NAIST-IS-DD0661001

博士論文

パラメータサーベイアプリケーションのための メタデータを用いた 広域分散型データアクセス機構に関する研究

池部 実

2011年3月17日

奈良先端科学技術大学院大学
情報科学研究科 情報システム学専攻

本論文は奈良先端科学技術大学院大学情報科学研究科に
博士 (工学) 授与の要件として提出した博士論文である。

池部 実

審査委員：

砂原 秀樹 教授	（主指導教員）
山口 英 教授	（副指導教員）
藤川 和利 准教授	（副指導教員）
伊達 進 准教授	（大阪大学）

パラメータサーベイアプリケーションのための メタデータを用いた 広域分散型データアクセス機構に関する研究*

池部 実

内容梗概

ユーザは，科学技術計算シミュレーションなどの大規模計算処理を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などの広域分散環境上で実行することで科学技術の発展に寄与してきた．グリッド環境で実行した科学技術計算から出力されるデータは，インターネット上に分散するストレージへ保存される．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されてきた．しかし，広域分散環境において，他のユーザとデータを共有する際，従来のファイル名によるデータアクセス手法では，データ参照コストが大きい．データ参照コストとは，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手間を意味する．データを生成したユーザ以外のユーザが，データアクセスする際に，ユーザが知っているデータの特性やデータが持つ意味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することが可能となる．本研究では，広域分散環境上で実行するパラメータサーベイアプリケーションを対象として，データの特性や意味をメタデータで表現し，メタデータを収集・管理するための広域分散メタデータ管理システム MetaFa (Metadata administration Factory) を提案する．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ことがで

*奈良先端科学技術大学院大学 情報科学研究科 情報システム学専攻 博士論文, NAIST-IS-DD0661001, 2011 年 3 月 17 日.

きる．また，MetaFa 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可能である．さらに，本研究ではユーザがメタデータを用いてデータを検索し，検索結果からデータアクセスするための仮想ファイルシステム PVN-FS (Personal View Namespace File System) を提案する．PVN-FS は，MetaFa が収集・管理しているメタデータを用いてデータを検索する．ユーザは得られた検索結果を仮想ディレクトリ上に組み込み，アプリケーションを修正することなく，データへアクセスすることが可能である．MetaFa および PVN-FS のプロトタイプシステムを実装し，性能評価を実施した．実験結果から，広域分散環境において，パラメータサーベイアプリケーションから発生する小さなデータからメタデータを収集・管理する手法として MetaFa システムの有効性を示した．また，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手法として PVN-FS の有用性を示した．本研究では，MetaFa により広域分散環境において様々なメタデータを統一的に管理し，PVN-FS ではメタデータによる柔軟なデータアクセスを実現し，既存アプリケーションを修正することなく再利用を可能とし，メタデータを用いた柔軟なデータアクセスを実現した．本研究では，階層型ファイルシステムなどにとらわれないメタデータを用いたデータアクセスシステムの実現に貢献すると考える．本研究では，広域分散環境におけるメタデータを用いたデータアクセスのための基盤技術を確立し，ユーザ間でデータを共有する際のデータ参照コストの削減を実現した．

キーワード

メタデータ, 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, ファイルシステム, データグリッド, パラメータサーベイ, ワークフロー

A study on metadata based data access system for parameter sweep applications on the wide area distributed environments*

Minoru Ikebe

Abstract

Users contributed greatly to the development of science technology to run large scale scientific simulation on the wide area distributed computing environments as the grid computing or the cloud computing. An enormous amount of data generated by scientific applications on the grid or the cloud computing were output to storages distributed on the Internet. Still to find out a target data is difficult for users applying the data access method by traditional hierarchical namespace to the grid and cloud computing environments. If a user shares data with other users, the cost of data access for other users is high. The cost of data access means to find the target data from a huge amount of data. Metadata is 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 thesis, the author propose a wide-area distributed metadata management system, MetaFa, (Metadata administration Factory) which enable users to access data and collect the metadata by such characteristics and semantics especially automatically and efficiently for parameter sweep applications. MetaFa system can acquire the metadata in real time while keeping an I/O performance of the

*Doctoral Dissertation, Department of Information Systems, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DD0661001, March 17, 2011.

application on a worker node. Also, the MetaFa can manage a variety of application metadata without the application metadata schemas.

Furthermore, I propose PVN-FS (Personal View Namespace File System) which enable to flexible data access based on characteristics and semantics of data. The PVN-FS is enabling to mount original namespace on user's file systems. I implemented a prototype of my proposal system to evaluate an effectively of MetaFa/PVN-FS. Finally, I discussed about a possibility of MetaFa/PVN-FS being applied to the flexible data access method using metadata on the wide-area distributed computing environments.

I realized two purposes in my research. First, MetaFa enable to collect/manage the metadata in wide-area distributed environments. Second, PVN-FS enable to flexible data access using semantics of data. My research contributes to realization of I/O architecture based the metadata for parameter sweep applications in the grid environment.

Keywords:

Metadata, Grid computing, Cloud computing, File system, Data gird, Parameter sweep, Workflow

目 次

第 1 章　序論 **1**

1.1. 研究背景 1

1.2. 問題意識 3

1.3. 研究の目標と意義 6

1.4. 本論文の構成 8

第 2 章　グリッド環境におけるデータ管理とメタデータ **10**

2.1. データ管理システム 10

2.2. メタデータ 12

2.2.1　メタデータ 12

2.2.2　メタデータの用途 12

2.2.3　Dublin Core 13

2.2.4　特定アプリケーション向けデータフォーマット 13

2.3. データ管理におけるメタデータの利用方法 17

2.3.1　ファイルカタログ機能 17

2.3.2　ファイル検索機能 17

2.4. 科学技術計算アプリケーションとメタデータ 18

2.4.1　科学技術計算アプリケーションとメタデータ 18

2.4.2　科学技術計算アプリケーションにおけるデータ特性 19

2.4.3　科学技術計算アプリケーションとデータ鮮度 21

2.4.4　パラメータサーベイアプリケーション 22

2.4.5　パラメータサーベイのアプリケーション性能分析 23

2.5. メタデータの収集・管理，メタデータを用いたデータアクセスにおける要求事項 25

2.5.1 データ管理システムへの要求事項 25
2.5.2 データグリッドシステムにおける要求事項 26
2.5.3 メタデータを用いたデータアクセスにおける要求事項 . . . 26
2.5.4 広域分散環境でのメタデータ収集における要求事項 28
2.5.5 広域分散環境でのメタデータ管理における要求事項 29
2.6. 要求事項の分析・まとめ 30

第 3 章 関連研究 33
3.1. 関連研究の分類 33
3.2. メタデータを用いたデータアクセスシステム 34
3.2.1 POSIX ファイルシステム API 互換システム 34
3.2.2 非 POSIX ファイルシステム API 互換システム 38
3.2.3 メタデータを用いたデータアクセスのまとめ 43
3.3. メタデータ収集機能 46
3.4. メタデータ管理機能 48
3.4.1 メタデータ管理 48
3.4.2 メタデータ I/O 高速化のためのメタデータ管理 50
3.4.3 メタデータ複製における一貫性制御 52
3.4.4 データの鮮度を考慮したメタデータ管理 53
3.4.5 メタデータ管理における問題点のまとめ 53
3.5. 関連研究の問題点のまとめ 54

第 4 章 MetaFa: 広域分散環境におけるメタデータ収集・管理システム 56
4.1. MetaFa の概要 56
4.2. MetaFa におけるメタデータの定義 57
4.3. MetaFa の設計方針 60
4.3.1 メタデータ収集の自動化手法の方針 61
4.3.2 広域分散環境下におけるメタデータの管理方針 62
4.4. MetaFa システムのプロトタイプ 67
4.4.1 MetaFa システムの概要 67

4.4.2 MetaFa-MC 68
4.4.3 MetaFa-Manager 71
4.4.4 MetaFa-MMS 72
4.4.5 MetaFa-AA 75
4.5. MetaFa の実験・評価 76
4.5.1 実験 (1) – ファイル I/O スループットへの影響 77
4.5.2 実験 (2) – メタデータ収集に関する実験 87
4.5.3 実験 (3) – スキーマレスメタデータ管理手法 89
4.5.4 実験 (4) – メタデータインデックス作成に関する実験 . . . 93
4.5.5 実験 (5) – 広域分散環境でのメタデータ複製におけるスケーラビリティ 94
4.6. MetaFa の考察 102
4.7. 広域分散環境におけるメタデータ収集・管理システムのまとめ . . 103

第 5 章 広域分散環境におけるメタデータを用いたデータアクセスのための仮想ファイルシステム PVN-FS 105

5.1. メタデータを用いた検索可能なデータアクセスシステムの機能要件 106
5.2. メタデータを用いたデータ検索可能なデータアクセスシステムの設計 107
5.3. PVN-FS の概要 108
5.4. PVN-FS 上でのメタデータによるデータ検索 110
5.5. PVN-FS のデータアクセスと名前空間 112
5.6. 実験 – PVN-FS 上でのデータ処理速度の計測 115
5.7. PVN-FS の考察 122
5.7.1 PVN-FS の I/O 性能 122
5.7.2 PVN-FS でのデータにおける一貫性制御 123
5.7.3 PVN-FS でのディレクトリにおける一貫性制御 123
5.7.4 PVN-FS の利便性 124
5.7.5 アクセス透過性 124
5.7.6 アプリケーションの再利用性 124

5.8. 今後の課題 125
5.9. まとめ 125

第 6 章　結論 **126**
6.1. 本研究の考察 126
6.2. 本研究で実現したこと 127
6.2.1　メタデータを用いた検索可能なデータアクセスシステム . 128
6.2.2　広域分散環境でのメタデータ収集 128
6.2.3　広域分散環境でのメタデータ管理 129
6.3. 本研究によって得られた知見 129
6.4. 本研究の貢献 130
6.5. 今後の展望 131
6.6. 本研究のまとめ 133

謝辞 **135**

参考文献 **137**

研究業績 **143**

図 目 次

1.1 メタデータによるデータアクセス 7
1.2 メタデータによるデータアクセスを実現するために必要な機能 . . 8

2.1 計算処理アーキテクチャの変遷 11
2.2 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 20
2.3 各組織におけるデータグリッドの構成方法 27

3.1 Gfarm アーキテクチャ 35
3.2 Gfarm 上でのデータアクセス 36
3.3 HDFS アーキテクチャ 39
3.4 SRB のデータ検索インタフェース 43
3.5 SRB におけるデータアクセス 44
3.6 astro3d におけるメタデータ 49
3.7 サブツリーパーティショニング方式 51

4.1 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 57
4.2 MetaFa システムアーキテクチャ 60
4.3 MetaFa コンポーネント 67
4.4 ワーカーノード上での MetaFa-MC の挙動 70
4.5 MetaFa-MMS コンポーネント 72
4.6 メタデータの構造 74
4.7 MetaFa-MMS のメタデータ複製機能 75
4.8 MetaFa の有無によるファイル書き込み性能の比較 (レコード長:64KB) 80
4.9 MetaFa の有無によるファイル書き込み性能の比較 (レコード長:1024KB) 81

4.10 ファイルシステムベンチマーク実験環境 (1) 82
4.11 ファイルシステムベンチマーク実験環境 (2) 82
4.12 MetaFa と NFS, Gfarm によるファイル書き込み性能の比較 (レコード長:64KB) 84
4.13 MetaFa と NFS, Gfarm によるファイル書き込み性能の比較 (レコード長:1024KB) 85
4.14 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) 86
4.15 EXIF 情報取得時のメタデータ収集時間 (ファイル数) 88
4.16 メタデータスキーマの有無に応じたテーブル構成 92
4.17 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 93
4.18 メタデータインデックス作成処理時間 95
4.19 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) . 96
4.20 メタデータ複製時のスケーラビリティ (PlanetLab ネットワーク) . 99
4.21 Dummynet を用いた実験構成 100
4.22 メタデータ複製時のスケーラビリティ (Dummynet) 101

5.1 PVN-FS の構築，データアクセスフロー 109
5.2 メタデータの指定方法の例 111
5.3 PVN-FS のディレクトリとファイル属性データベースのテーブルのマッピング 112
5.4 pvn_mkdir.py によるディレクトリ作成 116
5.5 コマンドの実行時間の比較 118
5.6 RTT を増加させた場合の実行時間の比較 (cat) 119
5.7 RTT を増加させた場合の実行時間の比較 (grep) 120
5.8 RTT を増加させた場合の実行時間の比較 (sed) 120
5.9 RTT を増加させた場合の実行時間の比較 (awk) 121
5.10 RTT を増加させた場合の実行時間の比較 (ls) 121

表 目 次

1.1 WWW とデータグリッドにおけるデータアクセス手法の比較 . . . 5

2.1 Dublin Core の基本要素 16
2.2 アプリケーションが生成するデータ量 20

3.1 メタデータによるファイルカタログ機能 (POSIX 互換) 36
3.2 メタデータによるファイル検索機能 (POSIX 互換) 38
3.3 メタデータによるファイルカタログ機能 (非 POSIX 互換) 40
3.4 SRB のメタデータ 42
3.5 メタデータによるファイル検索機能 (非 POSIX 互換) 44
3.6 メタデータを用いたデータアクセスシステムの分類 45
3.7 Rcommands におけるメタデータ 46
3.8 メタデータ収集の関連研究のまとめ 47

4.1 基本メタデータの要素一覧 59
4.2 3 つのメタデータ収集方法の比較 62
4.3 inotify イベント一覧 69
4.4 メタデータインデックス 74
4.5 ワーカーノードのスペック 78
4.6 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 83
4.7 MetaFa-MC 上におけるスキーマありなしでのメタデータ読込処理速度比較 91
4.8 ワーカーノードとメタデータ管理サーバのマシンスペック 91
4.9 PlanetLab ノードのスペック要件 97
4.10 PlanetLab ノードへの平均 RTT 98

