 (Feb. 2011)

ユーザによる柔軟なデータ管理のための広域分散メタデータ管理システムの提案と評価

池部 実[†1] 猪俣敦夫[†1]
藤川和利[†1,†2] 砂原秀樹[†1,†3]

グリッドやクラウドで処理したデータは，インターネット上に分散するストレージへ出力される．従来，ユーザが目的のデータへアクセスする手段は，ファイル名のみが利用できていた．しかし，従来のデータアクセスをグリッドやクラウド環境に適用すると，膨大なデータの中から目的のデータをユーザが発見するまでの負担が大きい．一方，データアクセスの際に，データの特性やセマンティクスを用いることができると，ユーザは直感的にデータへアクセス可能となる．本論文では，ユーザが広域分散処理環境でデータアクセスする際に，データの特性やセマンティクスを利用するための広域分散メタデータ管理システム MetaFa を提案する．MetaFa では，データを処理するワーカノード上でアプリケーションの I/O 性能への影響を低く抑えながら，リアルタイムにメタデータを取得する．メタデータ管理サーバでは，アプリケーションのメタデータスキーマを管理せずにメタデータを管理することが可能となる．さらに本論文では，MetaFa システムの有用性を評価するために，プロトタイプを実装し，広域分散環境で動作するメタデータ管理の性能評価を実施し，MetaFa システムの有用性を示した．

MetaFa: Metadata Management System for Flexible Data Access in the Wide Area Network Environments

Minoru Ikebe,[†1] Atsuo Inomata,[†1]
Kazutoshi Fujikawa[†1,†2] and Hideki Sunahara[†1,†3]

An enormous amounts of data generated by applications on the grid or the cloud computing are still output to storages scattered on the Internet. However in such the situation, a burden of users that they find out their target file become increasingly bigger if an existing access method is applied to the environment without change the current architecture. Metadata is very useful to represent the characteristics of data. Meanwhile if users can access by the characteristics or semantics, they can specify the target data intuitively. In this paper, we propose a wide-area distributed metadata management system MetaFa which is enable to access data and collect the metadata by such the characteristics or the semantics especially automatically and efficiently. MetaFa can acquire the metadata at real time with low-efficiency for an application I/O performance on a worker node. Also, the metadata management server can manage a variety of application metadata without managing the application metadata schemas. To evaluate an effectively of MetaFa, we implemented a prototype of MetaFa system. Finally, we mentioned and discussed about a possibility of MetaFa can be applied to the wide-area distributed computing.

1. はじめに

多くの計算資源を提供する環境としてグリッドコンピューティング[7] やクラウドコンピューティング[8),19)] などの，インターネットを活用した情報処理基盤が，近年，急速に普及しつつある．ユーザがグリッドコンピューティングやクラウドコンピューティング環境を利用して処理をすると，膨大な数のデータがインターネット上へ蓄積される．これらの環境では，ユーザは計算機資源やデータの位置を意識せずに処理を行うことができる．我々が想定する広域分散環境とは，インターネットを経由し，大学，研究機関などの組織をつないだ環境である．グリッドなどの広域分散環境を利用するアプリケーションとしては，分散する膨大なデータを処理するデータインテンシブアプリケーション[10)] がある．データインテンシブアプリケーションでは，科学技術分野において，実験機器から出力される数百 GB から数 TB のデータを処理するものや，センサネットワークでは，定期的に大量のセンサから出力される膨大な情報を逐次処理するものなど多岐にわたる．また，グリッドコンピューティングで取り扱う大規模なデータを管理するためのデータグリッドシステムがある．データグリッド技術を用いることでデータインテンシブアプリケーションが出力した膨大なデータを管理できる．多くのデータグリッドシステムでは，Lustre に代表されるクラスタファイルシステムのようにデータを保存するノード，データの場所を管理するノードなどデータに関

†1 奈良先端科学技術大学院大学情報科学研究科
Graduate School of Information Science, Nara Institute of Science and Technology
†2 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University
†3 慶應義塾大学大学院メディアデザイン研究科
Graduate School of Media Design, Keio University

する情報をデータ自体と分離してデータ管理を行う．ユーザに対しては，ユーザが利用する環境（UNIX など）と，データアクセスの互換性を保つために階層型ファイルシステムとして提供している．

一方，ストレージデバイスの大容量化，低価格化によって，一般ユーザでも数百 GB から数 TB の規模のデータが蓄積可能になってきた．ユーザは大量のデータを保存していくが，ファイルシステム上のどのディレクトリに保存したか覚えていないことがたびたび発生する．そのため，ユーザは必要なデータに対してすぐにアクセスできないことが生じる．ユーザは，デスクトップ検索ツールを利用することで，いつ，どこで，誰が作成したデータなのか，どのような種類のデータ，データの内容などのデータが持つセマンティクスを用いて検索できる．すなわち，ユーザはデータの保管場所を意識せずに，利用したいデータに関する知識を用いてアクセスすることができる．

多くのデータグリッドシステムでは，ユーザに対してファイルシステムやストレージシステムとして提供するため，データとユーザが利用可能なセマンティクスを結び付ける情報が存在しない．そのため，データグリッドシステムにおいて，データのセマンティクスでデータアクセスするためには，別途デスクトップ検索ツールなどを用いて，データとデータのセマンティクスを関連づける情報を作成する必要がある．我々は，データのアプリケーションレベルのセマンティクスを抽出し，ユーザの持つ知識を利用してデータへアクセスする “アプリケーションレベルセマンティクス・ファイル I/O オペレーション” を広域分散ネットワーク環境で実現することを目指す．アプリケーションレベルのセマンティクスとは，アプリケーションが生成したデータについて，特徴的な内容を体系的に表すものである．たとえば，一般的な画像データでは EXIF [4] 情報，天文学関連の画像データでは FITS [13]，医学分野の画像データは DICOM [12] などがアプリケーションレベルセマンティクスとなる．そこで本論文では，グリッドやクラウド環境において生成されたデータから，アプリケーションレベルセマンティクスを抽出し，広域で管理するための広域分散メタデータ管理システム “*MetaFa*（*Metadata administaration Factory*）” を提案する．さらに，ユーザがアプリケーションレベルセマンティクスを用いて，データへアクセスする際のデータ検索用 API も提供する．ユーザは MetaFa システムを利用することにより，ユーザが持つセマンティクスを用いて柔軟なデータ検索が可能となり，広域分散環境において直感的なデータアクセスが可能となる．ユーザが持つセマンティクスを利用したデータアクセスシステムの例として，階層型ファイルシステムのまま，ファイルの持つ属性をメタデータとして扱い，メタデータを用いて検索可能なファイルシステムとして Semantic File System（SFS）[9] が提案されている．SFS では，記述したメタデータの値に合致するファイルをローカルファイルシステムから検索し，ディレクトリ以下にシンボリックリンクを作成し，ユーザやプログラムからアクセス可能にしている．このように，メタデータを用いてファイルシステム上のデータを検索することで，階層型ファイルシステムでのデータの位置を意識せずに，データ操作が可能となる．ユーザは，メタデータを利用することにより，分散する大量のデータをユーザの知識を用いて，データのアクセスや整理ができる．すなわち，メタデータを用いたデータ管理手法は，ユーザによるデータ管理の負担を軽減することが可能である．MetaFa では，メタデータを用いた柔軟なデータアクセスのための広域分散メタデータ管理を実現する．

なお，本論文では，アプリケーションレベルセマンティクスとデータ管理のための情報を合わせてメタデータと呼ぶ．データ管理のためのメタデータを基本メタデータ，アプリケーションレベルセマンティクスをアプリケーションメタデータと呼ぶ．

本論文の構成を以下に示す．2 章では，アプリケーションが取り扱うデータの分析を行い，メタデータ管理に対する要求をまとめ，広域分散環境でのメタデータを用いたデータ管理システムについて述べ，既存の問題点の洗い出しを行う．3 章では，提案する MetaFa システムのアーキテクチャ，設計方針について述べる．4 章では，MetaFa のプロトタイプ実装について述べる．5 章では，MetaFa のプロトタイプの性能評価を報告し，提案手法の有効性を示す．6 章では，5 章での結果などから，MetaFa システムについて考察する．7 章で，本論文をまとめ，今後の課題について述べる．

2. アプリケーションにおけるデータの特性とメタデータの活用に関連した既存研究と動向

本章では，グリッドアプリケーションが扱うデータの特性について述べる．また，既存のメタデータを用いたデータ管理システムを説明し，それらが持つ問題点の洗い出しを行う．

2.1 アプリケーションにおけるデータ特性

グリッドアプリケーションにおけるデータの特性について述べる．グリッドコンピューティングが対象とする科学技術分野では，アプリケーションが取り扱うファイルのデータサイズや個数は年々増加傾向にある．生命科学分野におけるアプリケーションが取り扱うファイルサイズとファイル数の関係を**図 1** に示す[22]．図 1 から，1 アプリケーションが取り扱うファイル数は，10^9 個，最大のファイルサイズはペタバイトスケールである．将来的に，生命科学分野においても，対象とするデータ量はますます増加していくものと考えられる．

続いて，本論文において，アプリケーションで取り扱うデータの鮮度とメタデータの関係

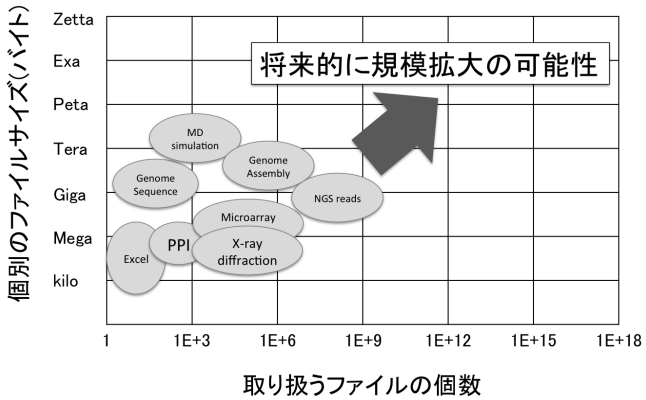

図 1　生命科学分野のアプリケーションにおけるファイルサイズとファイル数の関係
Fig. 1　Relationship of file size and file number in the bio-science applicaitons.

