2.2.2 研究開発について

2.2.2.1　広域分散メタデータ管理システム

藤川　和利

1. はじめに

奈良先端科学技術大学院大学を広く学外にアピールするためには、大学内で発生する様々な有益な情報を効率よく管理し、学外に情報発信していくことが重要である。学内の教員・学生が生成する情報を個々の利用者が何らかの形で学内のある管理システムに登録することは非常に煩わしいものであると考えられる。

そこで本研究では、学内で発生する情報のメタデータを抽出し、メタデータを利用して情報にアクセス機構を広域分散ネットワーク環境で実現することを目指す。本研究におけるメタデータとは、生成された情報について、特徴的な内容を体系的に表すものである。例えば、一般的な画像データでは EXIF 情報、天文学関連の画像データでは FITS、医学分野の画像データは DICOM などがメタデータとなる。学内において生成された情報から、メタデータを抽出し、メタデータを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、利用者がメタデータを用いて、情報へアクセスする際のデータ検索用 API も提供する。利用者は MetaFa システムを利用することにより、メタデータ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。データ量の増加とともに、クラウド・グリッド環境ではデータが分散するため、目的とするデータの発見が難しくなる。これまで、階層型ファイルシステムのまま、ファイルのも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[1]が提案されている。Semantic File System 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。

2. MetaFa の設計方針

ここでは、メタデータ管理の要求要件を踏まえ、広域分散環境でメタデータを取り扱う際に、データの鮮度を維持したまま、個々のアプリケーションがもつ差異を吸収してメタデータを管理する手法など、提案する MetaFa システムの設計方針について述べる。

2.1 MetaFa の概要

MetaFa システムのアーキテクチャを図 1 に示す。MetaFa では、クライアント・サーバモデルにてメタデータ管理を行う。計算処理を実行するワーカーノード上に、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。ワーカー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[2],[3]。ワーカー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、ワーカー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。

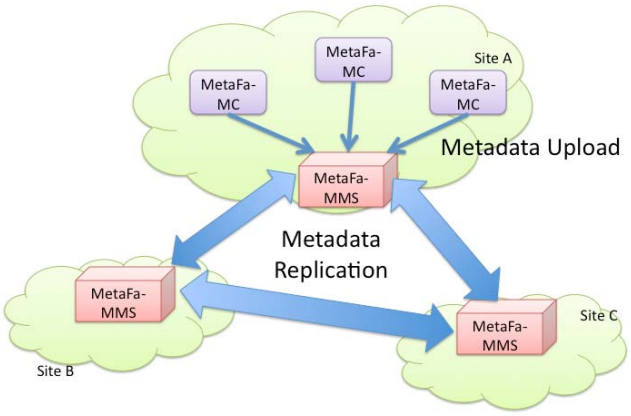

図 1: MetaFa システムアーキテクチャ

2.2 メタデータ

ここでは、MetaFa で取り扱うメタデータの定義を行う。MetaFa では、2 種類のメタデータを定義する。データ管理用の基本メタデータと、ユーザが定義するアプリケーションメタデータである。基本メタデータの要素を表 1 に示

す。インターネット上の分散ストレージに存在するデータを管理すること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。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は、ファイル自身がもつファイル属性の他に、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。

表 1: 基本メタデータの要素一覧

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	ワーカー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	ワーカーノード上でのユーザ ID
uid_name	ワーカーノード上でのユーザ名
st_gid	ワーカーノード上でのグループ ID
gid_name	ワーカー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	ワーカーノードのホスト名

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。一般的な文書ファイル、画像データのアプリケーションメタデータは既存ソフトウェアによって、自動的に収集可能である。また、アプリケーションが生成するデータのアプリケーションメタデータは、アプリケーションを利用するユーザがア

プリケーションメタデータのスキーマを決定し、それを利用するものとする。

2.3 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 ツールに代表されるデスクトップ検索システムである。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挙げられる。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 100%使い切る問題があった。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、ただちに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。ファイルの生成、更新が起きた瞬間に生成されたデータから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取得が可能となる。

2.4 メタデータの管理方針

ここでは、メタデータの管理方針について説明する。1 つめのメタデータ管理方針は、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ものである。メタデータのスキーマを管理する方法は、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毎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

ョン毎にテーブルが分離され、問い合わせのクエリをアプリケーション毎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は、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ーノードで複数のアプリケーションを実行するため、ワーカー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全てのデータベース、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、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を行う。

2 つめのメタデータ管理方針として、全てのメタデータ管理サーバで、全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全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も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高くなる。