4.11 PlanetLab ノードのロードアベレージ 99

5.1 PVN-FS の提供するインタフェース 110
5.2 PVN-FS 実験に用いたノードのスペック 115
5.3 実験に用いたファイル 117

第1章　序論

本章では，研究背景として広域分散環境下での科学技術計算を実行するための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について述べる．グリッド環境上で科学技術計算を実行する上でのデータ管理の問題点について述べる．また，本研究の目標として広域分散環境上でのデータ管理において，メタデータを用いたファイルI/Oオペレーションを実現するために必要な機能について述べる．

1.1. 研究背景

1946年にENIAC (Electoric Numerical Integrator and Computer) が開発されて以来，様々な計算機が開発され，現在も計算機の進歩はとどまるところを知らない．計算機が誕生した直後は，計算機が高価であったため，大型計算機を複数のユーザで共有し，計算機資源を少しずつ利用していた．時代が進むにつれ，計算機の性能は向上し，安価となり個人で複数台の計算機を占有して利用可能となってきた．2010年11月 SC10[29] にて発表されたTop500[24] では，世界最速の計算機は，中国の天河一号Aが，2.507PFlopsであった．

科学技術計算では，ベクトル型の大型スーパーコンピュータから，大量のスカラ型のコモディティPCをInfinibandなどの広帯域低遅延なネットワークで接続して構成されたクラスタコンピュータなどを，分野やアプリケーションの目的によって様々な計算機を利用している．

また，1969年にARPANETという全米規模の計算機ネットワークの構築が始まり，1982年にTCP/IPが制定され，TCP/IPによるインターネットが誕生して30年近くが経過した．さらに，ここ数年でインターネットのバックボーンネット

ワークが，10 Gigabit Ethernet から 100 Gigabit Ethernet へと広帯域化が進んできている．

このような技術的な背景によって，2000 年代序盤頃から大学や研究機関が保有するクラスタコンピュータなどの計算機資源を広帯域ネットワークで接続し，超高速計算を行うためのグリッドコンピューティング [43] や，2000 年代中頃から，データセンタに設置された計算機上に仮想化技術および，分散処理技術を用いた多目的に利用可能なクラウドコンピューティング [44][51] と呼ばれる新たなコンピューティングパラダイムが登場してきた．

グリッドコンピューティングのグリッドとは，国中に網目上に張り巡らせ，電力を伝えるための送電線網 (electrical power grid) に由来している．ユーザは，テレビや電子レンジなどの電源プラグをコンセントに差し込み，電源を入れるだけで，電化製品を利用可能である．このとき，ユーザは，コンセントの向こうで，電気がどのように作られ，どのように電気が運ばれてきているかなど何が起きているのかを意識する必要はない．グリッドコンピューティングでは，電気のように，ユーザに対してサービスする側の資源 (計算機，ストレージ) を意識せずに利用可能にする．クラウドコンピューティングは，2006 年 8 月に開催された Search Engine Strategies Conference[19] の中で，Google 社の Eric Schmidt が，従来，主流であったクライアント・サーバモデルのコンピューティング環境に代わり，ユーザの手元に存在したデータサービスとアーキテクチャが，雲 (クラウド) の中のどこかにあると言及したことが始まりとされている．雲 (クラウド) は，インターネット網を雲の図で表すことに由来する．また，クラウドコンピューティングは，従来の自社運用 (オンプレミス) での環境から，インターネット上に配置されたサーバ資源を利用する環境への移行を表している．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の考え方は，Sun Microsystems の”The Network is the Computer” という標語が実現したものである．ユーザは，このようなグリッド環境やクラウド環境では，計算機資源やデータの位置を意識せずに処理できる．本研究での広域分散環境とは，グリッドコンピューティングのように，インターネットを経由し大学や研究機関等の組織を接続した環境，および，クラウドコンピューティングのように，データセンタに設置された計算機資源を利用する環境，

データセンタとデータセンタを接続した環境を意味する．

1.2. 問題意識

科学技術分野では，急速な計算機科学・計算機技術の進歩により，シミュレーションやデータ解析など計算機を利用した研究が大きく拡がりつつある．特に，クラスタコンピューティングやグリッドコンピューティングなどの計算機システムによって多くのパラメータを持つシステムや事象を計算によって理解，予測することが可能になりつつあり，科学の基礎から応用，そして社会での実践と深く広くインパクトが生じている [68].

John Taylor 氏によって提言された e-Science[3] とは，高速ネットワーク上で科学データの相互利用，統合利用，人的交流を目指した基盤技術に関する学術領域である．e-Science を実行するための大規模計算システムでは，大規模データの取り扱いが大きな問題となっている．e-Science の計算シミュレーションから生成されるデータは最終的な結果データだけではなく，計算過程における中間データなども含まれる．中間データを保持する理由は，計算途中でエラーが発生した場合や，途中でパラメータを変更した場合に，中間データを用いて途中から計算を再実行するためである．また，センサグリッドやリモートセンシングなどの技術によって，CERN の LHC(Large Hadron Collider) や天文台の電子望遠鏡などの実験装置から出力される大量の観測データを容易に得ることが可能となった．グリッドコンピューティングなどの広域分散環境では，大量の計算資源を用いて，数 KB から数 PB のデータを処理している．

2005 年の段階で，CERN の大型ハドロン衝突加速器 LHC は，1 秒間に 1GB のデータを出力し，1 年間で出力するデータ量は 10PB となる．このほかにも，ピクサーアニメーションスタジオでは，1 ムービーあたり 100TB ものデータを生成する．これらの大量のデータを保存する方法は，ファイルシステムに限らず，関係データベース (RDB) や，XML ファイル，XML データベースなど様々な方法がある．

グリッドコンピューティングでは，実験装置やシミュレーションからで生成され

るデータをどのように蓄積し，ユーザが必要なデータに対して，どのようにアクセスするかが大きな問題である．e-Science に限らず，WWW (World Wide Web) などにおいても大量の情報が創出される情報爆発時代において，データ管理に必要な機能は，以下の 3 つの機能が必要である．本研究では，3 つの機能のうち，広域分散環境におけるデータ発見，データアクセスについて考える．

- Data Discovery (データ発見)
- Data Access (データアクセス)
- Data Integration (データ統合)

WWW 技術においては，Google や Yahoo!などの大規模検索サービス，セマンテックウェブ技術などを用いてユーザは，インターネット上に存在する情報の中から必要な情報を検索し，目的の情報に対してアクセスすることが可能である．グリッドコンピューティングでは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．データグリッドシステムでは，広域分散環境に存在するストレージやファイルシステムを統合して，広域分散ファイルシステムを提供する．データグリッドシステムを用いることでグリッド環境上で実行したアプリケーションが出力するデータを管理できる．WWW 技術とデータグリッド技術をデータアクセスプロトコルや対象とするデータなど比較したものを表 1.1 に示す．WWW 技術では，ユーザはサービス提供者が提供する HTML ファイルに対してブラウザでアクセスする．Web におけるデータは，リンク構造によって他のデータと関連づけられている．データグリッドシステムでは，アクセス対象がデータベース，ファイルなど種類が異なり，アクセスプロトコルも FTP，SSH，GridFTP など多岐に渡る．さらに，データグリッドではデータ同士の関連性を記述する手段がない．

多くのデータグリッドシステムでは，Lustre[9] 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，データの場所などの情報を管理するノードなどデータに関する情報をデータ自体と分離してデータを管理する．これらのデータグリッドシステムでは，ユーザが利用する UNIX 環境での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．すな

表 1.1 WWW とデータグリッドにおけるデータアクセス手法の比較

	WWW	データグリッド
データアクセス プロトコル	HTTP	FTP，SSH，HTTP， GridFTP
アクセス対象データ	HTML ファイル	ファイル，データベース
データ管理	サービス提供者が 管理，公開	ユーザ，グループ による管理など
データとデータの関連	リンク構造	記述できない

わち，POSIX ファイルシステム API と互換性をもつデータアクセス手法を提供し，データ参照のための識別子を広域分散環境向けに拡張している．

階層型ファイルシステムでは，ディレクトリにて木構造の名前空間を表現し，ファイルを各ディレクトリに保存する．しかしながら，階層型ファイルシステムはユーザの利便性やシステムの制限により，限界に近づいている [46][53]．例えば，複数ユーザで，階層型ファイルシステムを利用する場合，名前の衝突が発生するケースや，アプリケーションによっては，i-node が不足する場合などの問題も発生する．ユーザは，大量のデータを保存したファイルシステムに対して，アクセスしたいデータをどこに保存したか覚えていないことがたびたび発生する．階層型ファイルシステムでは，アクセスのための識別子が，ファイルの名前しか存在しないため，ユーザに対して，目的のデータを発見し，アクセスするため負担が大きくのしかかっている．

このような問題に対して，ユーザは，オペレーティングシステムや，サードパーティーが提供するデスクトップ検索ツールを用いてデータを発見し，アクセスする手間を削減している．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ユーザが知る情報を用いて検索可能である．

科学者は，グリッド環境で科学技術シミュレーションを実行する．シミュレーションモデルは，複数の研究者で構築する場合や，別々の研究者が異なるシミュレーションモデルを構築し，シミュレーション結果を比較する場合がある．この

ため，研究者同士でシミュレーションモデルやデータを共有する．研究者は，参照しようとしているデータがどのようなものか，誰が生成したデータなのか，どのプログラムでアクセスすればよいのかなどデータを適切に認識するために手間がかかっている．本研究では，ユーザがデータを参照するための手間をデータ参照コストと呼ぶ.

ユーザは，広域分散環境においても，デスクトップ検索のようにユーザの持つ知識を用いて手軽にデータ検索でき，データ参照コストを抑えることが可能なシステムを求めている．しかしながら，多くのデータグリッドシステムでは，ユーザが利用可能なデータに関する情報を保持していない．広域分散環境において，ユーザがデータに関する情報を用いてアクセスするためには，データを保存しているファイルシステム上で，デスクトップ検索ツールを用いて検索用のインデックスを構築する必要がある．さらに，広域分散環境では，分散するデスクトップ検索ツールのインデックス情報を統合して管理する必要がある.

1.3. 研究の目標と意義

グリッド環境やクラウド環境などの広域分散環境下において必要とされるデータアクセスとは，WWW と同様に存在するデータの中からユーザの必要とするデータに関する情報 (メタデータ) を用いたデータアクセスである (図 1.1)．例えば，広域分散環境上の分散するデータに対して，ユーザは実行したジョブの結果データへアクセスしたい場合，結果データがどこにあるかひとつひとつ調べなければならない．ユーザは，ジョブの日時やプログラム名などでアクセスすることで，データ参照コストを削減可能となる.

本研究の目標は，ユーザの必要とするデータに関するメタデータを用いたデータアクセス機構を広域分散環境で実現することである．本研究で取り扱うメタデータは，データの管理に必要な情報を保持する基本メタデータとアプリケーションが持つ特殊な情報をアプリケーションメタデータと呼ぶ．アプリケーションメタデータとは，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．例えば，画像データについてのアプリケーションメタデー

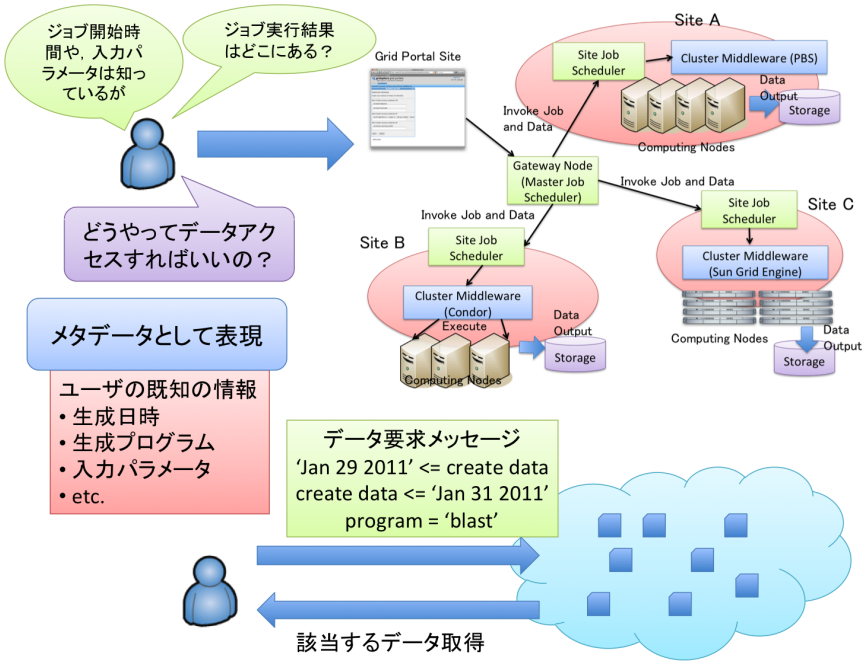

図 1.1 メタデータによるデータアクセス

タとは，一般的な画像データでは，EXIF[38] 情報，天文学関連のアプリケーションで扱う画像データでは FITS[5]，医学分野の画像データは DICOM[54] などである．本研究では，アプリケーションメタデータを用いたデータアクセスを，“メタデータ・ファイル I/O オペレーション (File I/O operation using metadata)” と呼ぶ．メタデータ・ファイル I/O オペレーションを実現することで，ユーザは，従来の階層型ファイルシステムにとらわれることなく，ユーザが有するデータに関する知識を用いたデータ管理・データアクセスが可能となる．