を説明する．データの鮮度とは，データが生成されてからユーザが利用するまでの時間と定義する．データの鮮度が必要な場合には，データが生成されてからユーザが利用するまでの時間が短いことを意味する．ユーザがメタデータを介してデータへアクセスする場合，データの鮮度に応じて，メタデータも利用可能な状態になっていなければならない．

アプリケーションによって扱うデータの特性の違いを以下で説明する．グリッドコンピューティング環境においては，3 種類のアプリケーションに大別できる．1 つめは，High Performance Computing（以下，HPC）アプリケーションである．HPC アプリケーションの例は，タンパク質立体構造予測などがある．タンパク質立体構造予測では，入力データ（塩基配列）に対して，複数のパラメータを与え，同時に大量の計算機を用いて演算を行う．異なるパラメータを用いて演算した結果に対して，統計処理を行い，最終的な結果を得る．HPC は，計算機資源を限られた時間内で処理を行い，得られた結果から統計処理を行うワークフローで実行する．そのため，データは生成されてから，ユーザが利用できるようになるまでの時間はできる限り短い方がよい．

2 つめは，Data-Intensive アプリケーションである．Data-Intensive アプリケーションは，高エネルギー実験装置などの超大型実験施設から得られる大容量データを処理する．これらのアプリケーションのワークフローは実験装置から出力された生データを Tier0 で保存し，生データを処理して 1 次データを作成し，Tier1 へ保存する．Tier1 に保存しているデータは，次々と別の組織へと複製される．ユーザは，実験で観測されたデータ（生データや 1 次データ）が 1–2 週間後などに利用可能であればよい．ユーザは，1 次データを利用して解析処理を行い 2 次データを生成する．生成した 2 次データを共有し，別のユーザが 2 次データを利用して新たなデータを次々と生み出す．そのため，ユーザはデータの鮮度をそれほど重要視していない．データが複製され，分散して管理されるためメタデータの複製や分散管理が必要となる．

3 つめは，センサネットワークアプリケーションである．たとえば，気象センサから発生するデータを取り扱うアプリケーションである．センサから発生される 1 つ 1 つのデータサイズは小さい．センサは，地理的に分散した場所に配置され，逐次データを出力する．センサネットワークにおけるデータの鮮度は，データのリアルタイム処理や，蓄積したデータから過去の状況を分析するなど，多岐にわたるため一概に決定されない．

また，ユーザは，Web サーバをクラウドコンピューティング，特に，IaaS（Infrastructure as a Service）技術を用いて手軽に利用できる．Web サーバ運用においては，ログデータが重要である．ログデータは，サーバがクラックされたなど，インシデントが発生した際に状況把握のために，管理者にとって重要情報源となる．インシデントの発生時には，すぐにログデータへアクセスする必要があるため，データの鮮度が重要である．アプリケーションメタデータを用いたサーバログデータ管理方法としては，ユーザが定義したカテゴリごとに分類してログデータを管理するなどがある．たとえば，日付ごと，アクセスしてきた送信元 IP アドレスなどのカテゴリに分類する．このような管理を行うことで，目的のデータに対してデータの持つ意味から即座にアクセスすることが可能となる．

従来は，ユーザグループの要求に基づき，アプリケーションごとに最適化された専用のデータ・メタデータ管理システムが構築されてきた．しかし，すべてのアプリケーションに対して，汎用的に利用できるシステムは提案されていない．本論文では，グリッドやクラウド環境において，アプリケーションレベルセマンティクスを広域で管理するうえで，それぞれのアプリケーションのデータに対する鮮度などの要求に対して，1 つのシステムでシンプルなアーキテクチャを保ちながら，対応できることを目的の 1 つとする．

2.2　アプリケーションにおけるメタデータの問題点

ユーザにとって，アプリケーションメタデータを取得することは大きな負担となる．ある分野では多くのアプリケーションメタデータの取得はユーザの目視によって行われてきた．さらに，アプリケーションメタデータのスキーマもユーザグループがそれぞれ定義してきたため，同じアプリケーションでもユーザグループが異なると，互換性がないことがある．ユーザは，アプリケーションメタデータを取得する際に，できる限りユーザの負担を軽減す

ることが求められている．

アプリケーションメタデータは，ユーザやアプリケーションごとにメタデータのスキーマが定義されているため，異なるスキーマのメタデータを統一的に管理するシステムは，複雑化し，性能が出にくくなる可能性が高い．また，グリッドコンピューティング環境で，データ処理が終了した直後に，出力したデータをまとめて処理する場合，つまり，メタデータに即時性が求められる場合には後からまとめてメタデータを取得する方法や目視でメタデータを記述する方法では間に合わない．2.1 節で述べたように，アプリケーションによって，メタデータがすぐに必要になる場合，2–3 日後にデータおよび，メタデータが利用可能であればよい場合など様々である．そのため，データの鮮度を考慮したメタデータ取得，管理技術が必要となる．

2.3　広域分散環境でのメタデータ管理システムにおける問題点

グリッドコンピューティングでは，ユーザがインターネット上の計算機資源を用いて，大規模計算を実行し，膨大なデータを出力する．グリッド環境では，ユーザのジョブはジョブスケジューラによって各サイトに割り当てられ，実行される．各サイトは，生成されたデータとメタデータを管理する．ユーザは，各サイトのメタデータ管理ノードへデータ検索のリクエストを発行しなければならない．このため，ユーザはどこにデータがあるかを意識してデータアクセスを行う必要がある．また，データグリッドでは，サイトを越えてデータの複製を行うため，データが別のサイトへ移動・複製された際，メタデータの一貫性を保証しなければならない．さらに，メタデータを広域分散環境で管理する場合，ユーザがどのメタデータ管理サーバへ問い合わせても同一の結果が得られることが重要となる．

2.4　メタデータを活用した関連研究

本節では，既存のメタデータを用いたデータ管理システムについて述べる．広域分散データ管理システムにおいて，メタデータの利用方法は以下の 2 種類に分類できる．

1 つめは，データの位置管理である．ユーザに対して，データの位置透過性を提供するために，実際のデータの位置をメタデータを用いて論理的な名前空間へマッピングする方法である．2 つめは，データに関する属性情報をメタデータに記述し，メタデータの内容に基づいてデータを検索する方法である．以下の項で，それぞれの関連研究について説明する．

2.4.1　メタデータによるデータ位置管理

メタデータで分散するデータの位置を管理するシステムについて述べる．主に，クラスタファイルシステムなどで用いられる手法である．代表的なシステムとして，Lustre[15]，Gfarm[17),21)] がある．これらのシステムでは，複数のストレージ，ファイルシステムを仮想的に統合し，仮想ディレクトリ構造をユーザへ提供する．仮想ディレクトリ構造とは，既存ディレクトリツリー構造の中に，分散するファイルシステムの提供するマウントポイントを組み込むことである．ユーザからは，既存ディレクトリツリー構造からどのファイルサーバに存在しているデータであるかを意識せずに利用できる．アプリケーションがファイル操作を行う場合，データの位置などを管理するメタデータ管理サーバに対して，open などファイル I/O 命令を発行し，メタデータ管理サーバからデータが存在するストレージ側へデータ操作命令が送信される．メタデータは，ユーザが直接参照することはなく，データ管理システムのみが利用する．これらのシステムでは POSIX ファイルシステムのセマンティクスをサポートしている．そのため，ユーザは，通常のファイルシステムと同様に扱え，既存のアプリケーションプログラムをそのまま利用できる．Gfarm 以外のシステムでは，広帯域・低遅延な単一組織内のクラスタコンピューティング環境を前提としており，広帯域・高遅延な広域分散環境へそのまま適用することはできない．

2.4.2　メタデータによるデータ検索

メタデータを用いてデータ検索を行い，検索結果から目的のデータへアクセスするシステムについて述べる．代表的なシステムに，Storage Resource Broker（SRB）[1)–3)]，iRODS[16)] がある．SRB は，San Diego Supercomputer Center（SDSC）で開発されたグローバル論理名とファイル階層をユーザに提供する論理的分散ファイルシステムである．SRB は，MCAT（Metadata CATalog）サービス，SRB サーバ，SRB クライアントから構成される．MCAT には，SRB で管理されるユーザやリソースに関するメタデータが保管され，SRB サーバからの問合せに応じて論理名と物理属性のマッピングやメタデータの作成，更新を行う．iRODS（integrated Rule-Oriented Data System）は，ルール指向のデータグリッドシステムで，ネットワークで接続された複数のストレージリソースを 1 つのファイルシステムとして提供する．ルールとは，ルールを実行するための条件と動作を設定することができる．条件は，時間，実行頻度，データオブジェクトの名前などを記述することができる．iRODS では，iCAT（iRODS metadata catalog）がアカウント，リソースの管理を行う．これらのシステムでは，データが持つ属性情報をメタデータとして管理し，ユーザの要求に応じてデータの検索結果を返す．iRODS は，BSD ライセンスのオープンソースであるが，SRB のすべてのソースコードは，アカデミック組織や，政府機関のみが利用可能である．また，SRB，iRODS ではシステムレベルメタデータと呼ばれるデータを管理するために必要なメタデータ，ユーザレベルメタデータと呼ばれるユーザが定義したアプリケーション用のメタデータを導入し，検索に利用している．メタデータに多くの属性情報を記述す

ることで，ユーザやコミュニティの共通の知識によってデータ共有・アクセスが可能である．これらのシステムでは検索結果からデータへアクセスするための専用コマンドや API，Web インタフェースが提供されている．SRB では，ユーザが定義したメタデータを手動，または，自動的に収集を行っている．しかし，アプリケーションレベルメタデータごとに，テーブルの定義が必要となり，複雑なシステムとなりやすい．

3. MetaFa の設計方針

本章では，メタデータ管理の要求要件の分析や関連研究の問題点をふまえ，広域分散環境において，データの鮮度を維持したまま，アプリケーションが持つセマンティクスの差異を吸収してメタデータを管理する手法など，MetaFa の設計方針について述べる．

3.1 MetaFa の概要

MetaFa（*Metadata administration Factory*）は，広域分散ネットワーク環境でメタデータの収集・管理を行うためシステムである．想定するグリッドコンピューティングのアプリケーション実行環境を**図 2** に示す．グリッドコンピューティングのアプリケーションの実行形態はユーザがアプリケーションのジョブをジョブスケジューリングノードへ投入する．

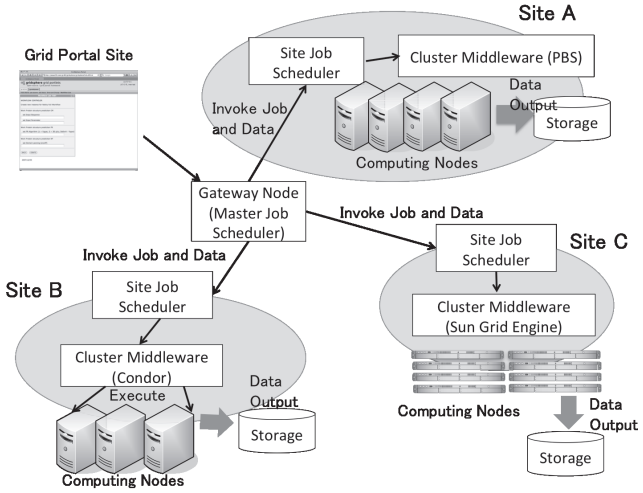

図 2　グリッドコンピューティングアプリケーション実行環境
Fig. 2　Grid computing application environments.

ジョブスケジューラは，VO（Virtual Organization）上のワーカノードへジョブを配布する．ワーカノードはジョブを実行し，処理結果をローカルディスクへ保存する．MetaFa は広域分散ファイルシステム，広域分散ストレージ上へ出力されるデータからメタデータを収集し，ユーザへメタデータを利用するインタフェースを提供する．

3.2 MetaFa の設計方針

MetaFa システムのアーキテクチャを**図 3** に示す．MetaFa では，クライアント・サーバモデルでメタデータ管理を行う．計算処理を実行するワーカノード上に生成されるデータからメタデータを取得するエージェントを配置する．ワーカノード上でメタデータを分散ハッシュテーブル（DHT）などを用いて管理することは可能である[18),20)]．ワーカノードの計算資源を最大限に活用するため，ユーザが実行するアプリケーションプログラム以外の処理を極力行わない方針をとる．そのため，ワーカノードで収集したメタデータを集中して管理するメタデータサーバを導入する．MetaFa は，ワーカノード上のエージェントプログラムがメタデータを収集し，メタデータサーバにメタデータを管理するクライアント・サーバモデルとした．

3.2.1 メタデータ

本項では，MetaFa で取り扱うメタデータを定義する．MetaFa では，データ管理用の基本メタデータと，ユーザが定義したアプリケーションメタデータの 2 種類のメタデータを

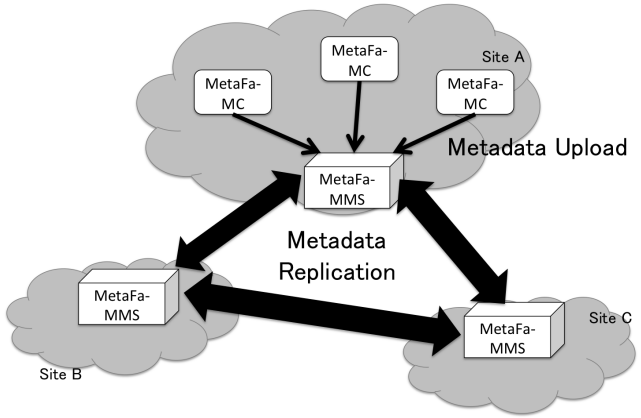

図 3　MetaFa システムアーキテクチャ
Fig. 3　MetaFa system architecture.