(ア) MetaFa システムのプロトタイプ

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 2 に示す。

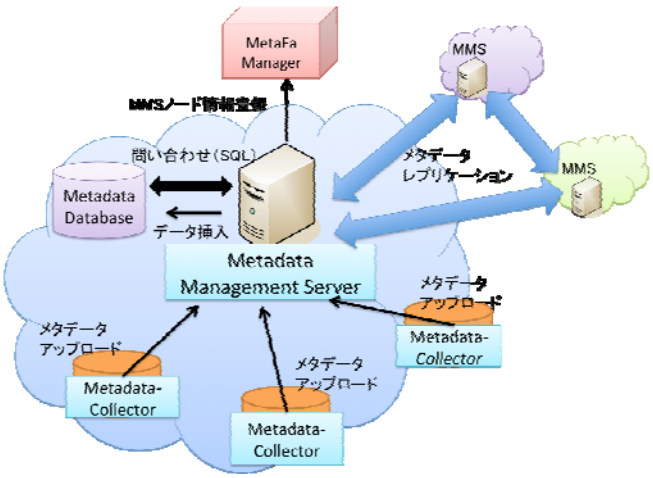

図 2: MetaFa のコンポーネント

3.1 MetaFa-MC

MetaFa-MC (Metadata Collector)は、ジョブを実行するワーカーノードで動作するメタデータを収集する。MetaFa-MC は、ワーカー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。収集したメタデータは、各ワーカーノードの MetaFa-MC が保存する。MetaFa-MC は、アプリケーションが実行結果のファイルを出力したとき、またはファイルが編集されたとき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 3 に示す。MetaFa-MC は、ファイルシステムイベントを inotify[4]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でのメタデータの取得を行う。inotify 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、監視対象であるディレクトリを watch リストに追加する。inotify の監視対象は、ディレクトリまたはファイルを指定することができる。MetaFa-MC では、ディレクトリのみを監視の対象とする。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ことが可能である。また、設定ファイルに監視ディレクトリを記述することも可能である。inotify から受け取る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指定できるので、MetaFa-MC では、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 が発生した際に、ファイルの書き込みが終了するのでこのタイミングでメタデータの抽出を行う。

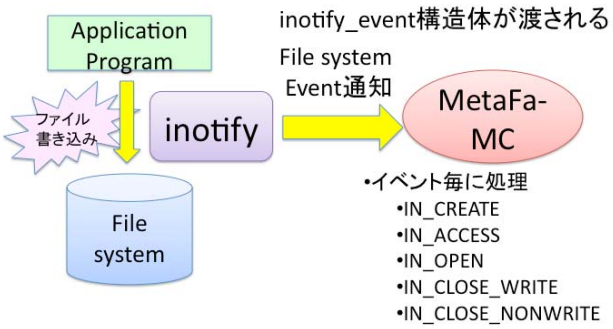

図 3: ワーカーノード上での MetaFa-MC の挙動

表 2: inotify イベント一覧

Event	Event の内容
IN_ACCESS	ファイルの読込が発生
IN_ATTRIB	メタデータ(i-nodeなどのファイル属性)が更新
IN_CLOSE_WRITE	書込のためにオープンされたファイルがクローズ
IN_CLOSE_NOWRITE	書込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

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、sqlite3[5]を採用した。sqlite3 は、メモリ上にデータベースを配置することができる。MetaFa では、メタデータデータベースをワーカーノードのメモリ上

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、高速にメタデータを処理できる。また、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 デーモンは、ファイルシステムイベントが発生した際にイベント毎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 ATTRIB、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を受信し、イベント毎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。MetaFa-MC では、アプリケーション固有のメタデータを抽出するために、はじめに、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。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、アプリケーションメタデータとしてメタデータデータベースへ格納する。すなわち、一般に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 側で登録しておき、生成されたデータの拡張子にてどの抽出プログラムを呼び出すか判断する。MetaFa プロトタイプ実装では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。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ー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。

3.2 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 へ接続をする。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、既に登録されている 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、既に接続してきた 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 へ問い合わせを行う。このとき、MetaFa-Manager は、代替の MetaFa-MMS を MetaFa-MC へ通知する。

3.3 MetaFa-MMS

MetaFa-MMS (Metadata Management Server)は、MetaFa-MC で収集したメタデータを管理するサーバである。MetaFa-MMS のコンポーネントを図 4 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、MetaFa-MMS は、蓄積したメタデータを他の MetaFa-MMS へ複製する Metadata Replicator、データベースに対して、メタデータを Python 辞書型データで格納しているため、メタデータ検索用にメタデータのインデックスを作成する Metadata Indexer を実装した。