研究目標を実現するためには，図 1.2 に示す 2 つの基盤が必要となる．

- 広域に分散するデータからメタデータを収集し，ユーザへ提供するための基盤
- メタデータを用いたデータ検索可能なデータアクセス基盤

本研究では，グリッド環境，クラウド環境などの広域分散環境において，実行されるパラメータサーベイアプリケーションから生成されるデータからメタデー

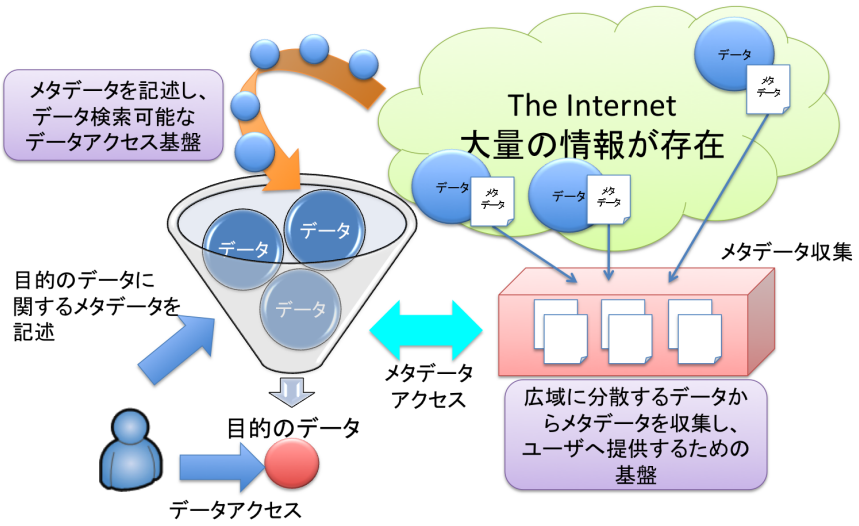

図 1.2 メタデータによるデータアクセスを実現するために必要な機能

タを抽出し，メタデータを広域分散ネットワーク環境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 administration Factory)* を提案する．また，MetaFa システムが管理するメタデータを用いてデータ検索を行い，検索結果を既存ファイルシステムに組み込み可能なデータアクセスのための仮想ファイルシステム *PVN-FS (Personal View Namespace File System)* を提案する．

1.4. 本論文の構成

本論文の構成を示す．2 章では，一般的なデータ管理とメタデータの関係について述べる．そして，データ管理システムが抱える問題点について述べる．データ管理システムが抱える問題点を解決するためにデータ管理システムやユーザからの要求要件をまとめる．3 章では，既存のデータ管理システムの問題点を解決するための 2 章でまとめた要求要件に照らし合わせ，既存システムを比較する．最後に，既存システムで解決できていない課題について述べる．4 章は，広域分散環境におけるメタデータ収集・管理機構 MetaFa を提案する．要求要件に照らし合わせ，広域分散環境でのメタデータ管理手法を比較する．また，I/O スループットに影響を与えないメタデータ収集方法について述べる．5 章では，メタデー

タを用いたデータ検索可能な仮想ファイルシステム PVN-FS を提案する．6 章では，提案したシステムについて考察し，本研究で得られた知見や，本研究が果たす社会に対する貢献を述べ，今後の展望について述べ，本論文をまとめる．

第2章　グリッド環境におけるデータ管理とメタデータ

本章では，グリッド環境におけるデータ管理システムについて述べ，既存のデータ管理システムが抱える問題点についてまとめる．また，データ管理システムにおけるメタデータの利用方法，メタデータの管理方法についてまとめ，問題点を議論する．本論文で対象とするパラメータサーベイアプリケーションについて述べる．最後に，メタデータを用いたデータ管理，データアクセスシステムに求められる要求事項を整理する．

2.1. データ管理システム

本節では，計算機アーキテクチャとデータ管理システムの変遷について述べる．また，本研究で対象とするデータ管理システムについて述べる．

計算機の構成として，1CPU に対して 1 ディスクが主流であったが，1980 年頃から SMP(Symmetric Multiprocessing) のような物理メモリを共有した対称共有メモリアーキテクチャが登場した．さらに HDD を RAID 構成にして接続し，計算機アーキテクチャのスケールアップが図られた．スケールアップとは，垂直スケールとも呼ばれ，計算機のプロセッサそのものを高性能化するなど，計算機自体の性能向上を意味する．さらに，1990 年代頃から，スケールアウトするために，複数の CPU ノードと SAN(Storage Area Network) を組み合わせた SAN ベースアーキテクチャ，複数のコモディティ計算機を並べた Brick ベースアーキテクチャなど，並列処理可能なクラスタベースのアーキテクチャ(クラスタコンピュー手 l んぐ) が登場した．スケールアウトとは，水平スケールとも呼ばれ，計算機の台数を増やして分散処理，並列処理して計算能力の向上を図る．さらに，2000

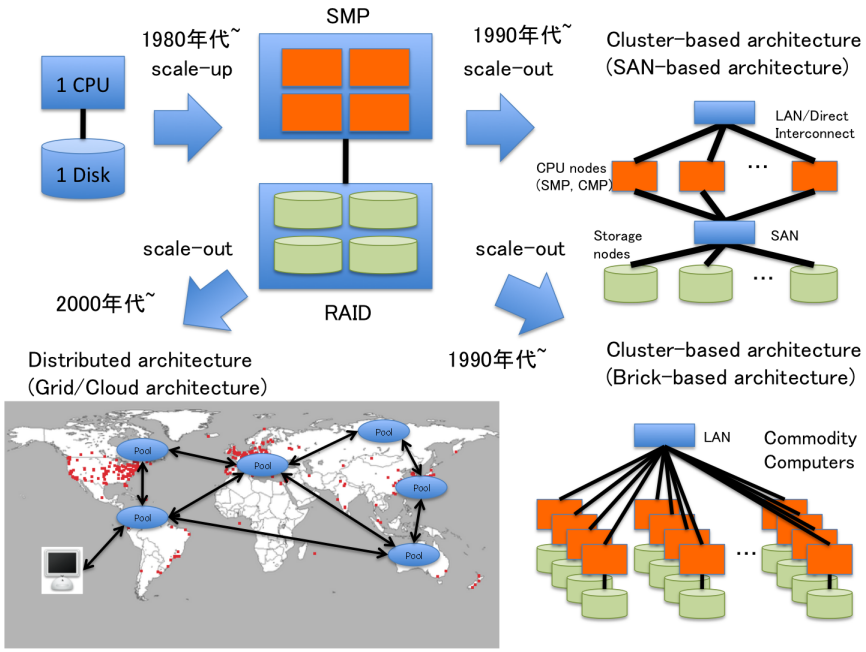

図 2.1 計算処理アーキテクチャの変遷

年頃からクラスタベースアーキテクチャからさらにスケールアウトさせるために，広域に分散するクラスタベースアーキテクチャを広帯域ネットワークで接続したアーキテクチャ(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) が登場してきた)(図 2.1).

クラスタコンピューティングにおいて，異なる計算機間でデータを共有する場合，NFS(Network File System)[56] が用いられてきた．グリッド環境において，NFS はデータ共有のために利用できない．これは，異なる組織間でデータを共有するために NFS を用いた場合，NFS クライアントと NFS サーバの通信においてインターネットを経由するため，遅延やジッタの影響を大きく受け，ファイル I/O 性能低下を招くためである．このため，データ共有は，データをテープドライブにコピーし物理的に送付する方法や，FTP などのデータ転送プログラムを用いていた．現在は，計算機の能力も向上し，広域ネットワークも 10Gigabit Ethernet など広帯域となり，GridFTP などのグリッド技術を用いて高速にデータ転送する環境が整ってきた．

本研究では，主にファイルシステムを対象とする．これは，本研究で対象としているグリッドコンピューティングで扱うアプリケーションが，入出力のためにファイルを用いているためである．

2.2. メタデータ

科学技術アプリケーションでデータを表現するために利用されているメタデータについて述べる．本節では，メタデータの定義について述べ，メタデータの用途について述べる．また，本研究で取り扱うメタデータについて述べる．

2.2.1 メタデータ

メタデータは，”data about data”，データについて記述されたデータを意味する．近年，インターネットにおいて，膨大なコンテンツが流通している．これらのコンテンツの検索，利用，再利用するために記述されたデータがメタデータである．一般的にメタデータには，インデックス，属性，記述子が含まれる．メタデータが持つ要素の例は，言語，フォーマット，タイトル，アクセス権限，撮影場所，時刻などが挙げられる．

2.2.2 メタデータの用途

メタデータは，情報資源を効率よく探し出すために用いられる．メタデータは，情報資源に記述されたデータであり，記述するための規格，表現形式が決定されており，情報資源を探し出すために，情報資源に付与されている情報資源の場所やその他の情報などの記述を含むデータである．

1960 年代頃からファイルシステムにおける i-node などのファイル属性を管理するためにメタデータを利用してきた．また，1970 年代頃から，データベース管理システムにおいて，各テーブルの定義，データ型などを管理するためにメタデータを利用してきた．近年では，Web ページに記述された内容について，それが何を意味するのかを表すメタデータを一定の規則に従って付加することにより，計

算機が効率よく情報を収集・解釈可能にするセマンティック Web が登場してきている．セマンテック Web は，インターネットをデータの集合から知識のデータベースに進化させる試みである．

2.2.3 Dublin Core

汎用的なメタデータフォーマットである Dublin Core[1] について述べる．Dublin Core とは，図書館の書誌情報を管理するために 1995 年に Dublin で開かれたワークショップにおいて標準化されたメタデータである．Dublin Core は，WWW(World Wide Web) 上における資源に関する情報を記述し，有用な情報の探索・発見に役立てることが可能である．Dublin Core の 15 の基本要素を表 2.1 に示す．Dubin Core は，データグリッドシステムにおいてよく利用されるメタデータフォーマットである．Storage Resource Broker (SRB) と呼ばれるグリッド向けデータ管理システムでは，Dublin Core をシステムレベルメタデータとして利用している．

2.2.4 特定アプリケーション向けデータフォーマット

本節では，画像データ，気象データや，天文学の画像データにおけるデータフォーマットを説明する．データフォーマットが定義されているデータは，ヘッダ部やデータ部など構造が決まっており，ヘッダ部などに記述されている内容をメタデータとして利用可能である．

- EXIF
  EXIF 情報とは，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．

- GRIB
  GRIB(GRIdded Binary)[14] は，WMO (World Meteorogical Organization)[27] によって 1985 年に提唱された気象データ向けの格子点データフォーマットである．GRIB は，次の 5 つのパートから構成されている．

1. データがGRIB形式であることを示すデータ，バージョン，データサイズ
2. データの製造元，高度，単位，観測日時，格子点が表す物理量
3. 格子点の数，種類など格子点に関する情報
4. 格子点データ
5. ファイルの終わりを示すデータ

- NetCDF
  NetCDF(Network Common Data Form)[13]は，非営利団体UCAR(University Corporation for Atomospheric Research)[25]のプロジェクトUnidata[26]で提案されているデータフォーマットで，GRIBと同様に，大量の格子点のバイナリデータが格納されている．NetCDFは，GIS(Geographic Information System)など衛星データのフォーマットとしても利用される．ncdumpコマンドにてヘッダ部分を取得することが可能である．NetCDFは，2つのパートから構成されている．
  - ヘッダ部
    変数のデータ以外の次元，属性，変数の情報
  - データ部
    固定サイズのデータ，もしくは，可変長サイズのデータ

- FITS
  FITS (Flexible Image Transport System)は，天文学分野においてデータ交換に用いられるデータフォーマットである．FITSは，天体スペクトルのデータ，X線観測のイベントデータなど天文分野で取り扱うデータを扱うことが可能な汎用フォーマットである．FITSは，ヘッダ部とデータ部から構成されている．
  - ヘッダ部
    データの目的，種類，構造，バイト数，レコード数

- データ部
  データ配列

このようにデータフォーマットが標準化や団体によって定義されている場合，ヘッダ部や各パートにおいてのデータ構造をアプリケーションメタデータとして利用可能である．

表 2.1 Dublin Core の基本要素

要素名	定義とコメント
title	リソースに与えられた名前． 通常，リソースの公式な名称を指す.
creator	リソースの内容に主たる責任を持つ人や組織．通常，その名前を記す.
subject	リソースの内容に含まれるトピック. トピックを示すキーワードやキーになるフレーズ，分類コードを使う.
description	リソースの内容の説明. 要約，目次，文書の内容を表現した画像への参照，あるいは自由形式の説明文
publisher	このリソースを提供している母体．通常，その名前を記す.
contributor	リソースの内容に協力，貢献している人や組織，サービス. 通常，その名前を記す.
date	リソースのライフサイクルにおける主要な出来事に関連する日.
type	リソースの内容の性質もしくはジャンル. 一般的な分野，機能，内容の集約度などを示す用語を用いる.
format	リソースの物理的あるいはデジタル化の形式. 主として，メディアタイプや量（サイズ）を示す.
identifier	リソースへの，曖昧さのない参照. URI，DOI(Digital Object Identifier)，ISBN など.
source	リソースが別のリソースの全体もしくは一部から 派生したものであるときは，その元リソースへの参照.
language	リソースの内容を記述している言語.
relation	関連するリソースへの参照.
coverage	リソースの範囲もしくは対象．場所（地名，緯度経度），時間区分 （時代，日付，期間），管轄区分（管理責任者名）などの分類を記述
rights	リソース内に保持される，あるいはリソースに適用される権利に 関する情報．知的所有権，著作権，財産権などについての 言明もしくはその情報を提供するサービスへの参照.