表 1　基本メタデータの要素一覧
Table 1　The elements of basic metadata.

基本メタデータの要素名	要素の説明
id	MetaFa システム内で一意な識別子
dataid	ワーカノード内で一意な識別子
inode	ファイルの inode
filename	ファイル名
filepath	ファイルパス（ファイル名は除く）
st_mode	ファイルのパーミッション
st_uid	ワーカノード上でのユーザ ID
uid_name	ワーカノード上でのユーザ名
st_gid	ワーカノード上でのグループ ID
gid_name	ワーカノード上でのグループ名
st_size	ファイルサイズ
st_atime	最終アクセス時刻
st_mtime	最終更新時刻
st_ctime	ファイル作成時刻
hash	MD5 で生成したファイルのハッシュ値
hostname	ワーカノードのホスト名

扱う．インターネット上の分散ストレージに存在するデータの管理を前提としているため，データの位置や保存しているデータへアクセス可能なプロトコルなどを管理する必要がある．グリッドコンピューティングで取り扱うデータは，ファイル形式が多いため，データを管理するために，ファイルが持つファイル属性を利用する．ファイル属性とは，i-node などファイルシステムが持つそのファイルについてのデータである．基本メタデータの要素を**表 1** に示す．基本メタデータの要素は，ファイル属性のほかに，ファイルのハッシュ値，データが保存されているホスト名などを持つ．アプリケーションメタデータは，データについてのアプリケーションレベルのセマンティクスを表現する．一般的なデータ形式の場合は，データ構造が既知であるので，アプリケーションメタデータのスキーマは決定的である．一般的なデータ形式とは，画像データや文書ファイルのことを意味する．文書ファイル，画像データのアプリケーションメタデータは既存プログラムによって，自動的に収集可能である．グリッドコンピューティングなどで実行するアプリケーションが生成するデータのメタデータは，アプリケーションを利用するユーザがメタデータのスキーマを決定し利用する．本論文においては，アプリケーションを利用しているユーザがアプリケーションメタデータのスキーマをすでに決定しているものとする．

3.3　メタデータ収集の自動化手法の方針

メタデータを自動的に収集する既存のアプローチとして，2 種類の方法がある．1 つめは，Google Desktop に代表されるデスクトップ検索である．これは，ファイルシステムを定期的にクローリングして，インデックスを構築し，ユーザが検索したキーワードにマッチするファイルを提示するシステムである．その特徴は，インデックス構築と同時にアプリケーション用のメタデータを取得することが可能であること，ファイルのオープン時や移動時などのタイミングで，リアルタイムにメタデータを取得することが可能であることがあげられる．しかしながら，ファイルの新規作成や更新が発生していない場合でも，ファイルシステムを定期的にクロールするため，大量のディスク I/O が発生する．また，Google Desktop はインデックス作成のために，CPU リソースをすべて使いきるという問題があった．2 つめは，Spotlight のように，ファイルの生成・更新などのイベントをメタデータ取得エンジンに通知し，通知されるとただちにメタデータを取得する手法である．Spotlight は，Mac OS X 独自のファイル検索システムであるが，この手法を応用することで，メタデータ取得のために CPU リソースの大量消費や大量のディスク I/O を発生させることなく，メタデータを取得することができる．MetaFa では，アプリケーションがファイル作成時に呼び出す open()/close() などのシステムコールが呼び出された際のファイルシステムイベントを取得し，即時にメタデータを取得する方法を採用する．生成されたデータから即座にメタデータを収集することで，データの鮮度を考慮したメタデータの管理が可能となる．

3.4　メタデータの管理方針

本節では，メタデータの管理方針について説明する．1 つめの方針は，アプリケーションメタデータを含めて，メタデータのスキーマを管理しない．メタデータのスキーマとは，メタデータとして記述する属性と属性値のデータ形式を表す．その管理方法は，個別のアプリケーションメタデータのスキーマをメタデータ管理システムへ登録して，定義されたアプリケーションメタデータのスキーマごとにデータベースのテーブルを作成し，管理する．しかしながら，この方法では，アプリケーションごとにテーブルが分離され，問合せのクエリをアプリケーションごとに識別しなければならない．そのため，1 つのシステム上で複数のアプリケーションのメタデータを管理することにより，システムが複雑化し，メタデータの管理コストが高くなる．グリッドコンピューティングでは，異なるユーザがワーカノードで複数のアプリケーションを実行するため，ワーカノードやメタデータを保持するメタデータ管理サーバに，実行する分だけのアプリケーションメタデータのスキーマを保持しなければならない．各サイトに設置されたメタデータ管理サーバに対して，アプリケーションメタデー

タのスキーマの数だけのデータベースもしくは，テーブルを構築するコストは，メタデータ管理システム側にとって大きい．また，ユーザが途中でアプリケーションメタデータのスキーマを変更した場合，存在するメタデータ管理サーバのすべてのデータベース，テーブルを修正する必要がある．メタデータをスキーマレスで管理する方法では，ユーザがメタデータのスキーマを途中で変更した場合でも既存のデータベースやテーブルを修正する必要がなく，変更前後のメタデータスキーマをそのまま利用できる．これらの理由から，MetaFa では，シンプルなアーキテクチャを保持したまま，複数のアプリケーションメタデータを管理する方法として，メタデータのスキーマレス管理手法を採用した．

2 つめの方針は，すべてのメタデータ管理サーバで，すべてのメタデータを保持する．これは，ユーザがどのサイトに所属していても，どのメタデータ管理サーバへ問い合わせても，同じ検索結果を返すためにすべてのメタデータを保持する．つまり，ユーザはどのメタデータ管理サーバに問い合わせても同一の結果を得ることができる．MetaFa では，ユーザに対して，メタデータ透過性を提供する．また，メタデータ管理サーバの持つデータ量は増えることになるが，メタデータ管理システム全体の耐障害性は強くなる．

4. MetaFa システムのプロトタイプ

提案する MetaFa の有効性を検証するために，MetaFa プロトタイプを Linux 上に Python を用いて実装した．本章では，MetaFa プロトタイプについて述べる．

4.1 MetaFa システムの概要

MetaFa システムは，Metadata Collector（MetaFa-MC），Metadata Management Server（MetaFa-MMS），MetaFa-Manager から構成される．MetaFa システムの基本コンポーネントを**図 4** に示す．詳細を次節以降で述べる．

4.2 MetaFa-MC

MetaFa-MC（Metadata Collector）は，ジョブを実行するワーカノードで動作するメタデータを収集する．MetaFa-MC は，ワーカノードでデーモンプロセスとして動作し，メタデータを収集し，MetaFa-MMS へアップロードする．ワーカノードで収集したメタデータは，MetaFa-MC でも保存する．MetaFa-MC は，アプリケーションがファイルを出力したとき，またはファイルが更新された際にメタデータを取得する．MetaFa-MC がファイルシステムイベントを取得し，メタデータを取得する一連の流れを**図 5** に示す．MetaFa-MC は，ファイルシステムイベントを inotify[6] と呼ばれる Linux カーネルの機能から取得する．ファイルシステム上でイベントが発生したイベントドリブン方式でメタデータを取得する．inotify

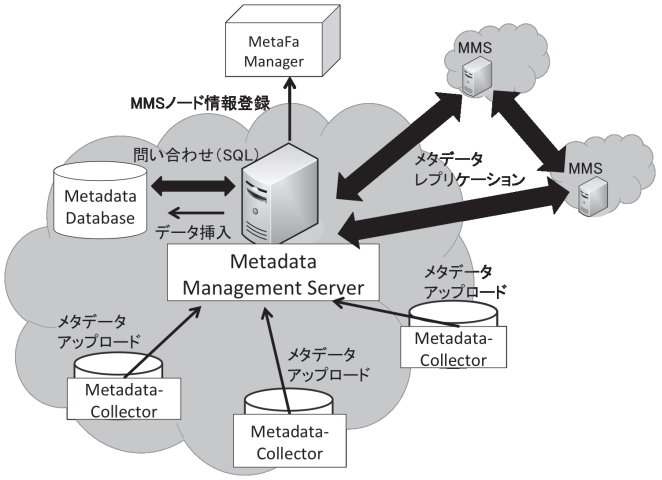

図 4　MetaFa コンポーネント
Fig. 4　MetaFa components.

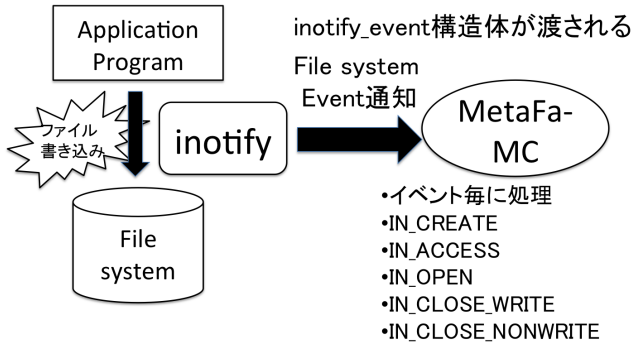

図 5　ワーカノード上での MetaFa-MC の挙動
Fig. 5　MetaFa-MC behavior on the worker nodes.

は，Linux カーネルが提供する機能で，ファイルシステムイベントを監視する．MetaFa-MC を起動すると，inotify インスタンスを作成し，監視対象のディレクトリを watch リストに追加する．監視対象のディレクトリは，MetaFa-MC の起動時に引数として渡す．あるいは，MetaFa-MC の設定ファイルに監視ディレクトリを記述する．inotify から受け取る

表 2　inotify イベント一覧
Table 2　The inotify events.

Event	Event の内容
IN_ACCESS	ファイルの読み込みが発生
IN_ATTRIB	メタデータ（i-node などのファイル属性）が更新
IN_CLOSE_WRITE	書き込みのためにオープンされたファイルがクローズ
IN_CLOSE_NOWRITE	書き込み以外のためにオープンされたファイルがクローズ
IN_CREATE	監視対象内でファイルやディレクトリが作成
IN_DELETE	監視対象内でファイルやディレクトリが削除
IN_DELETE_SELF	監視対象のディレクトリまたはファイルが削除
IN_MODIFY	ファイルが修正
IN_MOVE_SELF	監視対象のディレクトリまたはファイル自身が移動
IN_MOVE_FROM	ファイルが監視対象ディレクトリ外へ移動
IN_MOVE_TO	ファイルが監視対象ディレクトリ内へ移動
IN_OPEN	ファイルがオープン

ことができるイベントの一覧を**表 2** に示す．inotify インスタンス側で，inotify からどのイベントを受け取るかを指定できるため，MetaFa-MC は，IN_CREATE，IN_MODIFY，IN_MOVE_FROM，IN_MOVE_TO，IN_CLOSE_WRITE を受信するように設定してある．これらのイベントを受信することで，ファイルの作成，更新が起きたことを確認できる．IN_CLOSE_WRITE が発生し，ファイルの書き込みが終了するタイミングでメタデータの抽出を行う．

MetaFa-MC がメタデータを保存するためのメタデータデータベースに，メモリ上にデータベースを配置することができるため sqlite3 [11] を採用した．MetaFa では，メタデータデータベースをワーカノードのメモリ上に配置することで，メタデータを取得するときディスク I/O に与える影響を小さくでき，高速に処理できる．また，MetaFa は inotify からのファイルシステムイベントを用いてメタデータの生成・更新を行うことで，ファイルシステムをクロールしなくともメタデータデータベースの構築・維持が可能である．MetaFa-MC は，ファイルシステムイベントが発生した際にイベントごとにメタデータ取得関数を呼び出す．メタデータ取得関数は，IN_CREATE，IN_MOVE_FROM イベントが発生した際には，メタデータを新規に取得し，メタデータデータベースへ格納する．また，IN_MODIFY，IN_MOVE_TO イベントが発生した際は，メタデータの更新を行う．更新の際は，i-node をキーとしてローカルメタデータデータベースへアクセスする．MetaFa-MC デーモンは，IN_CLOSE_WRITE イベントを受け取ると，ローカルメタデータデータベースに対して，COMMIT コマンドを発行する．複数のファイルシステムイベントを受信し，イベントごとに対応することでデータの変更を即時に検知することができる．収集したメタデータは，MetaFa-MMS へ定期的にアップロードする．現在の実装では，120 秒間隔で MetaFa-MMS へメタデータをアップロードしているが，アップロード間隔は自由に変更可能である．