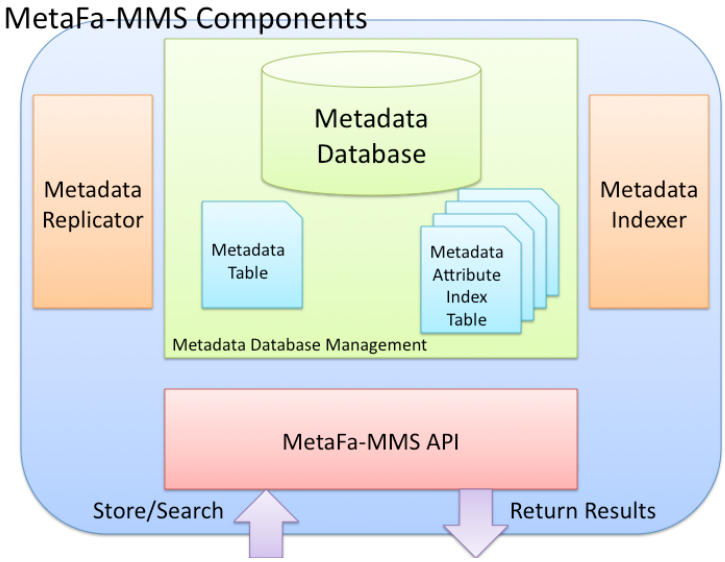

図 4: MetaFa-MMS コンポーネント

4. MetaFa の評価

アプリケーションメタデータ取得におけるオーバーヘッドを計測し、提案システムの有効性を検証する。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 はメタデータの要素として、撮影日時、画像の解像度などの情報を持つ。このとき、MetaFa-MC では、作成したファイルを拡張子(JPEG、もしくは、JPG)にて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、80KB 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 でデータ生成を行った際には、MetaFa-MC にてメタデータの取得は行っていない。本実験の結果を図 5 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。NFS や Gfarm 上でデータの生成を行った場合、生成したファイルサイズが 80KB であり、小さなファイルを大量に生成するため、I/O 操作の都度、通信が発生するため、大きなオーバーヘッドとなっていることと考えられる。NFS では、NFS サーバ側への書き込みが頻繁に発生することになる。また、Gfarm では、メタデータサーバとローカルディスク、ファイルサーバノードに

対して大量の通信と書き込みが発生する。一方、MetaFa では、ローカルディスクに対して I/O 操作を行い、I/O 操作の後に、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。そのため、I/O 操作時における通信は発生しない。これは、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である。NFS や Gfarm では、遠隔にあるサーバに対してファイル I/O 処理を行う。MetaFa では、ローカルディスクに対してファイル I/O 処理を行うため、通信のオーバーヘッドはほとんど存在しない。すなわち、NFS や Gfarm と MetaFa では、オーバー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が示された。

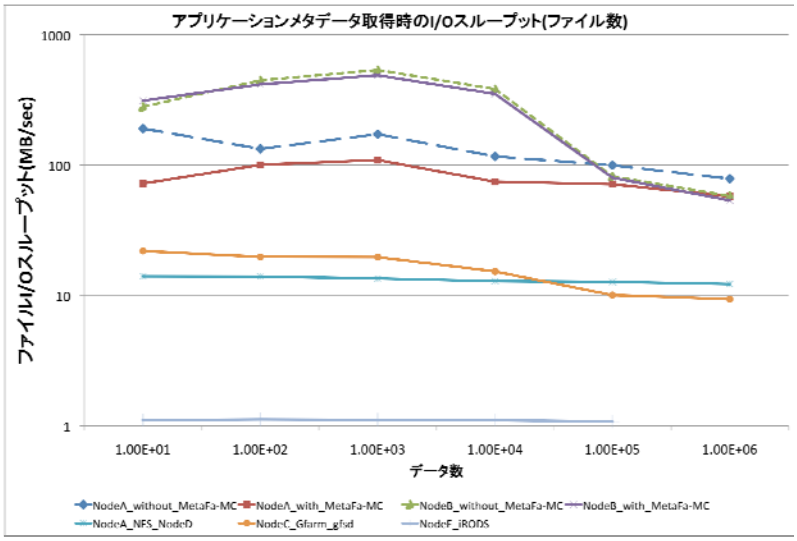

図 5: アプリケーションメタデータ取得時のスループットの比較(ファイル数)

5. おわりに

本研究では、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、メタデータ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ー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。メタデータはワーカーノードに一時的に蓄積することに限定するため、MetaFa システムではメタデータを管理する専用ノードを用意する。メタデー

タ管理サーバでは、ワーカーノードからのメタデータのアップロードを受け付ける。それだけでなく、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様々なアプリケーションのメタデータを管理することができる。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。

参考文献

[1] Gifford, D., Jouvelot, P., Sheldon, M. and O'Toole, J.: Semantic File Systems, *Proc. of 13th ACM Symposium on Operating Systems Principles*, pp.16–25 (1991).

[2] Thanh, T.D., Mohan, S., Choi, E., Kim, S. and Kim, P.: A Taxonomy and Survey on Distributed File Systems, *Proc. of International Conference on Networked Computing and Advanced Information Management*, pp. 144–149 (2008).

[3] Xing, J., Xiong, J., Ma, J. and Sun, N.: Memory Based Metadata Server for Cluster File Systems, *Proc. of International Conference on Grid and Cooperative Computing*, pp.287–291 (2008).

[4] Watching filesystem events with inotify (online), available from <http://lwn.net/Articles/104343/>. (accessed 2010-8-30).

[5] SQLite (online), available from <http://www.sqlite.org/>. (accessed 2010-8-30).