2.3. データ管理におけるメタデータの利用方法

本節では，クラスタ環境やグリッド環境，クラウド環境におけるデータ管理のためのメタデータ利用方法について述べる．データ管理におけるメタデータの利用方法は，ファイルカタログ機能とファイル検索機能の 2 点に分類できる．

2.3.1 ファイルカタログ機能

ファイルカタログとは，データのもつ i-node などのファイル属性情報や，ファイルがどこに保存されているか (ファイル名，ディレクトリ) を保持するための機能である．クラスタコンピューティングにおける分散ファイルシステムなどでは重要な機能である．例えば，Lustre はクラスタ向け並列ファイルシステムで，１つのストレージサーバ (Object Storage Server, OSS) の中の複数のディスクや，複数のストレージサーバを論理的に束ね，１つのファイルシステムを構成するシステムである．Lustre は，ストライピング機能を持ち１つのファイルを複数のディスクドライブ (Object Storage Target, OST) に保存することが可能である．Lustre では，１つのファイルを複数の OST に保存するため，どの OST からファイルを読み出すのかをメタデータで管理している．

2.3.2 ファイル検索機能

ファイル検索とは，データに関するキーワードなどを用いて，目的のデータを発見するための機能である．一般ユーザは，Google Desktop や Spotlight などを用いてファイル検索を実現している．Google Desktop，Spotlight などはファイルやメール，その他のデータを全文検索するためにインデキシングを行い，ファイル検索用データベースを構築している．ユーザのキーワードなどによる検索の結果からデータのリストを作成し，ユーザへデータアクセスを提供する．

2.4. 科学技術計算アプリケーションとメタデータ

本節では，科学技術計算アプリケーションにおけるデータの特性について述べ，メタデータを用いたデータアクセスが適したパラメータサーベイ型のアプリケーションについて述べる．

2.4.1 科学技術計算アプリケーションとメタデータ

科学技術計算シミュレーションを実施する研究者 (以下，ユーザ) は，シミュレーションモデルを構築する．シミュレーションモデルは，1 つのモデルを複数人で構築する場合や，別々のユーザがそれぞれ 1 つずつモデルを構築する場合などがある．このとき，ユーザは他のユーザが構築したモデルやデータは知らない．そのため，ユーザ間でデータ共有する場合，どのようにすればよいかが問題となる．

本研究では，他のユーザが共有しているデータの内容がどのようなものか，いつ，どこで誰が作成したデータがどういったものかを認識するための手間をデータ参照コストと呼ぶ．ユーザは，データ参照コストをどうにかして削減したいと考えている．例えば，天文学関連では，望遠鏡などの装置によって，データ形式が異なる．そのため，ユーザはアーカイブデータの取得先を明示的に知り，取得したいデータが存在するサイトにてデータを検索し，取得しなければならない．

これらのアーカイブデータに対して，メタデータを記述することは，異なるデータを抽象し，データの特徴量を記述することで検索可能となる．ユーザは，メタデータを用いたデータアクセスにおいてはデータアクセスの効率化，データ参照コストの削減が可能となる．

また，ユーザは，個々のシミュレーション結果を個別のデータとして管理するのではなく，シミュレーションに対しての結果データをグループとして扱いたいと考えている．ユーザは各シミュレーションに対する個々のデータを関連づけて管理するためにメタデータを記述していた．しかしながら，個別のデータに対してメタデータを記述することはユーザにとって大きな負担となっていた．ユーザは，メタデータを自動的に記述する機能を求めている．

科学技術計算アプリケーションは，最終的な結果を出力するまでに，単一のジョ

ブで完結するとは限らず，前処理や後処理を含む複数のジョブで構成されることが一般的である．例えば，メイン処理の前段階として，入力データから処理に必要な部分だけを取り出すために処理する．メイン処理の後段階として，メイン処理で出力されたデータを統計解析や可視化の処理をする．クラスタ環境における複数のジョブの連携では，多くの場合スクリプトプログラムを用いていた．しかし，グリッド環境ではジョブの連携が複雑化するため，単純なスクリプトでは対応できない場合がある．ジョブの流れをワークフローを用いて表現し，ジョブを実行する手法が提案されている．ワークフローシステムは，グリッド上の計算資源を有効に利用するために重要なシステムである．ワークフローとは，複数のジョブの依存関係を記述し，ワークフローエンジンが計算機資源管理システムと連携することで，ジョブを依存関係順に効率よく実行する．例えば，ワークフローシステムでは，前処理に 1CPU，メイン処理に 16CPU，後処理に 1CPU を割り当て，ジョブを実行することが可能である．ジョブ間でのデータ転送に，メタデータを用いることができれば，ユーザは，どのジョブにどのデータを割り当てるかなどを意識しなくてもジョブを実行可能となる．

2.4.2 科学技術計算アプリケーションにおけるデータ特性

グリッド環境やクラウド環境で実行される e-Science 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．e-Science アプリケーションにおけるデータの特性としては，以下の点が挙げられる．

- アプリケーションが取り扱うデータ数の増加
- データサイズの増加
- データのライフサイクルの二極化

e-Science のアプリケーションが取り扱う 1 ファイルのデータサイズや 1 つのジョブで取り扱うファイルの個数が年々増加傾向にある．生命科学分野においてアプリケーションが扱うファイルサイズとファイル数の関係を，図 2.2 に示す [71]．図 2.2 は縦軸がデータのデータサイズ，縦軸がデータの個数を示している．図中の

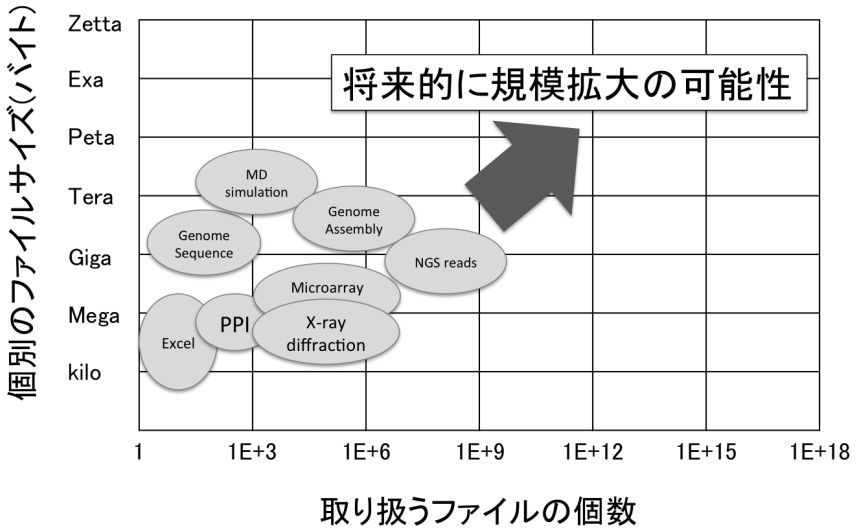

図 2.2 生命科学分野のアプリケーションにおけるファイルサイズとファイル数の関係

円で示した領域は各アプリケーションが扱うデータサイズ，データの個数を示している．図 2.2 より，アプリケーションが取り扱う最大のデータの個数は 10^9 個，最大のデータサイズはペタバイトスケールであることがわかる．さらに今後，生命科学分野においても将来的に取り扱うデータ量はますます増加していくと予想されている．これは生命科学分野に限らず，他の分野においてもアプリケーションが取り扱うデータ量は増加傾向にある．e-Science におけるアプリケーションから生成されるデータ量を，表 2.2 に示す．年間のデータ生成量は，多いもので数 PB である．1 つのデータサイズは，数 kB から数百 GB である．

表 2.2 アプリケーションが生成するデータ量

	高エネルギー物理学	天文学	気象観測	バイオサイエンス
年間データ生成量	数 PB	数 TB 〜 数 PB	数 TB	約 10GB 〜 数百 GB
データサイズ	0.5KB 〜 数百 GB	数 MB	〜 数十 MB	〜 数百 kB

単位: $K = 10^3, M = 10^6, G = 10^9, T = 10^{12}, P = 10^{15}$

データサイズ: データひとつあたりのデータサイズ

次に，データのライフサイクルについて説明する．データのライフサイクルとは，データが生成されて活用され，アーカイブされ，破棄されるまでの一連の流れを意味する．バイオサイエンスや気象シミュレーションなどの多くのデータは生成された後にすぐに解析し，処理が終了するとアーカイブされる．一方で，CERN の LHC などの大規模実験施設から出力されたデータのライフサイクルは年単位となっている．これらのデータは，アーカイブされた後の参照が多い．

本研究では，バイオサイエンスや気象シミュレーションのように，データのライフサイクルが短いデータにおけるデータの鮮度に注目する．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可能になるまでの時間と定義する．データの鮮度が必要という場合には，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．つまり，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でなければならない．

2.4.3 科学技術計算アプリケーションとデータ鮮度

グリッド，クラウド環境で実行されるアプリケーションとデータ鮮度について述べる．

High Performance Computing (以下，HPC) は，単位時間内に多くの計算を処理することである．HPC アプリケーションのひとつの例としては，タンパク質立体構造予測アプリケーションがある．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処理を行う．異なるパラメータを用いて得られたデータサイズが数 KB から数 MB，数千個の結果データを統計的に解析し，最終的な結果を得る．このように 1 つのプログラムに対して，異なる入力パラメータを与え，同時にプログラムを実行するアプリケーションをパラメータサーベイアプリケーションと呼ぶ．パラメータサーベイアプリケーションでは，ユーザが試行錯誤してパラメータの再調整し，ジョブを再実行することが可能である．パラメータの再調整時には，現時点で利用可能なデータを収集し，統計処理などを実行する必要があるため，データ鮮度は短くなる．つまり，データに対してメタデータが利用可能になるまでの時間が

短くならなければならない．

また，クラウド環境は，科学技術計算の他にも，Web サービスを実行するために利用される．例えば，ユーザは，Web サーバを 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の状況把握のために，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．ログデータにおいても，メタデータを用いたデータアクセスが有効ではないかと考える．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毎に分類してログデータを管理するなどがある．例えば，日付毎，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．

本研究では，データの鮮度が必要とされる，HPC アプリケーション，特に，パラメータサーベイアプリケーションを対象とする．

2.4.4 パラメータサーベイアプリケーション

本節では，本研究が対象とするパラメータサーベイのアプリケーションについて述べ，メタデータの利用例を述べる．

パラメータサーベイとは，広大なパラメータ空間に対して同じ計算を行い，パラメータ変動に対してどのように挙動するかを求めるための計算方法である．パラメータサーベイの例としては，モンテカルロシミュレーション，気象シミュレーション，たんぱく質立体構造予測 (SimFold[63] など) が挙げられる．パラメータサーベイ計算は，各計算間で依存関係がないため，独立かつ並列に実行可能である．パラメータサーベイアプリケーションでは，独立する大量の計算処理をグリッド上の計算機上に配布し，同一のプログラムに対して，異なる入力パラメータを用いて同時実行する．ユーザは各シミュレーション結果を収集し，統計解析処理して最終的な結果を得る．

パラメータサーベイのアプリケーションをグリッド環境に適用した例として，蛋白質立体構造決定のための遺伝アルゴリズムのグリッド化 [72] やグリッド環境上での気象シミュレーション [74] がある．

パラメータサーベイアプリケーションの実行には，数十秒で終了するものや，数十時間，数日単位の時間がかかるものがある．ユーザは，これらのアプリケーションをワークフローを用いて効率的に実行したいと考えている．例えば，パラメータサーベイアプリケーションでは，入力したパラメータによってシミュレーション結果の善し悪しが決定する．しかし，ワークフローで記述したジョブが全て終了するまで，シミュレーション結果は不明である．このような問題を解決するために，クラスタ環境でのたんぱく質立体構造予測システム Rokky に対して，ユーザがジョブ実行中に試行錯誤し，パラメータの調整が可能な提案がされている [67]．蟻川らの提案によって，ユーザが入力するパラメータについて試行錯誤し，良くない結果に関しては実行中にジョブを破棄し，パラメータを修正し，ジョブを再実行し，より精度の高い結果を短時間で得ることが実現されている．クラスタ環境において，パラメータサーベイアプリケーションは，Sun Grid Engine(SGE)[58] などのジョブスケジューリングシステムを用いて実行される．パラメータサーベイアプリケーションは，SGE などのジョブスケジューリングシステムに対して，アレイジョブ[1]として実行する．ジョブスケジューラシステムでは，全てのジョブが終了した後に，ユーザへメール等で連絡することが可能である．グリッド環境において，アレイジョブ実行中に終了しているデータだけを回収し，途中経過を確認することは困難である．科学技術計算アプリケーションで取り扱うデータに対して，メタデータを付与しておくことで，ジョブが終了したデータのみを回収し，途中経過を確認することが可能となる．

2.4.5 パラメータサーベイのアプリケーション性能分析

前項で説明した，グリッドやクラスタ環境でのパラメータサーベイアプリケーションの性能について分析する．各シミュレーションにおける，実行時間，取り

[1]同じジョブでパラメータを変更して繰り返し実行するジョブ

扱うファイルサイズ，ファイル量，ノード数 (CPU 数)，VO に参加しているサイト数について注目する．

グリッド環境上での気象シミュレーション [74] では 100 日分の単一シミュレーション時間が 180 秒から 250 秒，3ヶ月分の 50 サンプルシミュレーションを 3 サイト，10 ノードで実施した際のシミュレーション時間は，1,200 秒である．シミュレーションで取り扱うデータは，入力データが 3.4KB，出力データが，1,350KB である．

蛋白質立体構造決定のための遺伝アルゴリズムのグリッド化 [72] では，VO 上に存在する 1000CPU を数時間単位のシミュレーションが実行可能である．論文中では，VO 上に存在する 5 サイトの 1000 を超える CPU を用いている．シミュレーションで取り扱うデータは 7.76KB である．

タンパク質立体構造予測アプリケーション Rokky[67] では，与えられたタンパク質のターゲットを 48 時間など時間制限があるなかで解析する．Rokky では 1 サイトにおいて，30 から 40 ノードを用いてシミュレーションを実行している．Rokky が取り扱うデータは，10KB から 1.1MB である．

パラメータサーベイアプリケーションでは，ワークフローシステムを用いて実行する．メイン処理の後に実行される後処理において，依存するジョブにおける待ち時間は，メイン処理のシミュレーション時間，データ転送時間に依存する．後処理は，メイン処理が出力するデータ全て揃ってから実行する．本節で説明したシミュレーションで実行時間が短いものは，気象シミュレーションで，1 つのサンプルシミュレーションを実行した場合で 180 秒程度である．シミュレーション時間 180 秒のうち，メイン処理実行時間は 150 秒，データ転送時間は 10 秒程度である．データ鮮度を考慮すると，1 つのメタデータが利用可能になるまでの時間は 10 秒程度までである必要がある．

これらのアプリケーションが広域分散環境で動作するための最低要件として，満たすべき性能要件を以下のように設定した．広域分散メタデータ収集・管理システム，メタデータを用いたデータアクセスシステムは，以下の性能を満たすようにシステムの設計をする必要がある．

- 1 シミュレーションあたりでの取り扱うデータ

  - データサイズ: 10KB から 1.4MB
  - データ量: 1,000 個 から 100,000 個
- VO に参加しているサイト数: 3 から 10 サイト
- 1 シミュレーションあたりでのノード数: 1 サイトあたり 10 から 40 ノード
- 1 シミュレーションあたりの実行時間: 180 秒から 48 時間
- メタデータが利用可能になるまでの時間: 許容範囲 約 10 秒