MetaFa-MC でのアプリケーションメタデータの取得方法について説明する．はじめに，アプリケーション固有のメタデータを抽出するために，データの拡張子を確認する．データの拡張子が，あらかじめ MetaFa-MC に登録されいてる場合，拡張子に合わせてアプリケーションメタデータを抽出するためのプログラムを呼び出す．たとえば，拡張子が JPEG などの画像ファイルでは，EXIF 情報を表示する exif プログラムを呼び出す．exif プログラムが出力した結果を解析し，メタデータデータベースへ格納する．すなわち，拡張子が既知であり，抽出プログラムが存在するものは MetaFa-MC に登録しておき，生成されたデータの拡張子で，どのメタデータ抽出プログラムを呼び出すかを判断する．また，ユーザ独自に定義されたメタデータを抽出する場合は，MetaFa-Manager の Web インタフェースから，ユーザが扱うデータの拡張子とメタデータを抽出するプログラムを登録する．MetaFa-Manager は，登録された情報とプログラムをワーカノード側へ配布する．ただし，現在の実装では，拡張子が重複していた場合，正しくアプリケーションメタデータを抽出することはできない．このほかにも，拡張子がないデータからはアプリケーションメタデータを抽出することはできない．

4.3　MetaFa-Manager

MetaFa-Manager は，メタデータ管理サーバである MetaFa-MMS の情報を管理する．後述する MetaFa-MMS が，起動時に MetaFa-Manager へ接続する．このとき，MetaFa-MMS はホスト名と XML-RPC サーバのポート番号を MetaFa-Manager へ登録する．MetaFa-Manager は，すでに登録されている MetaFa-MMS ノードのリストを接続してきた MetaFa-MMS へ返す．また，同時に MetaFa-Manager は自身が管理する MetaFa-MMS ノードリストが更新されると，すでに接続してきた MetaFa-MMS へノードリストが更新されたことを通知する．

MetaFa-MMS が故障した場合には，MetaFa-MC はアップロードする先が消失する．そのため，MetaFa-MC は，MetaFa-MMS へのメタデータアップロード処理が 2 回失敗すると，MetaFa-Manager へ問合せを行う．このとき，MetaFa-Manager は，代替の MetaFa-MMS を MetaFa-MC へ通知する．

4.4　MetaFa-MMS

MetaFa-MMS（Metadata Management Server）は，MetaFa-MC で収集したメタデー

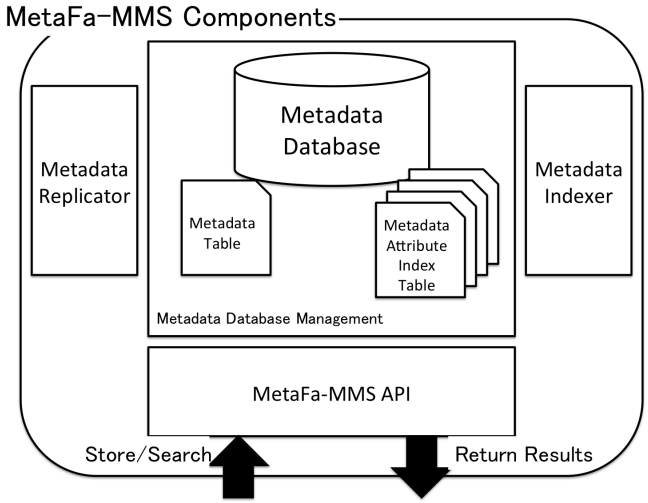

図 6　MetaFa-MMS コンポーネント図
Fig. 6　MetaFa-MMS components.

タの管理サーバである．MetaFa-MMS のコンポーネントを**図 6** に示す．MetaFa-MMS は，ユーザが問い合わせるための API（get），MetaFa-MC がメタデータをアップロードするための API（put）を提供する．さらに，他の MetaFa-MMS へ複製する Metadata Replicator，メタデータを Python の辞書型データで格納しているため，検索用にインデックスを作成する Metadata Indexer を実装した．なお，MetaFa-MMS は各組織に 1 台以上設置する．

4.4.1　RDBMS によるスキーマレスメタデータ管理

RDBMS で，データを取り扱うためには，格納するデータのスキーマが必要となる．MetaFa では，メタデータのスキーマを管理せずに，RDBMS でメタデータを取り扱うために，メタデータをシリアライズして Python の辞書型データとして扱う．シリアライズしたメタデータは，{'key1':'value1', ...} という構造をしている．1 つのメタデータに対して，(id，dataid，hostname，metadata_item）を 1 レコードとして RDBMS に格納する．id は，MetaFa システムにおける主キー，dataid は各ワーカノード上でデータを識別するための id，hostname はワーカノードのホスト名，もしくは IP アドレス，metadata_item にシリアライズされたメタデータを格納する．MetaFa-MMS は，metadata_item を文字列として扱うため，個々の要素のデータ型を意識する必要はない．MetaFa-MMS プロトタイプ実装では，RDBMS として PostgreSQL を採用した．

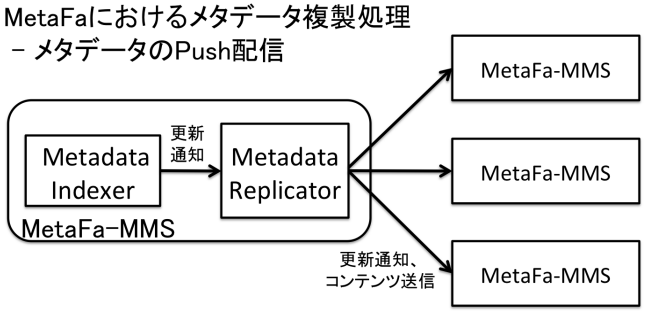

図 7　MetaFa-MMS のメタデータ複製機能
Fig. 7　Metadata replication function in MetaFa-MMS.

4.4.2　Metadata Indexer

MetaFa-MMS では，シリアライズしたメタデータを文字列として扱う．メタデータが文字列のままでは，属性と属性値による検索が SQL ではできないため，SQL で検索可能な形式に変換する必要がある．Metadata-Indexer は，MetaFa-MMS で，辞書型データのメタデータからメタデータインデックスを生成するためのプログラムである．Metadata-Indexer は，追加されたメタデータが存在しているかどうかを定期的に確認する．追加されたメタデータが存在した場合には，metadata_item を解釈し，それぞれキー（属性名）とバリュー（属性値）の組に分割する．さらに，それぞれのキーとバリューの組を，キーごとに作成したインデックステーブル（属性テーブル）に，バリューとメタデータの ID を格納する．

4.4.3　Metadata Replicator

MetaFa-MMS は，管理下にある MetaFa-MC からアップロードされるオリジナルのメタデータを管理する．各 MetaFa-MMS は，MetaFa システム内で管理するメタデータのすべてを保持する．そのため，オリジナルのメタデータを保有する MetaFa-MMS は，インデックス作成後に，他の MetaFa-MMS にメタデータを複製する．MetaFa-MMS はメタデータインデックスの作成後，Metadata Indexer から MMS-Replicator に対して，複製するメタデータ ID の範囲を知らせる．MMS-Replicator は，範囲内にあるメタデータをデータベースから読み込み，他の MetaFa-MMS に対してプッシュ配信を行う．MetaFa-MMS がメタデータを複製する流れを**図 7** に示す．MetaFa-MMS は受信したメタデータをメタデータデータベース内の replica テーブルに格納する．複製されたメタデータは，Metadata

Indexer によってオリジナルのメタデータと同様に，インデックスを作成する．

5. MetaFa の実験・評価

MetaFa の有効性を示すため，MetaFa システムをキャンパスネットワーク，広域分散ネットワークテストベッドである PlanetLab 上に展開し，以下の 3 つの実験を行った．次節以降，詳細を述べる．

- メタデータ収集処理におけるファイル I/O 性能に対する影響の調査
- メタデータのスキーマ管理手法とスキーマレスメタデータ管理手法の比較
- プッシュ配信によるメタデータ複製のデータ増加，ノード増加に対するスケーラビリティの確認

5.1 実験 1—ファイル I/O スループットへの影響

MetaFa システムでは，メタデータを即時に取得するため，open()/close() などのシステムコールが呼ばれたなどのファイルシステムイベントを受信した瞬間にメタデータを収集する．性能評価指標の 1 つとしては，ファイル I/O 性能がある．メタデータ収集処理におけるファイル I/O 性能に対する影響を調べるために 2 種類の実験を実施した．1 つめは，ファイルシステムベンチマークを実施し，MetaFa-MC の有無でファイル I/O スループット，特に，書き込み処理に対して，どのような影響があるかを調査した．2 つめは，アプリケーションメタデータを取得する際の MetaFa-MC のオーバヘッドを調査するためファイル I/O スループットを計測した．

5.1.1 iozone によるファイルシステムベンチマーク

ワーカノード上で MetaFa-MC を動かした場合，動かしていない場合で，iozone [14] ファイルシステムベンチマークを用いて実験を行った．アプリケーションメタデータの収集方法は，アプリケーションごとに異なり，データのヘッダ部分を読み取るものや，データ全体を解析するものなど多岐にわたる．ターゲットとするアプリケーションによって，メタデータ取得処理の内容が変わるため，本実験では MetaFa の基本性能を測定するために，基本メタデータのみを収集した．2 台の性能の異なるワーカノード上で iozone ベンチマークを行った．iozone ベンチマークは，ファイルサイズを 64 KB から 4 GB，レコード長を 16 KB から 16 MB の範囲で実行した．なお，本実験時には，メタデータのアップロードは行わず，ワーカノード上のメタデータデータベースへの保存のみを行った．

表 3 に示す異なる仕様のノード A，B 上のローカルファイルシステムに対して，MetaFa-MC を動作させた場合，動作させない場合で iozone ベンチマークを実行した．実験結果を

表 3　ワーカノードの仕様
Table 3　Specification of worker nodes.

ノード A	Intel Pentium 4 3.40 GHz, Mem: 2 GB, OS: Ubuntu 10.04 HDD: HITACHI HUA722020ALA330 2 TB, 7,200 rpm, Filesystem: ext4
ノード B	Intel Core i7 3.2 GHz, Mem: 12 GB OS:Ubuntu 9.04 (x86_64) HDD: HITACHI HUA722020ALA330 2 TB, 7,200 rpm, Filesystem: ext4
ノード C	Intel Xeon 3.6 GHz, Mem: 2 GB, OS: Ubuntu 9.04 HDD: Maxtor 6L160M0 160 GB, 7,200 rpm, Filesystem: ext3
ノード D	Sun UltraSPARC T2 1.4 GHz, Mem: 64 GB, OS: Solaris 10 HDD: SAS 146 GB, FC: SAS300 GB, 150,000 rpm, Filesystem: ZFS
ノード E	Intel Pentium 4 3.06 GHz, Mem: 2 GB, OS: Ubuntu 8.10 HDD SEAGATE ST3120026A 120 GB U100 7,200 rpm, Filesystem: ext3

図 8 に示す．横軸は生成したファイルサイズ (KB)，縦軸はファイルを生成した際の I/O スループット (MB/sec) を表す．図 8 より，ノード A，B ともにどのファイルサイズにおいてもファイル書き込み I/O 性能に大きな影響を与えていない．

図 8 で示したベンチマーク結果と，NFS 上で MetaFa-MC を動作した場合，動作させない場合と Gfarm 上で実行した iozone ベンチマークを実行した．本実験で用いたネットワーク環境を**図 9** に示す．実験に用いた各ノードのスペックを表 3，各ノード間の RTT を**表 4** に示す．まず，NFS サーバとして，/data ディレクトリをエクスポートしたノード C，/zfs-FC ディレクトリをエクスポートしたノード D を用いた．ノード A，B で，ノード C，D がエクスポートした領域をそれぞれ NFS マウントして MetaFa-MC を動作させた場合と動作させない場合で iozone ベンチマークを実施した．さらに，ノード E とノード C を用いて Gfarm の環境を構築した．ノード E で，Gfarm のメタデータサーバ（gfmd）を起動し，ノード C で Gfarm のストレージノード（gfsd）を起動した．gfsd で，ノード C 上の/data を Gfarm から利用できるように設定した．ノード C で，/home/gfuser ディレクトリ以下に，gfarm2fs で/data ディレクトリをマウントし，iozone ベンチマークを実行した．NFS，および Gfarm 上での実験構成を**図 10** に示す．これらのベンチマーク結果，および，ローカルファイルシステム上でのベンチマーク結果を**図 11** に示す．図 11 からも，ファイル書き込み性能において大きな差は見られなかった．すなわち，NFS においても MetaFa-MC が動作することを確認した．

本実験より MetaFa-MC は，ファイルシステムの書き込み性能に対して大きな影響を与えていないことが判明した．同時に，各ファイルサイズにおいてメタデータをすべて取得していることを確認した．MetaFa-MC では，メタデータのとりこぼしを発生させず，ファイ

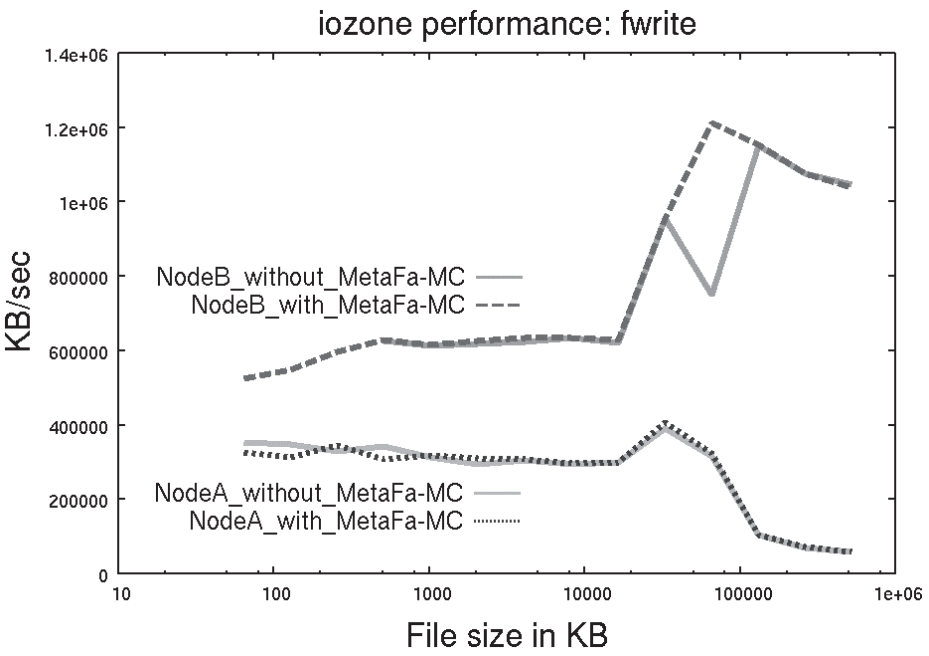

図 8　MetaFa の有無によるファイル書き込み性能の比較
Fig. 8　Comparison of the File I/O (WRITE) performance with/without MetaFa-MC.

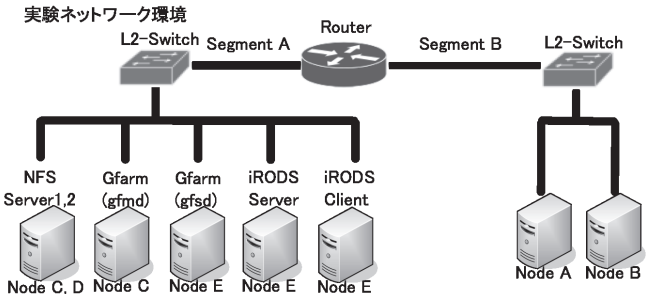

図 9　ファイルシステムベンチマーク実験環境（1）
Fig. 9　Filesystem benchmark environments (1).

表 4　ファイル I/O スループット実験における各ノード間の RTT
Table 4　The average RTT between each nodes.

Node A（NFS クライアント）– Node C（NFS サーバ）	0.171 msec
Node B（NFS クライアント）– Node C（NFS サーバ）	0.126 msec
Node A（NFS クライアント）– Node D（NFS サーバ）	0.265 msec
Node B（NFS クライアント）– Node D（NFS サーバ）	0.276 msec
Node C（gfmd）– Node E（gfsd）	0.173 msec
Node E（iRODS/MCAT サーバ）– Node E（iRODS クライアント）	0.111 msec

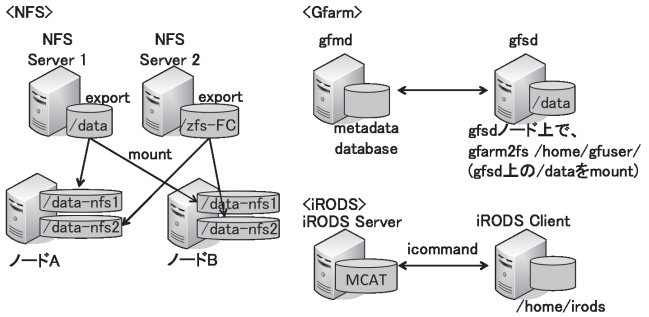

図 10　ファイルシステムベンチマーク実験環境（2）
Fig. 10　Filesystem benchmark environments (2).

ル書き込み性能に大きな影響を与えずにメタデータの取得が可能であることを示した．

5.1.2　アプリケーションメタデータ取得時のオーバヘッド計測

次に，アプリケーションメタデータ取得におけるオーバヘッドを計測した．本実験では，画像データ（JPEG ファイル）から基本メタデータとアプリケーションメタデータとして EXIF 情報を取得した．EXIF 情報は，画像ファイルのメタデータフォーマットであり，デジタルカメラの画像用メタデータとして利用されている．EXIF は属性として撮影日時，画像の解像度などの情報を持つ．このとき，MetaFa-MC は，作成したファイルを拡張子（JPEG，もしくは，JPG）で画像ファイルとして判断し，アプリケーションメタデータを exif コマンドで取得した．実験のパラメータとして，生成するデータのデータ数とデータサイズを変化させた．まず，80 KB の画像ファイルを 10^N（$N = 1$，2，3，4，5，6）個のデータをコピーする際に，MetaFa-MC を起動して，メタデータを取得した場合とメタデータを取得しない場合で，ファイル I/O のスループットを 3 回ずつ計測した．さらに，同様の画像ファイルのコピー処理を NFS，Gfarm，iRODS 上で行った．NFS，Gfarm，iRODS 上で行っ

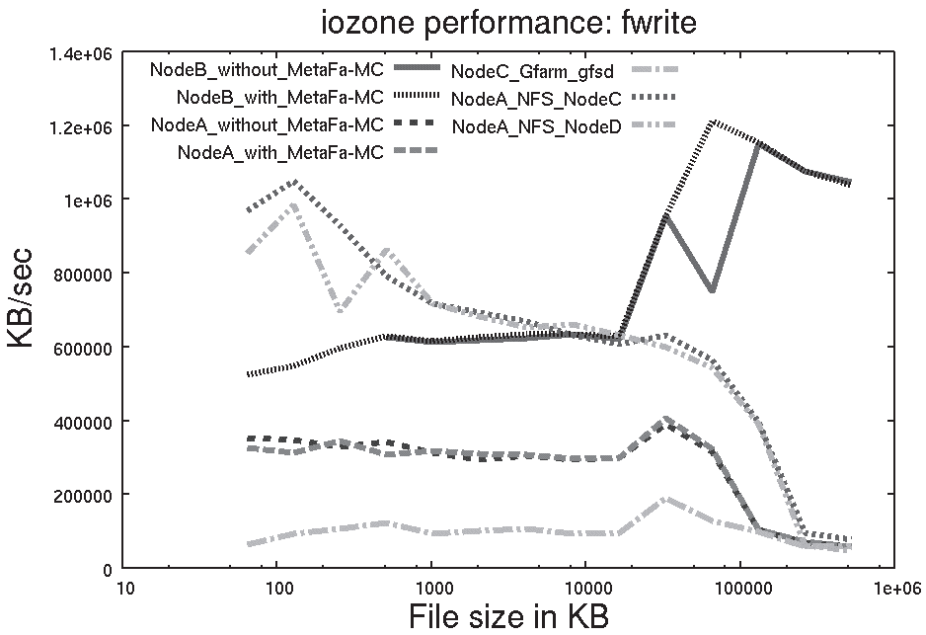

図 11　MetaFa と NFS，Gfarm によるファイル書き込み性能の比較
Fig. 11　Comparison of the File I/O (WRITE) performance with/without MetaFa-MC, NFS and Gfarm.

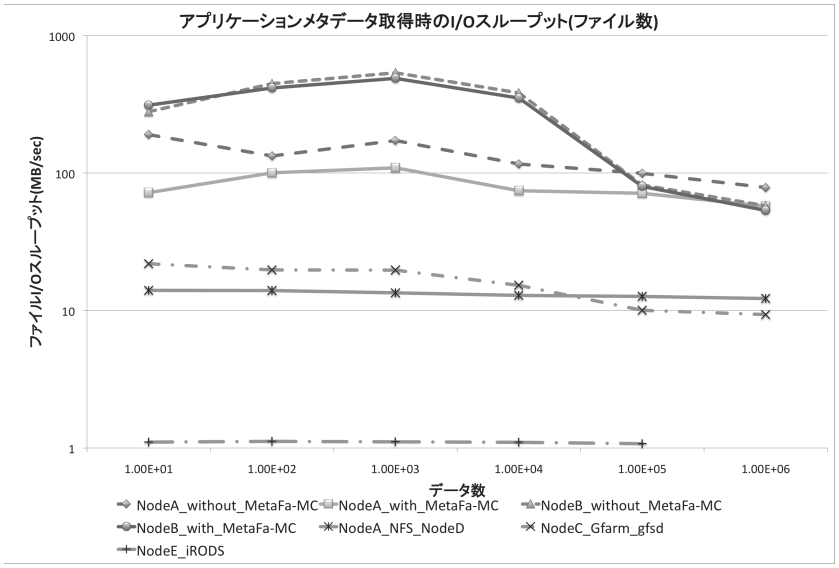

図 12　アプリケーションメタデータ取得時のスループットの比較（ファイル数）
Fig. 12　Comparison of the File I/O throughput with/without MetaFa-MC when acquring application metadata (File number).