2.5. メタデータの収集・管理，メタデータを用いたデータアクセスにおける要求事項

本節では，広域分散環境でのメタデータ収集・管理，メタデータを用いたデータアクセスにおける要求事項をデータ管理システム，データグリッド，データアクセスシステム，メタデータ収集，メタデータ管理の観点から説明する．

2.5.1 データ管理システムへの要求事項

まず，分散データ管理システムに対する一般的な要求事項について述べる．分散データ管理システムにおいては，以下の 4 点が基本的な要求事項として挙げられる [49]．本研究では，これらの要求事項の中で，Discovery，Access について注目する．

1. Discovery (データの存在確認，発見)
2. Access (データへのアクセス)
3. Understanding (データの解釈)
4. Policy management (データのポリシー管理)

2.5.2 データグリッドシステムにおける要求事項

グリッドコンピューティングにおけるデータグリッドシステムとして求められる 2 つの機能について述べる．

1 つめは，グリッド環境でのデータアクセスにおけるユーザの負担を軽減するために，データグリッドではファイルやメタデータのカタログを作成し，ユーザへデータ属性等を用いたデータアクセス機能を提供することが求められている．Thomson ら [47] は，科学技術計算を行うユーザに対して，実際のストレージやファイルの場所は隠蔽しながら，データに対してブール値やキーワードによる検索，アクセスを行うためのセマンテックベースのファイルカタログを含む機能や，ファイルシステムのすべての機能を提供する必要があると述べている．また，Finkelstein ら [30] は，データグリッドにおいて，ユーザはデータ属性に基づいてデータの問い合わせやデータアクセスが行えることを期待し，さらにアプリケーションがグリッドなどの環境においても再利用できることを望んでいると述べている．

2 つめの要求は，各組織毎のグリッドシステムの構成方法に依存しないことである．各組織におけるデータ管理システムの構成方法を図 2.3 に示す．図 2.3 の (1) では，各ワーカーノードに接続されたファイルシステムに出力する．(2) では，負荷に基づいて出力先のファイルシステムが選択され，ワーカーノードからデータが出力される．また，(3) では SAN(Storage Area Network) 環境を構築し，ワーカーノードは FC などで接続されたストレージにデータを出力する．または，専用の NFS (Network File System) 環境を構築し，ワーカーノードは NFS サーバが提供する領域をマウントし，データを出力する．広域分散環境でのデータアクセスを実現するためには，各組織でのデータグリッドの構成方法にかかわらずデータ，メタデータが利用可能になる必要がある．

2.5.3 メタデータを用いたデータアクセスにおける要求事項

広域に分散するデータに対して，メタデータを用いてデータの特徴量などの属性と属性値を記述することで検索可能となることが求められる．ユーザは，メタ

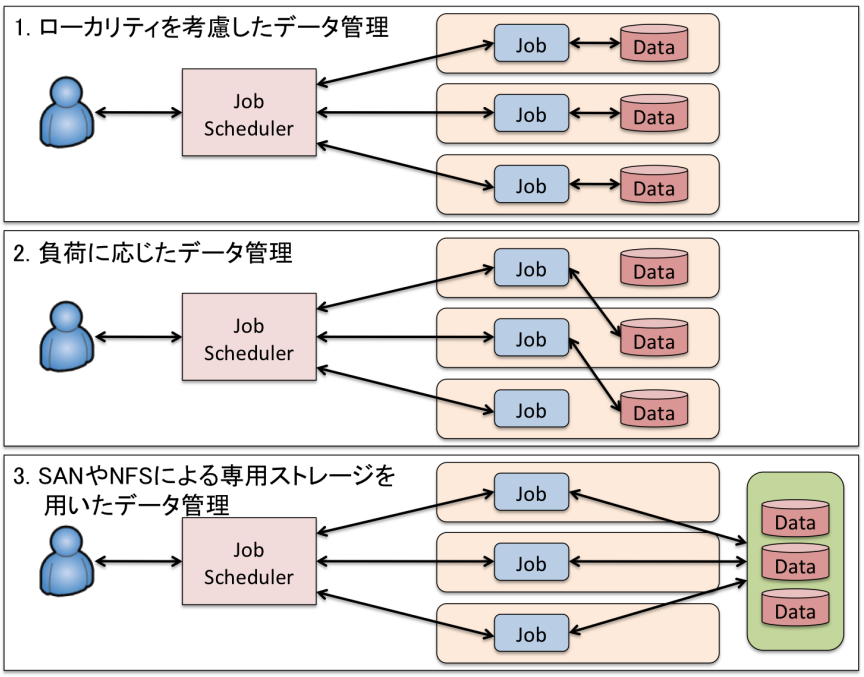

図 2.3　各組織におけるデータグリッドの構成方法

データを用いたデータアクセスが実現することで，データ参照コストの削減が可能となる．

パラメータサーベイアプリケーションが持つアプリケーションメタデータの属性値は，文字列，数値や日付などである．メタデータによるデータ検索を行う際に，データアクセスシステム側で，条件に一致する，もしくは，条件の範囲に一致する検索をサポートする必要がある．

多くの科学技術計算アプリケーションは，UNIX 上で動作することを前提としたアプリケーションである．ユーザは広域分散環境においても，既存アプリケーションを再利用できることを望んでいる．パラメータサーベイアプリケーションにおいても，Perl，Python などの LL 言語やシェルスクリプトを用いており，メタデータを用いて収集したデータに対して，既存のアプリケーションを改変することなく利用できる必要がある．そのため，データグリッドシステムでのデータアクセス機能に必要な機能として，POSIX ファイルシステム API との互換性を

持つ必要がある．

2.5.4 広域分散環境でのメタデータ収集における要求事項

本項は，広域分散環境でのメタデータ収集における要求事項を述べる．

Dublin Core などの汎用メタデータは，リソースの参照情報を記述することは可能であるが，アプリケーションメタデータまでは表現することは困難である．Dublin Core で EXIF，NetCDF などのアプリケーションメタデータを表現するためには，Dublin Core の description フィールドや，relation フィールドなどへ記述する．この場合，XML フォーマットで各アプリケーションメタデータを記述しなければならないため，ユーザの負担が増える．従来，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．また，自動的にメタデータを生成する場合においても，多くのステップにおいてユーザによる操作を伴うケースが存在した [45]．ユーザにとって，アプリケーションメタデータを記述することは大きな負担となる．ユーザからの要求としては，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．そのために，自動的にメタデータを収集する機能が必要となる．

また，自動的にメタデータ収集をする仕組みが必要であるが，メタデータを収集する際に考慮すべき点について述べる．パラメータサーベイアプリケーションは，ワーカーノードの計算資源 (CPU，メモリ，ディスクなど) を 100%に近い使用率で処理する．また，パラメータサーベイアプリケーションは，1 つのジョブあたり数 KB と小さなファイルが，1,000 個から 10,000 個単位で出力される．自動的にメタデータを収集するシステムは，ジョブを実行するワーカーノードに負荷を与えないことが求められる．

2.4.5 項で述べたように，試行錯誤を考慮したワークフローシステムを利用しながら，実行するパラメータサーベイアプリケーションや，1 つのシミュレーションの実行時間が数分程度と短い場合，データ鮮度が必要となる．つまり，メタデータがすぐに利用可能になっている必要がある．そのため，データの鮮度を考慮したメタデータ収集方法が求められる．

2.5.5 広域分散環境でのメタデータ管理における要求事項

グリッド環境でメタデータを管理する上での問題点，要求事項を述べる．

アプリケーションメタデータは，アプリケーション毎，ユーザ・グループ毎によって利用目的，利用方法，記述内容が異なるため，アプリケーションメタデータのフォーマットを一般化することは不可能である．そのため，アプリケーション毎にメタデータを定義し，アプリケーションもしくは，コミュニティ共通のメタデータフォーマットが制定されている．そのため，多くのデータグリッドシステムが，アプリケーションに最適なデータ管理，メタデータ管理を行うシステムとなる．アプリケーションメタデータでは，ユーザやアプリケーション毎にメタデータのスキーマが定義されている．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，テーブル構成が複雑化する．また，ワーカーノードでアプリケーションメタデータを収集する場合には，ワーカーノード側でもアプリケーションメタデータのスキーマを管理しなければならない．そのため，ワーカーノード側でメタデータを管理する場合，全ての異なるアプリケーションメタデータのスキーマを管理しなければならない．そのため，データグリッドにおけるメタデータ管理システムは，ユーザグループの要求に基づいた特定アプリケーション毎のメタデータ管理システムが構築されてきた．汎用的に利用でき，全てのアプリケーションに対して，1 つのアーキテクチャを用いて解決できる手法は提案されていない．本研究では，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メタデータを広域で管理する上で，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持しながら，複数のメタデータへ対応できることが求められる．

メタデータを用いたデータアクセスにおいて，ユーザからのデータ要求が，メタデータ管理サーバがインターネットなどの高遅延ネットワークを経由すると，データアクセス性能に大きく影響する．そのため，高遅延に強いメタデータ I/O を実現する，もしくは，メタデータ管理サーバを極力利用するユーザの近くに配置するなどユーザ側からできる限り高速なメタデータ I/O を実現することが求められる．また，ファイルシステムにおける負荷の 50%がメタデータ I/O である [32][42]．そのため，広域分散環境でのファイルシステムにおいてメタデータ I/O

の負荷を抑えるために，メタデータの配置や複製処理を考慮し，高速なメタデータ I/O を実現することが求められている．

グリッド環境で実行されるパラメータサーベイアプリケーションでは，ユーザのジョブはメタジョブスケジューラによって各サイトに割り当てられ，各サイトの Sun Grid Engine や PBS などのジョブスケジューラによってワーカーノードにジョブが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，ユーザはどこにメタデータ管理ノードがあるかを意識してデータアクセスを行わなければならない．また，データグリッドでは，組織を越えてデータ複製する場合がある．データが別の組織へ移動・複製された際，メタデータの一貫性についても保証しなければならない．しかしながら，パラメータサーベイアプリケーションでは，各ジョブの結果を回収し統計処理するため，結果データの全てが必要となるわけではない．そのため，メタデータを分散管理した場合においても，メタデータの一貫性については，強い一貫性は必要とならない．

2.4.5 項で述べたように，試行錯誤を考慮したワークフローシステムを利用しながら，実行するパラメータサーベイアプリケーションや，1 つのシミュレーションの実行時間が数分程度と短い場合，データ鮮度が必要となる．つまり，メタデータがすぐに利用可能になっている必要がある．そのため，データの鮮度を考慮したメタデータ管理方法が求められる．

2.6. 要求事項の分析・まとめ

本節では，これまでに述べた様々な要求事項の分析を行い，ユーザがグリッドやクラウド環境において，メタデータを用いたデータアクセスを実現するために，分散メタデータ管理システム，データアクセスシステムに必要な要求事項を議論する．

検討するべき要求事項は，以下の点である．本研究ではこれらの要求を満たすグリッド環境におけるメタデータを用いたデータアクセス機構，およびメタデー

タ管理機構について考える．

(要求事項 1) データアクセスインタフェース

1. データ参照コストの削減
   メタデータを用いたデータ検索機能が可能である．
2. 既存アプリケーションの再利用可能
   クラスタコンピューティング環境で動作していたアプリケーションをグリッド環境向けにユーザが改変する手間を削減する．
3. メタデータの記述性
   メタデータによってデータを検索する際に，アプリケーションが持つ属性に対して，値が一致するか，もしくは指定した範囲に一致することがサポートされる必要がある．

(要求事項 2) データのメタデータ収集

1. メタデータ収集の自動化
   ユーザによるメタデータ記述のコストを削減すること．
2. メタデータ収集時にシステムに対する負荷を与えない
   メタデータ収集時に，アプリケーションの実行性能へ影響を与えないこと．
3. データの鮮度を考慮したメタデータ収集
   パラメータサーベイアプリケーションにおけるシミュレーションの実行時間は，短い場合で 180 秒程度である．短い実行時間に対して，データの鮮度を考慮したメタデータ収集方法を提供する必要がある．

(要求事項 3) 広域分散環境におけるメタデータ管理

1. 1 つのメタデータ管理システムで複数のメタデータスキーマを管理
   グリッドミドルウェア上では，複数のアプリケーションが動作するため，各アプリケーションのメタデータスキーマを管理しなければなら

ない．アプリケーションメタデータスキーマを個別に管理すると，メタデータ管理コストが増加する．

2. メタデータ I/O の高速化
   ファイルシステムにおける負荷のうち，50%以上をメタデータ操作が占めている．メタデータを用いたデータ操作において，メタデータ I/O の高速化が必要となる．そのため，メタデータの配置，複製を考慮しなければならない．

3. メタデータ複製時の一貫性制御
   メタデータ I/O 高速化のために，メタデータを複製するが，パラメータサーベイアプリケーションでは，後処理における統計解析処理で，全てのデータが必要となるわけではないので，各メタデータ管理システム間でメタデータの不一致が発生しても大きな問題とはならない．

4. データの鮮度を考慮したメタデータ管理
   要求事項 2-3 と同様に，短い実行時間のシミュレーションなどに対して，データの鮮度を考慮したメタデータ管理方法を提供する必要がある．

第3章　関連研究

本章では，既存の広域分散環境でのメタデータを用いたデータアクセスシステムや，メタデータ管理システムなどの関連研究を述べ，また，2.6 節で述べた要求事項に対して，解決できていない問題点を述べる．

3.1. 関連研究の分類

メタデータを用いたデータアクセスは，適用する環境を問わず，POSIX ファイルシステム API との互換性あり，なしで 2 種類に分類できる．

- POSIX ファイルシステム API 互換性あり
  階層型ファイルシステムの名前空間によるデータアクセスを提供
- POSIX ファイルシステム API 互換性なし
  システム専用 API や，Web インタフェースを含む GUI によるデータアクセスを提供

また，これらのデータアクセスシステムにおけるメタデータの利用方法は次の 2 種類に分類できる．メタデータによるファイル検索機能を有しているシステムは，ファイルカタログ機能も備えている．

- メタデータによるファイルカタログ機能
- メタデータによるファイル検索機能

また，広域分散環境におけるメタデータを用いたデータアクセスシステムを実現するためには，広域分散環境でのメタデータ管理が必要となる．メタデータ管理システムを機能毎に分類し，関連研究を述べる．