た実験では，MetaFa-MC によるメタデータの取得は行っていない．実験環境は，表 3 に示したノードを用いた．NFS，Gfarm を構築した環境は，図 9 に示した実験構成と同様である．iRODS は，ノード E 上にメタデータを管理する MCAT サーバおよび，ストレージノードを構築した．iRODS システムを利用するため，iRODS が提供するコマンドラインインタフェース icommand を用いた．iRODS システムへのデータ登録（生成）は，icommand の iput コマンドを使用した．iRODS システムの実験構成を図 10 に示す．本実験の結果を**図 12** に示す．横軸は生成するデータ数，縦軸はファイル生成時のスループット (MB/sec) を示す．本実験では，MetaFa-MC とネットワークファイルシステム（NFS，Gfarm）を比較した．実験結果より，ネットワークファイルシステムとローカルファイルシステム上で MetaFa-MC を動作させた場合のファイル I/O スループットの差は明らかである．本実験で使用したファイルは，80 KB と比較的小さなファイルサイズであり，ネットワークファイルシステムでは書き込み処理やメタデータ処理が頻発して発生しているため，ファイルサーバやメタデータサーバへの通信が必然的に発生するのに対し，MetaFa では，ローカルファイルシステムに対して I/O 操作を行い，I/O 操作後にメタデータを抽出し，120 秒ごとに MetaFa-MMS へアップロードするため，I/O 操作時に通信は発生しない．NFS や Gfarm と MetaFa におけるファイル I/O におけるセマンティクス（更新伝播の迅速さ，i-node などのファイル属性書き込み，ファイル一貫性保持処理など）が大きく異なるため，NFS や Gfarm と MetaFa ではオーバヘッドが生じる原因が大きく異なる．そのなかで，MetaFa-MC を動作させた場合でもマシン性能の違いによって，スループットに影響はあるものの最低でもスループットの 8 割は達成できたことを示した．

次に，同一の実験環境で，生成するトータルのデータサイズを，10 MB，100 MB，1 GB，10 GB と変化させ，ファイル I/O スループットを 3 回計測した．このとき使用した平均データサイズは 5 MB である．実験結果を**図 13** に示す．横軸は，生成するトータルのデータサイズ（MB），縦軸はファイル生成時のスループット（MB/sec）を示す．本実験において

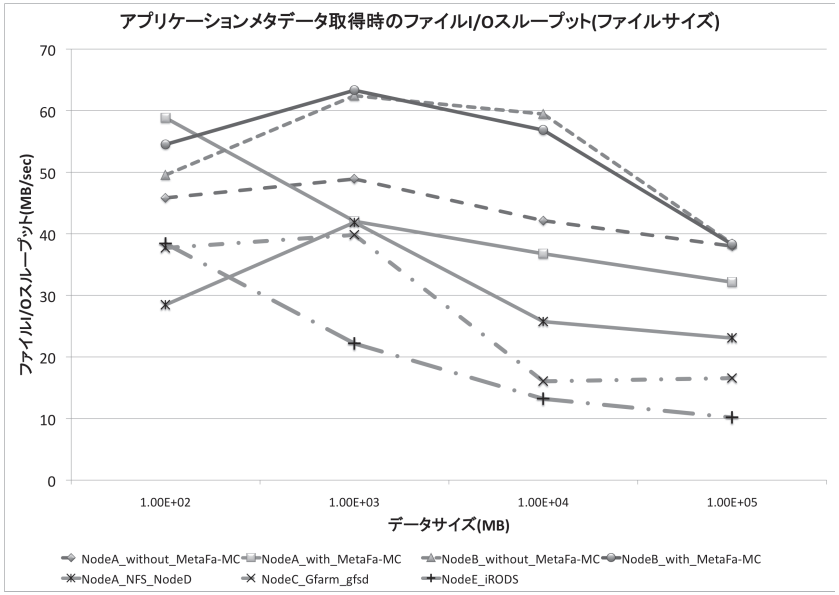

図 13　アプリケーションメタデータ取得時のスループットの比較（ファイルサイズ）
Fig. 13　Comparison of the File I/O throughput with/without MetaFa-MC when acquring application metadata (File size).

も，MetaFa-MC を動作した場合，NFS などのシステムと比較して数倍のスループットを維持している．MetaFa-MC を動作させない場合と比較したとき，生成するトータルのデータサイズが小さい場合に，スループットに大きく差がでている．このとき，実験を行ったノード A 上の top コマンドにて，プロセスの状況を確認したところ，Python のプロセスが 100%に近い CPU 利用率であった．大量のデータが発生し，メタデータを取得しようとした際にこのような現象が発生していることを確認した．

5.2　実験 2—スキーマレスメタデータ管理手法

MetaFa-MMS におけるスキーマレスメタデータ管理手法の効果を評価するために，メタデータのスキーマを管理した場合と管理しない場合での MetaFa-MC からの平均メタデータアップロード時間を比較した．本実験では，キャンパスネットワーク内に設置した複数の MetaFa-MC から MetaFa-MMS 上へメタデータをアップロードした．本実験時には，他の MetaFa-MMS へのメタデータ複製処理は行わない．MetaFa-MMS に，メタデータスキーマを管理するデータベースとスキーマを管理しないデータベースを用意した．メタデータのスキーマを管理する場合は，アプリケーションメタデータごとにスキーマを指定しテーブルを作成した．実験には，キャンパスネットワークに設置した 40 台のワーカノードと 1 台のメタデータ管理サーバを用いた．それぞれのノードの仕様を**表 5** に示す．

表 5　ワーカノードとメタデータ管理サーバのマシン仕様
Table 5　Specification of worker nodes and MetaFa-MMS server.

ワーカノード F	Intel Xeon Quad-core X5570 (2.93 GHz) x2, 24 GB DDR3-1333, CentOS 5.4 (x86_64), HITACHI H103030SCSUN300G SAS 300 GB 10,000 rpm x4
メタデータ管理サーバ	Intel Xeon L5520 (2.26 GHz), 12 GB DDR3-1333/PC3-10600, Ubuntu 9.04 (x86_64) HITACHI HUA7250S SATA 500 GB 7,200 rpm

キャンパスネットワークに設置した 10，20，30，40 台の MetaFa-MC から同じネットワークに設置した 1 台の MetaFa-MMS へメタデータをアップロードした．メタデータ管理サーバとワーカノード間の RTT は，0.227 ミリ秒であった．本実験で扱うメタデータは，画像の EXIF 情報，タンパク質立体構造予測アプリケーションのメタデータ，Web サーバのログと基本メタデータのみの 4 種類である．1 台の MetaFa-MC は，異なる種類のメタデータを 1,000，2,000，3,000，4,000 個アップロードする．本実験では MetaFa-MC に蓄積していたメタデータを読み出してアップロード処理を行う．

メタデータのスキーマを管理しない場合と，管理する場合のシステムの概要を**図 14** に示す．スキーマを管理する方法では，メタデータデータベース上にアプリケーションメタデータ用に 4 つのテーブルと，それぞれのアプリケーションプログラムと MetaFa システム内部でアプリケーションプログラムを識別するための applicaiton_id を管理するためのテーブルを用意した．MetaFa-MMS 上でスキーマを管理する場合，MetaFa-MC に蓄積するメタデータでもスキーマを管理する必要があるため，MetaFa-MC 側でもスキーマを管理する実装を行った．MetaFa-MC 側では，メタデータのスキーマを管理するためにメモリ上に配置した sqlite3 データベース内に，アプリケーションプログラムと MetaFa システム内部でアプリケーションプログラムを識別するための application_id を管理するテーブルと，それぞれのアプリケーションメタデータ管理テーブルを用意した．

まず，MetaFa-MC 側は，メタデータのスキーマを管理するデータベースとメタデータのスキーマを管理しないデータベースから 1,000，2,000，3,000，4,000 個のデータをランダムに読み出す．このとき，メタデータのスキーマを管理する場合，管理しない場合での

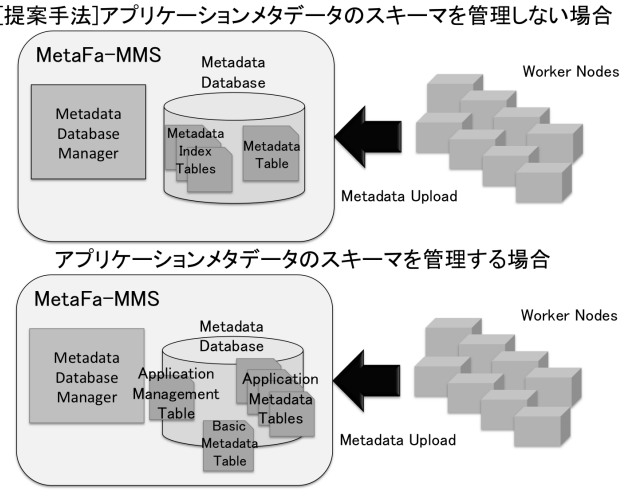

図 14　アプリケーションメタデータのスキーマ管理システム
Fig. 14　Application metadata schema management system.

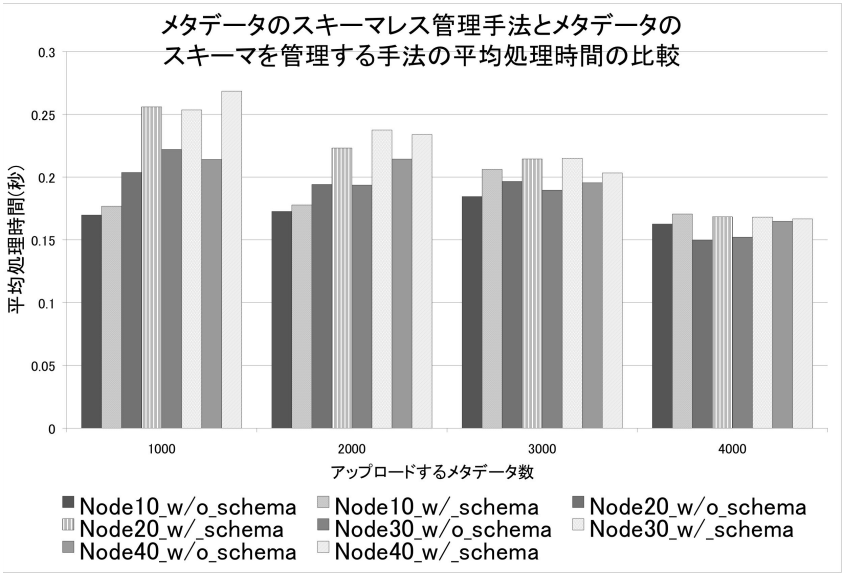

図 15　メタデータのスキーマレス管理とスキーマを管理するシステムの比較
Fig. 15　Comparison of metadata management with/withtout application schemas.

MetaFa-MC におけるメタデータデータベースからメタデータの読み込み処理速度を計測した．メタデータデータベースからは 4 種類のアプリケーションメタデータがランダムに読み出す．スキーマを管理する手法では，4,000 個のデータ読み込み時間は，平均 11.65 秒であった．スキーマを管理しない方法では，4,000 個のデータ読み込み時間は平均 4.64 秒であった．スキーマを管理する手法では，アプリケーションメタデータのスキーマをテーブルから読み出し，そこから基本メタデータとアプリケーションメタデータを読み出す．一方，提案手法では 1 つのテーブルからシーケンシャルに基本メタデータとアプリケーションメタデータを読み出すことができる．MetaFa-MC 側から MetaFa-MMS に対してメタデータをアップロードするために，メタデータをメタデータデータベースから読み込む際にもスキーマを管理しない手法が高速に処理を行うことができ，ワーカノードに与える負荷も低く抑えることができる．

MetaFa-MC は，XML-RPC プロトコルで MetaFa-MMS の put() API を呼び出し，メタデータをアップロードする．このとき，MetaFa-MC がメタデータをアップロードする平均時間を計測した．実験結果を**図 15** に示す．横軸は 1 台の MetaFa-MC からアップロードするメタデータの数，縦軸は 1 つのメタデータをアップロードする平均処理時間（秒）を示す．実線で示したグラフが，提案手法であるメタデータのスキーマレス管理手法，破線で示したグラフがメタデータのスキーマを管理する方法である．図 15 より，スキーマを管理しない提案手法がスキーマを管理する手法よりも高速にメタデータのアップロード処理が可能であることを確認した．

5.3　実験 3—広域ネットワークでのメタデータ複製におけるスケーラビリティ

MetaFa-MMS 間でメタデータ複製を行った際の全ノードへ複製処理が完了するまでの時間を計測した．N 台の MetaFa-MMS が存在するとき，オリジナルのメタデータを持つ MetaFa-MMS から他のサイトに存在する $N-1$ 台の MetaFa-MMS へメタデータを複製する．本実験で用いたノードは，実験 (2) で用いた表 5 のワーカノード F である．メタデータ複製元ノードは，実験 (2) で用いた表 5 のメタデータ管理サーバである．複製するメタデータの数を 10^N（$N = 2, 3, 4, 5, 6$）と変化させながら，複製先ノード数を 10，20，30，40 台と変化させ，MetaFa-MMS へ複製が完了するまでの時間を 3 回計測した．複製元の MetaFa-MMS と複製先の MetaFa-MMS 間の RTT は，0.227 ミリ秒であった．実験結

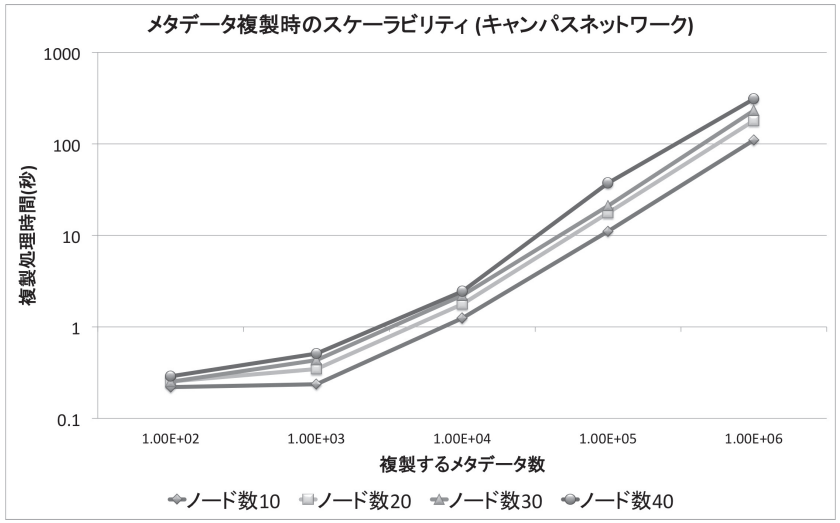

図 16　メタデータ複製時のスケーラビリティ（キャンパスネットワーク）
Fig. 16　Scalability of metadata replication on the campus network.

果を**図 16** に示す．横軸は複製したデータ数，縦軸は複製処理時間（秒）を示し，縦軸は対数表示である．図 16 より，データ数，ノード数の増加とともに，複製時間が単調増加していることが分かる．データ数が 10^5 個に，どのノード数の場合も 100 秒以内に複製処理を完了している．MetaFa プロトタイプでは，120 秒ごとに MetaFa-MC から MetaFa-MMS へメタデータをアップロードする．MetaFa-MC が 1 度にアップロードするメタデータの数が，10^5 個程度までであれば，複製処理と MetaFa-MC からのアップロード処理が時間差で行うことができるため，2 つの処理が重なることはないものと考えられる．

さらに，キャンパスネットワーク間でのメタデータ複製処理の性能を確認するために，日本国内に設置されている PlanetLab ノードを用いて実験を行った．PlanetLab[5] とは，プリンストン大学を中心として，次世代インターネットや分散アプリケーションなどの研究を行うためのインターネット上の地球規模のテストベッドである．PlanetLab では研究プロジェクトごとに，Linux-VServer が提供される．オリジナルのメタデータを持つ MetaFa-MMS を，オリジナルのメタデータを持つ MetaFa-MMS を，キャンパスネットワークの DMZ ネットワーク[⋆1]に設置し，日本国内の 20 台の PlanetLab ノードに対してメタデータ複製処理を行った．本実験では，ノード数を ping で RTT を計測し，RTT の小さいノードから 20 台のうち 5，10，15，20 台と変化させた．PlanetLab が設置しているノードに求める最低限のスペックを**表 6** に示す．しかしながら，各組織で，PlanetLab ノードの仕様は異なるため，実験に使用したすべてのノードがこの仕様を満たしているわけではない．複製元となる MetaFa-MMS から PlanetLab ノードに対する平均 RTT を**表 7** に示す．

表 6　PlanetLab ノードの仕様要件
Table 6　Minimum hardware requirements for planetlab nodes.

CPU	Intel cores x4 @ 2.4 GHz
RAM	4 GB
Disk	500 GB

表 7　PlanetLab ノードへの平均 RTT
Table 7　The average of RTT to the PlanetLab nodes.

ノード数	平均 RTT(ミリ秒)
5	0.696 ミリ秒
10	3.372 ミリ秒
15	5.447 ミリ秒
20	9.761 ミリ秒

先の実験と同様に，複製するメタデータ数を 10 倍ずつ増加させた．この実験では，複製するメタデータの数を最大 10^5 個とした．これは，PlanetLab ノード上で，1 GB 以上のメモリを使用した場合，強制的にプロセスを終了させられる制約により複製する上限個数を決定した．

実験結果を**図 17** に示す．キャンパスネットワークで行った実験結果と同様に，複製するデータ数の増加とともに処理時間は増加している．PlanetLab ノード 5 台と PlanetLab ノード 10 台の処理時間を比較すると，5 台の処理時間が 10 台の処理時間よりも大きくなっている．5 台の PlanetLab ノードと 10 台の PlanetLab ノードのロードアベレージを調べたところ，5 分間平均のロードアベレージの平均が，それぞれ 10.21，6.70 と 5 台の PlanetLab ノードのロードアベレージが高いことが分かった．PlanetLab ノードは常時，負荷が高く，

⋆1 DMZ（DeMilitarized Zone）ネットワークとは，ファイアウォールの外側に設置されたネットワークを意味する．

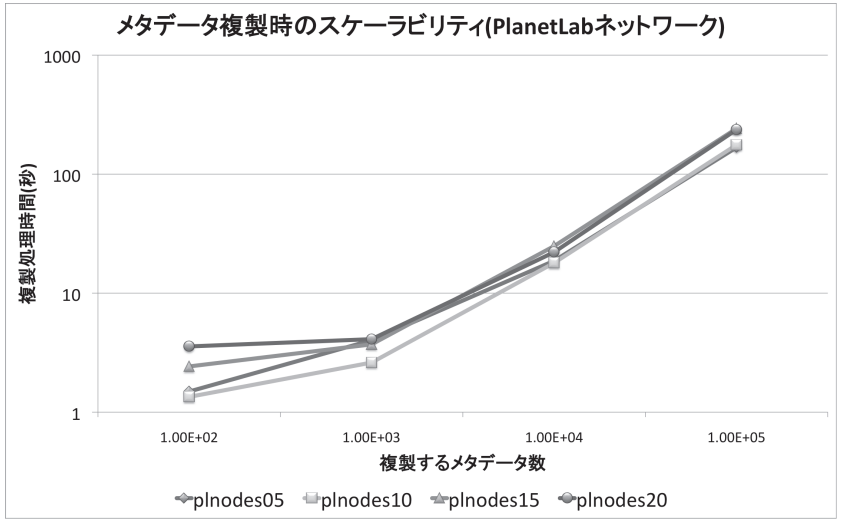

図 17　メタデータ複製時のスケーラビリティ（PlanetLab ネットワーク）
Fig. 17　Scalability of metadata replication on the PlanetLab network.

実験を行った 20 ノードの 5 分間の平均ロードアベレージの平均は，8.52 であった．5 分間の平均ロードアベレージの最大値は，29.94，最小値は，0.41 であった．PlanetLab ノードの平均負荷が高いため，ネットワーク遅延が処理時間に与える影響は大きいとは一概にはいえない．

負荷が低い PlanetLab ノードを利用することは難しいため，キャンパスネットワークに設置したノードの間に，Dummynet を設置し，擬似インターネット環境を構築した．この実験では，ロードアベレージが低いノードに対して，RTT を増加させた場合でのメタデータ複製処理を行い，ネットワーク遅延が複製処理時間に与える影響を調べた．本実験では，8 台のノードに対してメタデータの複製を行った．構成を**図 18** に示す．メタデータの複製元の MetaFa-MMS と 8 台の MetaFa-MMS の間に，Dummynet を設置した．2 つの NIC（em0，em1）をブリッジ接続として設定した．Dummynet では，ipfw コマンドで em0 から em1，em1 から em0 を通過するパケットに対してそれぞれ，5 ミリ秒，25 ミリ秒，50 ミリ秒，100 ミリ秒を設定した．これにより，RTT が 10 ミリ秒，50 ミリ秒，100 ミリ秒，200 ミリ秒となるようにした．複製するメタデータの数は，10^N（$N = 2$，3，4，5，6）個

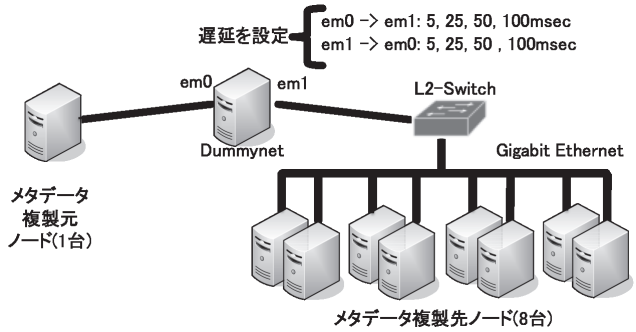

図 18　Dummynet を用いた実験構成
Fig. 18　Experiment environments using Dummynet.

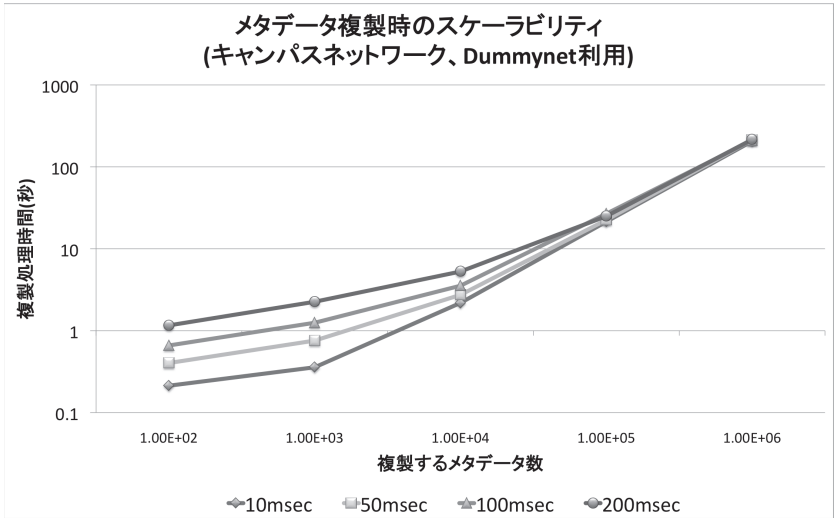

図 19　メタデータ複製時のスケーラビリティ（Dummynet）
Fig. 19　Scalability of metadata replication on the campus network using Dummynet.