3.2. メタデータを用いたデータアクセスシステム

本節では，メタデータを用いたデータアクセスシステムを，POSIX ファイルシステム API との互換性の有無で分類し，メタデータの利用方法別に述べる．

3.2.1 POSIX ファイルシステム API 互換システム

POSIX (Portable Operating System Interface) ファイルシステム API と互換性があるシステムでは，UNIX 環境で動作するアプリケーションが再利用可能である．

メタデータによるファイルカタログ

ファイルカタログは，クラスタファイルシステムなどで用いられる．代表的なシステムとして，Lustre[9]，Gfarm[57][69] がある．これらのシステムは，LAN やインターネット上に配置された，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が提供するマウントポイントを組み込み，ユーザ側からはローカルファイルシステムと同様なアクセスインタフェースを提供することである．Gfarm では，図 3.1 のように広域に分散するファイルシステムを統合し，仮想ファイルシステムをユーザへ提供する．ユーザは，既存ディレクトリツリーからどのファイルサーバに存在しているデータか意識せずに利用できる．Gfarm におけるデータアクセスの手順を図 3.2 に示す．ユーザは，gfarm2fs で，Gfarm の提供する領域をマウントする．gfarm2fs などの Gfarm クライアントは，メタデータサーバに対してファイルアクセス要求を送る．メタデータサーバは，該当するファイルがどのファイルシステムノードに配置されているかを Gfarm クライアントに対して通知する．Gfarm クライアントは，メタデータサーバから受け取ったファイルシステムノードへアクセスし，データを得る．ユーザは，直接メタデータを参照することはなく，Gfarm クライアントやファイルシステムノードのみがメタデータを利用する．Lustre や Gfarm

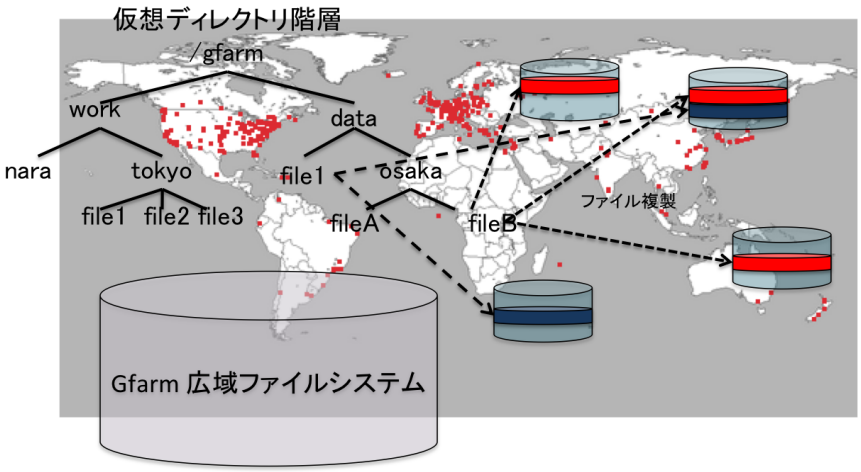

図 3.1 Gfarm アーキテクチャ

のメタデータは，ファイルシステムが保持する i-node などのファイル属性を用いている．これらのシステムでは POSIX ファイルシステムの API 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うことができ，既存のアプリケーション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．

Veeraraghavan らが提案している quFiles[50] は，メタデータを用いて異なるクオリティのデータを抽象化し，ユーザに対して同じファイル名であっても，状況に応じて，異なるファイルを提供する．例えば，'Foo' という動画ファイルがあった場合，PC からアクセスした場合は，高画質の Foo.mp4，携帯端末からアクセスした場合は，低画質の Foo.mp4，家庭用ビデオレコーダからアクセスした場合には，ビデオレコーダ専用フォーマットの動画を提供する．quFiles では，アクセスするデバイス毎にどのファイルへアクセスするかを Name Policy に定義しなければならない．

表 3.1 に本項で述べた関連研究を 2.6 項で述べた要求事項 (1) に照らし合わせた結果を示す．Gfarm，quFiles ともに，アプリケーションの再利用は可能であるが，quFiles は，Name Policy の記述によっては，同じ名前でも，異なるファイル

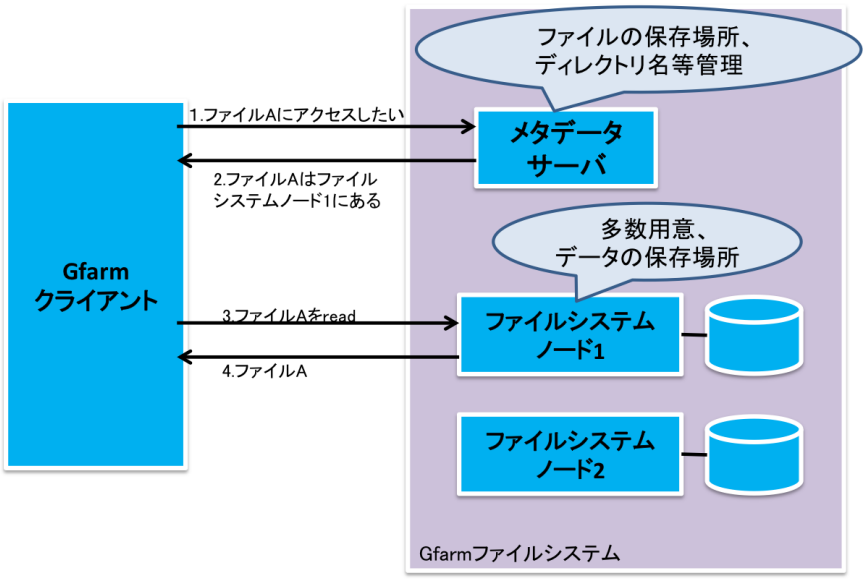

図 3.2 Gfarm 上でのデータアクセス

表 3.1 メタデータによるファイルカタログ機能 (POSIX 互換)

	Gfarm	quFiles
要求 (1)-1 データ検索可能	×	×
要求 (1)-2 アプリケーション再利用可能	○	△
要求 (1)-3 メタデータの記述性	–	–

にアクセスする可能性がある．これらのシステムでは，メタデータはデータの位置やアクセスするターゲットを管理するカタログであるため，データ検索機能は提供していない．そのため，メタデータの記述性も要件を満たしていない．

メタデータによるファイル検索

メタデータによるファイル検索は，ファイルに関するキーワードを用いて，階層型ファイルシステムやストレージからデータを発見しアクセスするための機能である．

Semantic File System (SFS) [41] は，既存のファイルシステムを拡張し，ファ

イル属性によって検索し，ユーザをナビゲート可能にしたシステムである．SFS では，Attribute directory と Value directory の A-V ペアと呼ばれるディレクトリの組を指定することで検索に一致したファイルを得ることができる．検索結果は，シンボリックリンクとしてユーザへ提供される．ユーザは，検索結果が表示される仮想ディレクトリ上でファイル，ディレクトリを編集や削除できない．また，SFS は通常のファイルシステム上のファイルを対象として，検索インデックスを作成する．検索インデックスは，2 分ごとにファイルの書き込みをチェックし更新する．SFS は，NFS サーバをベースとして実装されている．

滝田ら [73] は，全文検索システム Namazu[55] を用いてファイルの検索インデックスを構築し，検索ファイルシステムと通常のファイルシステムを区別なく利用することができるシステムを提案している．特殊なキーワードを伴うディレクトリ名が，検索キーワードとして変換され，検索式を生成し全文検索エンジンを用いてデータの検索する．検索したファイルの結果は，シンボリックリンクとしてユーザへ提供する．検索インデックスは，ファイル書き込み時に自動的に更新し，常に最新の内容に基づいた検索可能である．SFS や全文検索システム Namazu を用いたシステムはローカルシステムを前提としており，広域分散環境では動作することを想定していない．例えば，SFS をグリッドなどの広域分散環境で動作させた場合，メタデータインデックスへの問い合わせやファイルアクセスがネットワーク遅延による影響を大きく受け，応答時間が増大し，性能低下が生じることが容易に想像できる．

Perspective [34] は，P2P をベースとしたファイル管理システムである．Persepctive は，データのセマンティクスを記述して，ビューを作成し，データの集合を表示する．ファイルアクセスには，FUSE を利用している．Perspective では，P2P ベースでデータを送受信するが，ホームネットワークを想定しており，インターネットなどを越えてのデータ送受信は想定していない．

Spotlight[20] は，Apple の Mac OS X に標準搭載されているデスクトップ検索システムである．Spotlight は，検索窓にキーワードを入力することで関連するすべての対象をファイル検索できる．Mac OS X のファイルシステムは，HFS plus をベースしてファイルの内容から抽出したメタデータを同時に保持する．また，

表 3.2 メタデータによるファイル検索機能 (POSIX 互換)

	SFS	Namazu	Spotlight
要求 (1)-1 データ検索可能	○	○	○
要求 (1)-2 アプリケーション再利用可能	○	○	○
要求 (1)-3 メタデータの記述性	○	○	○

SmartFolder は Spotlight と連動した Mac OS X の仮想ディレクトリシステムである．SmartFolder では，Spotlight での検索結果を永続的に任意に作成した仮想ディレクトリにマッピングする．Spotlight は，Mac OS X でのみで動作し，グリッド環境で多く利用されている Linux では動作しない．

表 3.2 に本項で述べた関連研究を 2.6 項で述べた要求事項 (1) に照らし合わせた結果を示す．SFS，Namazu を用いた検索ファイルシステム，Spotlight は要求事項 (1) に関しては全て満たしている．しかしながら，どのシステムもグリッド環境のような広域分散環境下で動作することは想定していない．

3.2.2 非 POSIX ファイルシステム API 互換システム

本節では，POSIX ファイルシステム API と互換性のないデータアクセスシステムについて述べる．

メタデータによるファイルカタログ機能

HDFS(Hadoop Distributed File System)[7] は，Hadoop 向けの分散ファイルシステムである．HDFS は，メタデータを管理する NameNode と，データを管理する DataNode から構成される．図 3.3 に HDFS でのデータアクセスの流れを示す．HDFS クライアントは Distributed Filesystem へ open 命令を発行する．Distributed Filesystem は，NameNode に対して該当データのブロックの所在を問い合わせる．HDFS クライアントは，該当するデータのブロックの所在が判明すると，FSData InputStream へ読み込み命令を発行し，DataNode からデータを読み込む．HDFS では，ファイルシステムデータに対するストリーミングアク

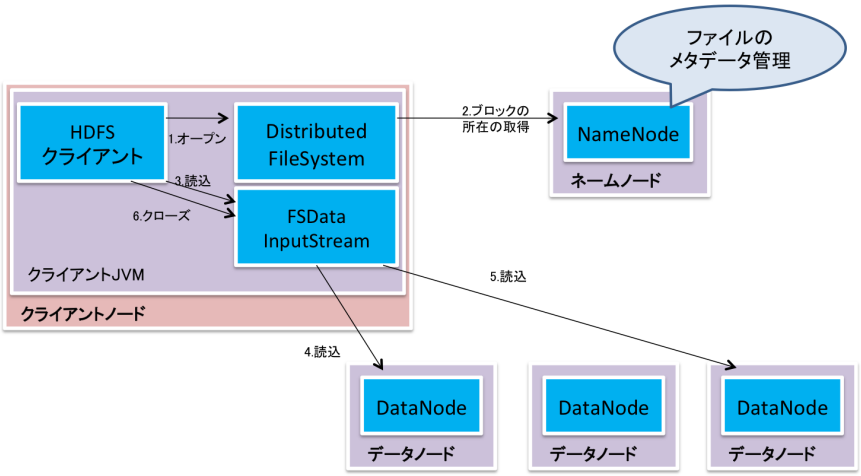

図 3.3 HDFS アーキテクチャ

セスを実現するため，POSIX ファイルシステム API の制約を弱め構築したシステムである．そのため，POSIX ファイルシステム API とは互換性がない．広域分散環境で Hadoop を動作させる場合，NameNode と DataNode の配置によって，アプリケーションの性能が著しく低下する．例えば，著者が実施した実験では，NameNode と DataNode 間で，10 ミリ秒の遅延が生じる場合で，ファイル書き込みスループットが最大 20 分の 1 以下になる．HDFS は，NameNode と DataNode 間の通信が高遅延であることを想定していないため，NameNode 間でのメタデータ同期や複製をしていない．

分散 Key-Value Store (KVS) システムにおいてデータの位置を管理するハッシュも，1 つのメタデータである．分散 KVS でのデータ管理システムの代表例としては，kumofs[70] が挙げられる．kumofs が提供する API は，memchaced[10] 互換の API(set/get) を提供している．memcached は，データとオブジェクトをメモリ内にキャッシュすることで，データベースからの読み込み回数を減少させ，データの読み込みを高速化する．kumofs は，非常にスケーラビリティのあるデータ管理システムである．分散 KVS では，小さなデータサイズのバリューを大量に格納することを想定したシステムが多い．そのため，分散 KVS は，グリッドなどで扱うファイルサイズの大きいデータを取り扱うことは比較的苦手である．しか

表 3.3 メタデータによるファイルカタログ機能 (非 POSIX 互換)

	HDFS	kumofs
要求 (1)-1 データ検索可能	×	×
要求 (1)-2 アプリケーション再利用可能	×	×
要求 (1)-3 メタデータの記述性	–	–

しながら，メタデータなどのデータを格納する場合には，十分な性能を発揮できると考えられる．P2P ベースや分散 KVS システムでのインタフェースはシンプルな API を提供している．これらのシステムでデータを取得する場合には，get API に，必要なデータのキーを引数として渡し，データを得る．

表 3.3 に本項で述べた関連研究を 2.6 項で述べた要求事項 (1) に照らし合わせた結果を示す．HDFS，kumofs ともに，メタデータによるデータ検索はできない．また，POSIX ファイルシステム API をサポートしていないため，アプリケーションの改変が必要となる．