とした．実験の結果を**図 19** に示す．横軸は複製するメタデータ数，縦軸は，複製処理時間（対数表示）を示す．これまでの実験と同様に，複製するデータ数を増加させると，複製処理時間が増加している．複製するメタデータの数が少ない場合（10^2，10^3，10^4）は，RTT

の影響を受けて複製処理時間が増加している．しかしながら，複製するメタデータの数が多い場合（10^5，10^6）は，RTT の影響をほとんど受けていないことが分かる．これは，複製するメタデータの送信時間よりも，MetaFa-MMS がメタデータを受信してから，メタデータデータベースへ格納する時間が大きいため，RTT の影響をほとんど受けていないものと考えられる．MetaFa-MMS は負荷が低い状態で，複製するメタデータの上限数を 10^5 程度にすれば，RTT の影響を大きく受けない．

6. 考　　察

本章では，5 章で述べた実験結果から MetaFa の効果について，特に MetaFa システムの性能面から考察する．はじめに，MetaFa が I/O 性能に与える影響について述べる．iozone や画像データの生成などの処理を行うことによって，MetaFa-MC がファイルの書き込み性能に対して，大きな影響を与えていないことを確認した．また，性能の異なるマシンのローカルファイルシステムにデータを出力しても，MetaFa-MC はファイル I/O 性能に大きな影響を与えていないことも確認した．これは，ワーカノード上でファイルシステムイベントを取得し，データ生成直後にメタデータを即座に取得し，メモリ上に設置したメタデータデータベースに取得したメタデータを蓄積することで，ディスク I/O に与える影響を最小限に抑えている．すなわち，これらの方針が有効であることを意味している．さらに，NFS サーバがエクスポートした領域上で，MetaFa-MC を動作させた場合でも，ディスク I/O に与える影響は小さいことを示した．Gfarm については，マシンの性能が異なる環境に Gfarm を構築したため直接の比較はできないが，Gfarm ではメタデータ管理機能をファイルサーバノードとは別のサーバに設置する．そのため，メタデータアクセスがネットワーク経由となり，メタデータアクセスがボトルネックとなる可能性が高い．Gfarm と比較して，MetaFa ではメタデータを高速にアクセス可能なメモリ上に一時的に蓄積し，その後，一括してアップロード処理を行うため一定の I/O 性能を維持できる．Gfarm では，メタデータ生成・操作において，POSIX ファイルシステムの制約を受けるが，MetaFa システムではその制約を受けない．これは両者のメタデータの取り扱い方の違いによるものである．

次に，メタデータのスキーマレス管理による処理速度の向上について述べる．まず，メタデータのスキーマを管理する手法とスキーマレスでメタデータを管理する手法のメリット，デメリットを定性的に述べる．スキーマレスで管理するメリットは，ユーザがあらかじめ MetaFa システムへメタデータのスキーマを登録せずに利用できる点である．特に，ユーザはグリッドやクラウドコンピューティング環境ではどこのワーカノードで処理を実行するかを特定できない．そのため，ユーザが処理を行う前にメタデータのスキーマをワーカノードに配布しておくことは難しい．スキーマレスで管理することにより，MetaFa-MC や MetaFa-MMS が各アプリケーションの個別のメタデータスキーマを管理せずにメタデータを蓄積することが可能である．個別に管理することは，データベースがシャーディング機能などデータベースの動的分割に対応していれば，メタデータのスキーマを管理することができる．しかしながら，シャーディング機能などは，商用データベースなどに限られているため，すべての MetaFa で商用データベースを利用することは難しい．メタデータをスキーマレスで管理することにより，RDBMS でもメタデータを柔軟に管理できる．スキーマレスで管理することはメタデータをシリアライズしてデータベースへ格納するため，ユーザがメタデータを利用するためには，インデックスを作成しなければならない．しかし，このインデキシング処理も並列化するなどして，処理の高速化を図れば，特に大きな問題とはならないものと考えられる．

最後に，MetaFa を広域分散環境で運用した場合の考察を与える．実験結果より複製するメタデータ数が 10^5 個程度までであれば，RTT の大きな影響を受けずに，次の複製処理が実行されるまでに複製処理を完了できる．次に，メタデータ複製処理における信頼性について述べる．まず，メタデータ管理サーバ間でのメタデータ複製は，インターネットを経由することを前提としているため，複製メッセージが消失することや途中経路の輻輳により，遅延が発生することが考えられる．MetaFa-MMS はあらかじめメタデータ複製を始める前に，送信予定のメタデータ ID の範囲（最小値，最大値）を送信する．メッセージを受信した MetaFa-MMS は，受信したメタデータ ID の範囲を記録する．複製先 MetaFa-MMS は，メタデータ複製メッセージを受信すると，メタデータ ID が受信予定であったか，受信予定数どおりであったかを確認する．受信した複製メッセージの中で，欠落したメタデータ ID が存在した場合は，該当するメタデータ ID だけ再送要求を行う．プロトタイプでは，実験を行った際はメッセージの消失は発生せず，すべてのメタデータを複製することができた．しかしながら，インターネット上での運用を考慮した場合は，様々なトラブルが考えられるため，メタデータ複製時の失敗におけるリカバリ処理について今後検討する必要がある．

7. おわりに

本論文では，ユーザがグリッドやクラウドなどの広域分散処理環境でデータアクセスする際に，データの特性やセマンティクスを利用できるようにするための広域分散メタデータ管理システム MetaFa（Metadata administration Factory）を提案した．MetaFa では，

データを処理するワーカノード上でアプリケーションのI/O性能へ影響を低く抑えながら，リアルタイムにメタデータを取得する機能を提供する．さらに本論文では，MetaFaシステムの有用性を評価するために，プロトタイプを実装し，広域分散環境で動作するメタデータ管理の性能評価を実施した．これらの結果から，広域分散コンピューティング環境において，ユーザがメタデータを用いて柔軟にデータアクセスを行うためのメタデータ管理手段としてMetaFaシステムの有用性を示した．

最後に，今後の課題について述べる．現在，MetaFa-MCはファイルシステムイベントを受け取り，そのタイミングでメタデータを取得し，120秒ごとにメタデータをアップロードしている．アプリケーションには様々なデータアクセスパターンがあるため，データアクセスパターンを考慮し，120秒ごとにアップロードしていたメタデータを，ディスクI/Oが少ないタイミングやワーカノードの負荷が低いがタイミングでメタデータをアップロードする戦略アルゴリズムを考案する必要がある．

謝辞　本研究の一部は，情報爆発時代に向けた新しいIT基盤技術の研究，文部科学省研究費補助金「特定領域研究」（課題番号21013041）による成果である．ここに謝意を示す．

参 考 文 献

1) Baru, C., Frost, R., Marciano, R., Moore, R., Rajasekar, A. and Wan, M.: Metadata to support Information-Based Computing Environments, *Proc. 2nd IEEE International Conference on Metadata'97* (1997).
2) Baru, C., Moore, R., Rajasekar, A. and Wan, M.: The SDSC storage resource broker, *Proc. 1998 Conference of the Centre for Advanced Studies on Collaborative Research* (1998).
3) San Diego Supercomputer Center: Storage Resource Broker (online), available from ⟨http://www.sdsc.edu/srb/index.php⟩ (accessed 2010-8-30).
4) CIPA: Exif Print, available from ⟨http://www.cipa.jp/exifprint/index_j.html⟩ (accessed 2010-8-30).
5) Consortium, P.: PlanetLab (online), available from ⟨http://www.planet-lab.org/⟩ (accessed 2010-8-30).
6) corbet: Watching filesystem events with inotify (online), available from ⟨http://lwn.net/Articles/104343/⟩ (accessed 2010-8-30).
7) Foster, I. and Kesselman, C.: *The GRID2 Blueprint for a New Computing Infrastructure*, Morgan Kaufman (2003).
8) Foster, I., Zhao, Y., Raicu, I., et al.: Cloud Computing and Grid Computing 360-Degree Compared, *IEEE Grid Computing Environments Workshop 2008*, pp.1–10 (2008).
9) Gifford, D., Jouvelot, P., Sheldon, M. and O'Toole, J.: Semantic File Systems, *Proc. 13th ACM Symposium on Operating Systems Principles*, Association for Compugin Machinery SIGOPS, pp.16–25 (1991).
10) Gorton, I., Greenfield, P., Szalay, A., et al.: Data-Intensive Computing in the 21st Century, *Computer Magazine*, Vol.41, No.4, pp.30–32 (2008).
11) Hipp, D.R.: SQLite (online), available from ⟨http://www.sqlite.org/⟩ (accessed 2010-8-30).
12) Medical Imaging & Technology Alliance: DICOM Homepage (online), available from ⟨http://medical.nema.org/⟩ (accessed 2010-8-30).
13) NASA/GSFC: FITS Documents (online), available from ⟨http://fits.gsfc.nasa.gov/documents.html⟩ (accessed 2010-8-30).
14) Norcott, W.D.: Iozone Filesystem Benchmark (online), available from ⟨http://www.iozone.org/⟩ (accessed 2010-8-30).
15) Oracle: Lustre a Network Clustering FS (online), available from ⟨http://www.lustre.org/⟩ (accessed 2010-8-30).
16) Rajasekar, A., Wan, M., Moore, R. and Schroeder, W.: A Prototype Rule-based Distributed Data Management System, *HPDC workshop on Next Generation Distributed Data Management* (2006).
17) Tatebe, O., Morita, Y., Matsuoka, S., et al.: Grid Datafarm Architecture for Petascale Data Intensive Computing, *Proc. 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, pp.102–110 (2002).
18) Thanh, T.D., Mohan, S., Choi, E., Kim, S. and Kim, P.: A Taxonomy and Survey on Distributed File Systems, *International Conference on Networked Computing and Advanced Information Management*, pp.144–149 (2008).
19) Vaquero, L.M., Rodero-Merino, L., Caceres, J. and Lindner, M.: A Break in the Clouds: Towards a Cloud Definition, *ACM SIGCOMM Computer Communication Review*, Vol.39, No.1, pp.50–55 (2009).
20) Xing, J., Xiong, J., Ma, J. and Sun, N.: Memory Based Metadata Server for Cluster File Systems, *International Conference on Grid and Cooperative Computing*, pp.287–291 (2008).
21) 建部　修，森田洋平，松岡聡ほか：ペタバイトスケールデータインテンシブアプリケーションのためのGrid Datafarmアーキテクチャ，情報処理学会論文誌：ハイパフォーマンスコンピューティングシステム，Vol.43, No.SIG 6, pp.184–195 (2002).
22) 小西史一：巨大データの扱いと解析，情報処理，Vol.50, No.9, pp.845–852 (2009).

(平成22年 5 月31日受付)
(平成22年11月 5 日採録)

池部　　実（学生会員）

平成 16 年大分大学教育福祉科学部情報社会文化課程卒業．平成 18 年奈良先端科学技術大学院大学情報科学研究科博士前期課程修了．同年より同大学情報科学研究科博士後期課程在学．グリッドコンピューティング，クラウドコンピューティング等，広域分散処理システムの研究に従事．IEEE 学生会員．

猪俣　敦夫（正会員）

平成 14 年北陸先端科学技術大学院大学情報科学研究科博士後期課程修了，平成 16 年筑波大学先端学際領域センター客員研究員，同年独立行政法人科学技術振興機構社会技術研究開発センターを経て，平成 20 年奈良先端科学技術大学院大学情報科学研究科・特任准教授，現在に至る．博士（情報科学）．情報セキュリティの研究開発に従事．電子情報通信学会，映像情報メディア学会，教育システム情報学会各会員．

藤川　和利（正会員）

昭 63 年大阪大学基礎工学部情報工学科卒業．平 3 年同大学大学院基礎工学研究科博士後期課程退学後，同年大阪大学基礎工学部助手等を経て，平成 14 年奈良先端科学技術大学院大学情報科学センター助教授，平成 17 年同大学情報科学研究科助教授，現在に至る．博士（工学）．分散処理システム，マルチメディアシステムの研究開発に従事．電子情報通信学会，IEEE，ACM 各会員．

砂原　秀樹（正会員）

昭和 58 年慶應義塾大学工学部電気工学科卒業．昭和 63 年同大学大学院博士課程修了．同年電気通信大学情報学部助手．平成 6 年奈良先端科学技術大学院大学情報科学センター助教授．平成 13 同大学情報科学センター教授．平成 17 同大学情報科学研究科教授，現在に至る．工学博士．インターネット，大規模広域分散環境，ネットワーク，並列処理，オペレーティングシステム，電子図書館に関する研究に従事．電子情報通信学会，ACM，IEEE 各会員．