メタデータによるファイル検索

WinFS[12] は，Microsoft 社が開発を行っていた次世代 Windows 用のファイルシステムである．WinFS は，従来の木構造による階層的な名前空間をベースとしたファイルアクセスだけではなく，様々なメタデータを用いたデータアクセスすることができる．メタデータを用いたデータ検索には，WinFS 独自の API を利用する．WinFS では，従来の Windows におけるファイル操作の API とは互換性を持たない．また，WinFS は開発が難航し，2006 年に開発が中止されている．

グリッドコンピューティングにおける代表的なデータ管理システムとして，Storage Resource Broker (SRB)[23][35]，iRODS[31] 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，ユーザやリソースに関するメタデータを管理する MCAT(Metadata CATalog) サービス，SRB エージェント，データを保管する SRB サーバ，SRB クライアントから構成される．

iRODS (integrated Rule-Oriented Data System) は，SRB をベースととした

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できる．iRODS では，iCAT (iRODS metadata catalog) がアカウント，リソースを管理する．iRODS は，iRODS サーバ，iRODS メタデータカタログ (iCAT)，iRODS ルールエンジン，ユーザインタフェースの 4 つのコンポーネントから構成されている．iRODS サーバは，データを管理する．iRODS メタデータカタログは，データベースにてデータのシステムレベルメタデータ，ユーザレベルメタデータを管理する．iRODS ルールエンジンは，データに対するアクセスルールなどを記述し実行するエンジンである．iRODS では，ユーザインタフェースとして Web インタフェースや GUI を提供する．

SRB や iRODS 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 は，大学をはじめとした研究機関や，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．SRB のシステムレベルメタデータを表 3.4 に示す．SRB のメタデータは Dublin Core をベースとして拡張したものである．ユーザレベル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．SRB では，ユーザが定義したメタデータを手動，または，自動的に収集する．しかし，ユーザレベルメタデータ毎に，テーブルの定義が必要となり，管理者の手間を煩わせる．SRB, iRODS は，データ検索やデータアクセスのために Web インタフェース，iCommands と呼ばれる SRB/iRODS 専用の UNIX コマンドを提供する．SRB では，データを検索するために，Web ブラウザでの検索インタフェースを提供している (図 3.4)．iRODS におけるデータアクセスの流れを図 3.5 に示す．アプリケーションは，iCommands などを経由して SRB エージェントへデータを要求する．SRB エージェントは，メタデータ

表 3.4 SRB のメタデータ

SRB Object Name	SRB においてユニークな名前
Collection Name	物理的なファイル名
Resource Name	論理的なファイル名
Data Type	データタイプ
DC.Title	Dublin Core title
DC.Creator	Dublin Core creator
DC.Publisher	Dublin Core publisher
DC.Date	Dublin Core date
DC.Type	Dublin Core type
DC.Format	Dublin Core format
DC.Identifier	Dublin Core identifier
DC.Source	Dublin Core source
DC.Language	Dublin Core language
DC.Coverage	Dublin Core coverage
DC.Rights	Dublin Core rights

管理サーバである MCAT へ問合せ，データの位置を得る．SRB エージェントは，該当データが存在する SRB サーバにアクセスする．アプリケーションは，SRB サーバ上で動作している SRB エージェントを経由して，該当するデータへアクセスする．

SRB や iRODS は，独自のコマンドラインや API を提供しているため，POSIX ファイルシステムを前提とした既存のアプリケーションを再利用することはできない．そのため，ユーザは，データグリッドシステムが提供する専用の API に合わせて，それぞれアプリケーションプログラムを修正する必要がある．

表 3.5 に本項で述べた関連研究を 2.6 節で述べた要求事項 (1) に照らし合わせた結果を示す．WinFS，SRB/iRODS ともに，メタデータによるデータ検索，メタデータの記述性は満たしている．しかし，データアクセスは専用 API を用いて

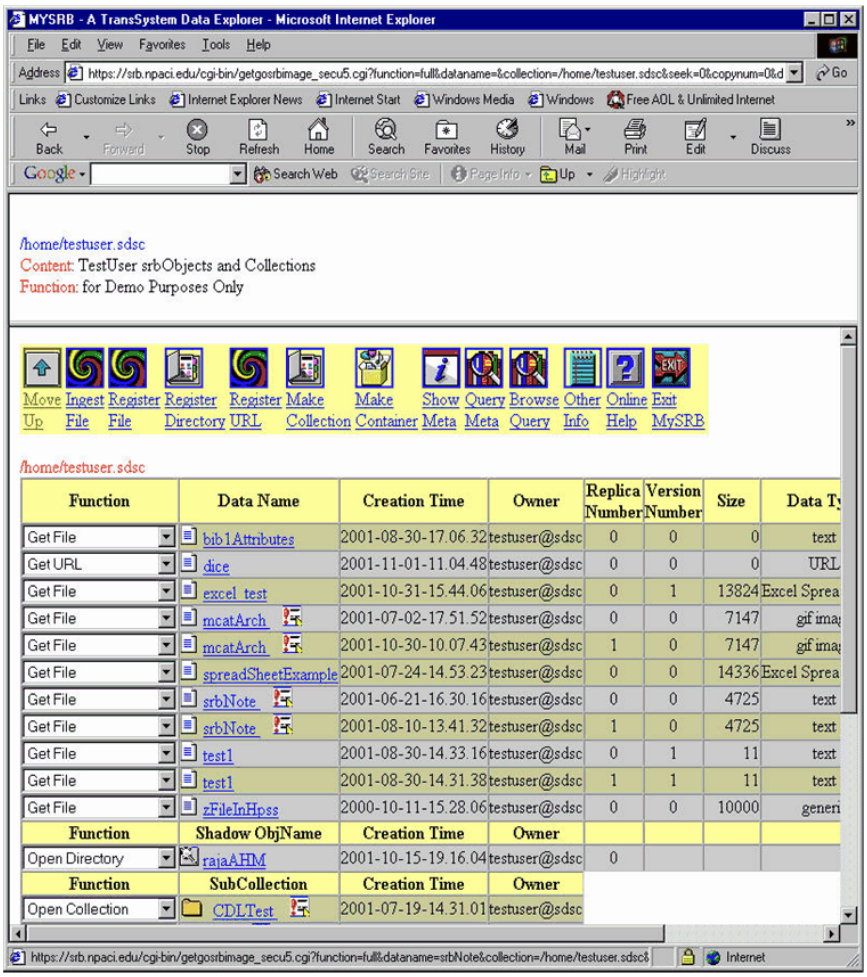

図 3.4 SRB のデータ検索インタフェース

いるため，アプリケーションの改変が必要となる．

3.2.3 メタデータを用いたデータアクセスのまとめ

メタデータを用いたデータアクセスシステムの分類を表 3.6 に示す．表 3.6 における POSIX 互換であり，ファイル検索機能を有するシステムが，2.6 節で述べた要求要件を満たす．しかし，3.2.1 節で述べた SFS などは，ローカルマシン上で動作することを前提に開発している．これらのシステムでは，メタデータインデックスをローカルマシン上でのみ保持しており，広域分散環境でのインデック

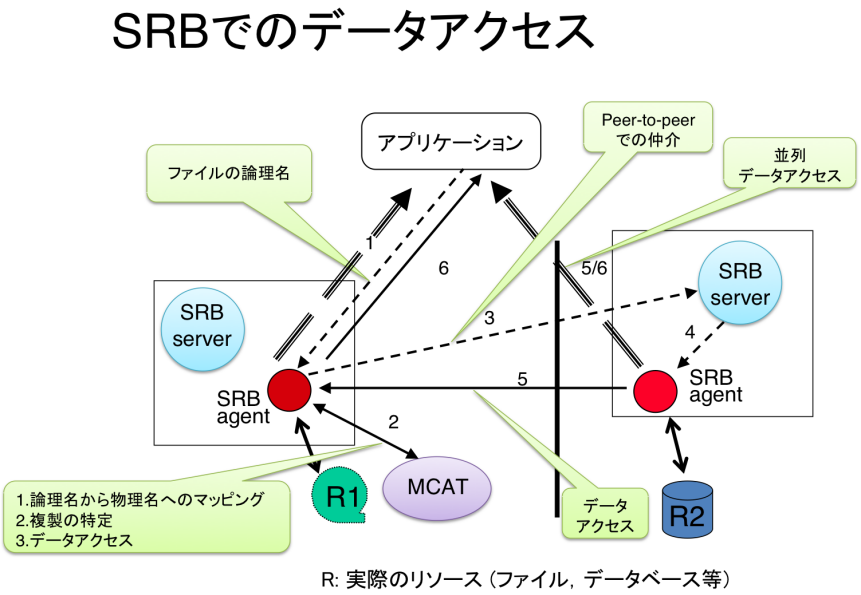

図 3.5 SRB におけるデータアクセス

表 3.5 メタデータによるファイル検索機能 (非 POSIX 互換)

	WinFS	SRB/iRODS
要求 (1)-1 データ検索可能	○	○
要求 (1)-2 アプリケーション再利用可能	×	×
要求 (1)-3 メタデータ記述性	○	○

スの複製処理や問い合わせ処理などを考慮していない．これらのシステムは，グリッドなどの広域分散環境で動作させた場合，メタデータの問い合わせ処理の応答時間がネットワーク遅延のために大きくなり，システムの性能低下が避けられない．

広域分散環境で動作する POSIX 互換なファイル検索可能なデータアクセスシステムを実現するためには，広域分散環境でのメタデータ管理システムが必要である．

表 3.6 メタデータを用いたデータアクセスシステムの分類

	POSIX	非 POSIX
ファイルカタログ	Lustre[9], Gfarm[57, 69], quFiles[50]	Hadoop HDFS[7], 分散 KVS(kumofs[70] など)
ファイル検索	SFS[41], Namazu を用いた 検索ファイルシステム [73], Perspective[34]	WinFS[12], SRB[23, 35, 36], iRODS[31]

表 3.7 Rcommands におけるメタデータ

メタデータ名	メタデータの値
Simulation metadata	情報 (実行したユーザ，実行日時，実行計算機名ほか)
Parameter metadata	シミュレーション実行時のパラメータ (温度，湿度，位置エネルギー，実行時間)
Property metadata	プロパティ値 (検索に利用する値: 最終的な結果，もしくは，平均エネルギーなど)
Code metadata	コード名，バージョンなど
Arbitrary metadata	インデックスデータ

3.3. メタデータ収集機能

本節では，メタデータ収集機能を有する関連研究について述べ，2.6 項で述べた要求事項 (2) を照らし合わせ，問題点を述べる．

ユーザは極力，自動的にメタデータの生成，収集することを求めている．このような要求に対して，様々な手法が提案されている．Typer[60] らは，表 3.7 の 5 つのメタデータを定義し，自動的に収集する Rcommands を提案している．表 3.7 のメタデータは，グリッド環境におけるジョブスケジューラにジョブを投入時に XML データとして出力される．このシステムでは，シミュレーションに対してのメタデータを取得可能である．

Beagle++[59] は，ユーザのデスクトップ PC 上でメタデータによるデータが可能なシステムである．Beagle++がメタデータの収集対象とするデータは，構造化ドキュメント，E メール，オフライン Web ページ，テキストファイルなどの一般的なファイルである．Beagle++は inotify を用いてファイルシステムイベントを取得して，メタデータを収集する．また，Goolgle の提供している Google Desktop や Apple Mac OS X の Spotlight なども，全文検索するためにデータからメタデータを抽出する．Google Desktop など全文検索するために検索用インデックスデータベースを構築するシステムでは，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 が発生する問題がある．

表 3.8 メタデータ収集の関連研究のまとめ

	Rcommands	Beagle++	Google Desktop	Spotlight
要求事項 (2)-1 収集自動化	○	○	○	○
要求事項 (2)-2 収集負荷	○	○	×	○
要求事項 (2)-3 データの鮮度に 応じたメタデータ収集	×	○	×	○

表 3.8 に，本節で述べた関連研究を 2.6 節で述べた要求事項 (2) に照らし合わせた結果を示す．Google Desktop などのデスクトップ検索ツールでは，ファイルシステムをクロールするため収集するための負荷が一時的に高くなる．ワーカーノードの負荷に応じて，クロールするタイミングをスケジューリングできればよいが，いつジョブが割り当たるかわからないため，クロールしているタイミングでジョブが投入される可能性もある．Beagle++や Spotlight では，収集負荷を小さく抑え，メタデータ更新も可能である．Rcommands では，ジョブ投入時に収集するため，負荷は小さいが，XML データが更新された場合のメタデータ更新方法については言及されていない．

多くのシステムがメタデータを自動的に収集するため，ユーザの手間を削減することを実現している．しかしながら，各システム毎に決まったメタデータしか収集できない．また，定期的にクローリングシステムでは，データの鮮度を考慮したメタデータ収集ができない．Beagle++や Spotlight では，ファイルシステムイベントをメタデータ収集エンジンにイベントの発生を通知するため，データの鮮度を考慮したメタデータ収集が可能となる．

3.4. メタデータ管理機能

本節では，既存の広域分散環境におけるメタデータ管理システムについて述べる．2.6 節で述べた要求事項 (3) について本節で述べた関連研究と照らし合わせ，問題点について述べる．メタデータの管理方法，メタデータ I/O の高速化，メタデータの複製についての関連研究を述べる．

3.4.1 メタデータ管理

本項では 2.6 節で述べた要求事項 (3)-1 に関する関連研究を述べる．

メタデータの管理方法としては，関係データベース，XML，DHT などの手法がある．多くのメタデータ管理手法は，メタデータを関係データベース (RDB) に格納する方法を採用している．一般的に，メタデータはスキーマが決定されており，メタデータのスキーマを RDB のテーブルのスキーマとして利用することが可能である．

Liao らの研究 [66] では，astro3d という天文学アプリケーション向けのメタデータ管理システムを提案している．astro3d では，メタデータを図 3.6 のように定義し，RDB を用いて管理している．astro3d のメタデータは，5 つのテーブルから構成されている．RUN TABLE は，実行時のパラメータを記録する．アプリケーションで使用するデータセットの属性は DATASET TABLE，および，PARTITION PATTERN TABLE に格納する．図 3.6 の例では，温度，圧力が属性として定義されている．astro3d のようにアプリケーション毎に，メタデータを管理することは可能である．しかし，メタデータ管理システム上で，複数のアプリケーションメタデータを定義して管理することは RDB のテーブル構造が複雑となり，メタデータ管理システムの管理者の管理コストが増加する．

また，XML ベースでのメタデータ管理手法として，異なるデータグリッドシステムの名前空間を統一的に関するためのディレクトリ管理サービス Resource Namespace Service (RNS)[52] を建部らが Open Grid Forum[15] にて提案している．RNS は，Web サービスとして提供されている．RNS は，ファイルシステムのように仮想階層名前空間を管理する．RNS では，ファイルシステムのディレ

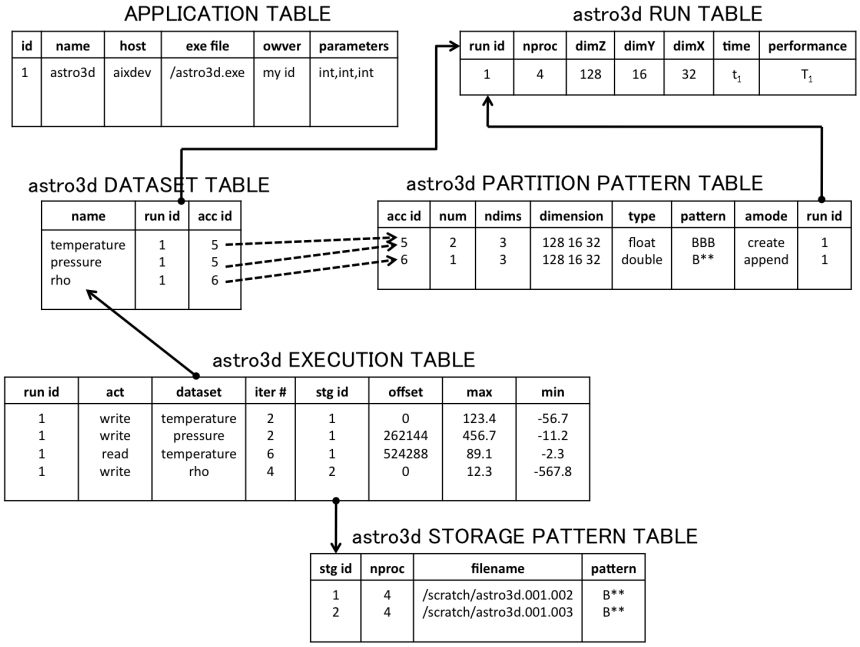

図 3.6 astro3d におけるメタデータ

クトリに相当するものを仮想ディレクトリ，ファイルに相当するものをジャンクションと呼ぶ．ジャンクションは，WS-Addressing の Endpoint Reference(EPR) によって，リソースへの参照を保存し，ファイルシステムにおけるシンボリックリンクの働きをする．異なるデータグリッドシステムの中で，RNS を用いることで，あらゆるリソースを統一的な階層型名前空間で管理することが可能である．RNS のエントリには，メタデータを自由に追加可能で，グリッドファイルシステムのメタデータ管理やバイオ，ナノアプリケーション分野のアプリケーションメタデータを管理可能である．RNS では XML ファイルにてメタデータを管理している．RNS では，異なるデータグリッドシステムの名前空間を統合可能である．しかしながら，RNS のジャンクションは，データの位置を管理するデータカタログ機能のみを提供している．また，RNS のエントリにメタデータを追加可能であるが，XML 形式のメタデータに限られている．RNS では，メタデータを検索するための手段が提供されていないため，要求事項を満たすことはできない．

3.4.2 メタデータ I/O 高速化のためのメタデータ管理

本項は，2.6 節で述べた要求事項 (3)-2 の関連研究について述べる．

メタデータ I/O は，2.5.5 節で述べたように，アプリケーション性能に大きな影響を与える可能性がある．この問題を解決するために，メタデータ I/O を高速化するメタデータ管理手法が研究されている．

Xing ら [48] の研究は，クラスタファイルシステムにおけるメタデータ I/O を高速化するためにメタデータをメモリ上で管理する手法を提案している．メタデータをメモリ上に配置することで，高速なメタデータ I/O が可能となる．

Weil らの研究 [61][64] は，サブツリーパーティショニング方式を提案している．ディレクトリの木の枝ごとにひとつのメタデータサーバで管理する手法である．サブツリーパーティショニング方式の概要を図 3.7 に示す．サブツリーパーティショニングは，それぞれのサブツリーがひとつのサーバに割り当てられるため，一貫性を確保することが容易である．それぞれのディレクトリにおけるサブツリーを管理するメタデータサーバを分割することで，メタデータ管理サーバの負荷を抑え，メタデータ I/O の高速化を図る．サブツリー方式では，ディレクトリツリーが大きくなった場合，分割してサーバ台数を増加することで，スケールアウト可能である．しかし，ひとつのディレクトリにアクセスが集中した場合は，そのサブツリーに対するメタデータ I/O は遅くなる．サブツリーパーティショニング方式は，ひとつのサブツリーをひとつのメタデータ管理サーバで管理するため，複数拠点からの利用は考慮されていない．集中型のメタデータ管理サーバでは，ユーザ側からのアクセスはネットワーク遅延が大きくなり，メタデータ I/O が遅くなり，ファイル I/O スループットが低下する．静的サブツリーパーティショニング方式は，あらかじめどのサブツリーをどのサーバが担当するかを割り当てる方式である．サーバとクライアントで共通のサブツリー割り当てマップを保持しておけば，直接クライアント側からサーバへデータアクセス要求を送信することができる．しかし，事前にデータアクセスがひとつのサーバに集中しないようにサブツリーの割り当てることは非常に難しい．動的サブツリーパーティショニング方式は，どのサブツリーをどのサーバに割り当てるかを実行中に変更可能な手法である．各サーバへのアクセスを均等に割り当てることができ，メタデータ

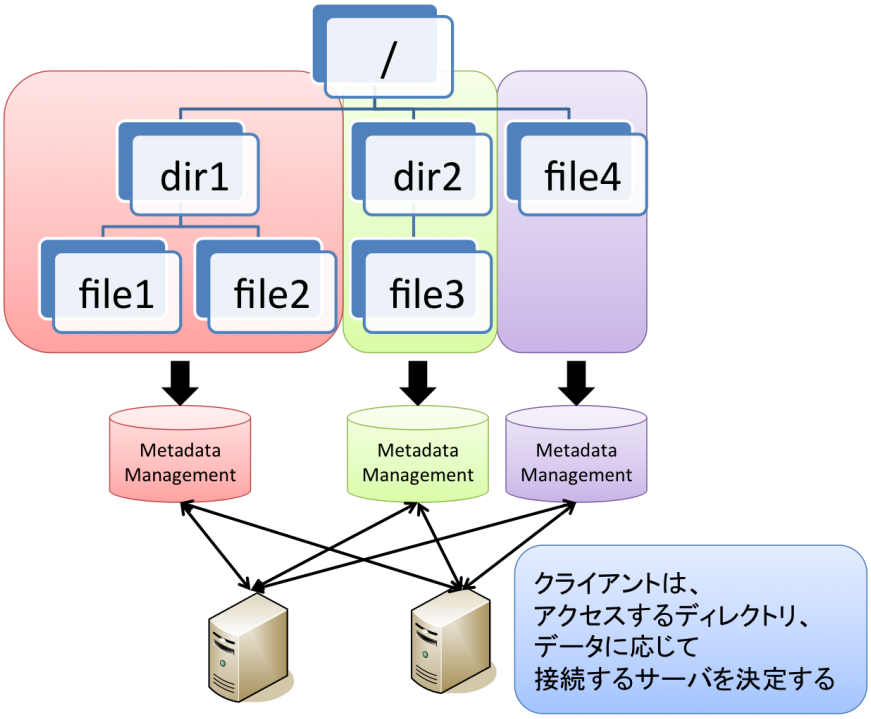

図 3.7 サブツリーパーティショニング方式

I/O がどのサーバに問い合わせても一定となる．動的サブツリーパーティショニング方式では，実行中にサブツリーとサーバの割り当てが変更されるため，クライアント側へサブツリー割り当てマップを通知する必要がある．

メタデータを分散ハッシュテーブル (DHT) にて管理すること研究が行われている．Chang-Kuo らの研究 [37] では，DHT ベースのメタデータサーバクラスタ (DHT-MDSC) を提供している．DHT-MDSC では，メタデータをバリューとして扱う．メタデータは，ファイル名，i-node，ファイルサイズ，ファイルインデックスを保持する．ハッシュキーは，ファイル名から生成する．また，Verma らの研究 [33] では，Ring File System (RFS) を実装し，クラウドコンピューティングアプリケーション向けのメタデータ管理を DHT にて行っている．RFS では，Google File System (GFS) のように，データを分割して管理する．クライアントは，DHT 上のメタデータサーバにデータを問合せ，データがどの chunkserver に

存在するかを得て，chunkserver にアクセスする．メタデータサーバは，MAC アドレスにて識別する．DHT では，ノードが増加するほどデータは分散し，各ノードの負荷は小さくなる．しかし，DHT では特定のデータにアクセスが集中すると，データアクセスの偏りが生じ，メタデータ I/O が遅くなる可能性がある．

3.4.3 メタデータ複製における一貫性制御

本項では，2.6 節で述べた要求事項 (3)-3 に関する関連研究を述べる．

分散ファイルシステムでは，ファイルカタログなどのメタデータを保持し，管理する．分散ファイルシステムの性能向上において，I/O バンド幅を大きくし，スループットを向上させることの他に，メタデータ操作のための処理時間，応答時間を短縮することが重要となる．多くのデータグリッドシステムでは，データを複製するが，メタデータの複製，分散管理については言及されないことが多い．

広域分散環境のように，高遅延ネットワークを含む環境では，メタデータ操作の度に高遅延ネットワークを経由したデータ要求が発生する．そのため，データ要求に対する応答時間が増加する．この問題に対処するため Gfarm では，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実装が進められている [76]．この研究の中で，RTT の長い通信路に依存しない Key-Value の連想配列をベースとしたメタデータを管理する手法が提案されている．メタデータサーバ間の通信には，バックグラウンドで非同期に行い，データの一貫性は結果整合性 (Eventual Consistency) にて保証している．結果整合性では一定の時間が経過し，最終的にすべての情報が同期され，データの一貫性を保証する．

パラメータサーベイアプリケーションが取り扱うデータは，ジョブ実行後の後処理 (統計解析や可視化) において，全ての結果データが必要であるわけではない．そのため，問い合わせるメタデータ管理サーバによって得られるデータ検索結果が異なってもよい．そのため，メタデータ複製における一貫性制御は，結果整合性や，弱い一貫性で充分である．

3.4.4 データの鮮度を考慮したメタデータ管理

前節で述べたGfarmの結果整合性による複製処理方法では，各メタデータ管理システムでメタデータが利用可能になるまでに時間を要する問題がある．検索結果は問い合わせるメタデータ管理システムによって異なってもよいが，データの鮮度を考慮した場合，メタデータが利用可能な状態になるまでの時間が，ワークフローにおいて，必要なデータが揃いジョブを実行するまでの待ち時間に依存する．この待ち時間は，短いシミュレーションで10秒程度である．この10秒程度の時間で，全てのメタデータ管理システムでメタデータが利用可能になっている必要があるため，結果整合性では全てのメタデータ管理システムで利用可能になるまでに時間がかかりすぎる問題がある．

3.4.5 メタデータ管理における問題点のまとめ

1. 要求事項(3)-1: メタデータ管理
   既存のメタデータ管理手法では，RDBやDHTなどを用いても，複数の異なるメタデータスキーマを持つメタデータを統一的に管理することは難しい．また，既存のメタデータ管理手法では，アプリケーションやユーザがメタデータのスキーマを意識しなければならない．

2. 要求事項(3)-2: 高速なメタデータI/O
   アプリケーション性能，とくにファイルI/Oスループットへ影響を与えないメタデータI/Oを実現するためには，メタデータサーバを分割して管理する手法が有効である．しかしながら，特定のメタデータがひとつのメタデータサーバにしか存在しない場合，アクセスが集中するとメタデータ管理サーバの負荷が高くなり，メタデータI/Oへ大きく影響を与える．また，目的のメタデータがリモートのメタデータ管理サーバに存在した場合，ネットワークの遅延がメタデータI/Oへ直接的に影響を与える．

3. 要求事項(3)-3: メタデータ複製における一貫性制御
   強い一貫性を採用する場合には，即座に全てのメタデータ管理サーバでメ

タデータの同期が可能となる．これはデータの鮮度を考慮したメタデータ管理を実現することになる．しかし，広域分散環境で強い一貫性を全てのノードで実現することは困難である．パラメータサーベイアプリケーションでは，統計処理などの後処理において全てのデータを必要としない．そのため，メタデータの一貫性制御は弱い一貫性で充分である．

4. 要求事項 (3)-4: データ鮮度を考慮したメタデータ管理
   Gfarm での組織間のメタデータ複製処理は，結果整合性を採用しており，最終的にメタデータサーバ間でメタデータの同期が取れるまで時間がかかる．このため，データの鮮度を考慮したメタデータ複製は行うことができない．

3.5. 関連研究の問題点のまとめ

本節では，前節までで述べた関連研究を問題点をまとめ，解決すべき点について述べる．

1. 広域でデータアクセス可能な POSIX ファイルシステム API と互換性のあるメタデータによるデータ検索機能
   この機能は，3.2.1 項で述べた SFS などが満たしているが，関連研究はローカルシステム上での動作するため，今回の目的である広域分散環境での動作は保証していない．メタデータを用いたデータアクセス基盤では，広域分散環境で動作する POSIX ファイルシステムと互換性を有するメタデータによるファイル検索機能の実現が必要となる．

2. ワーカーノードの I/O 性能に与える影響が小さなメタデータ収集機能
   Beagle++や，Spotlight のように必要に応じてメタデータを取得し，アプリケーションに応じたメタデータの取得が可能な機能の実現が必要となる．

3. 異なるアプリケーションのメタデータを統一的に管理する機能
   関連研究では，異なるアプリケーションのメタデータを統一的に管理する機能を有するシステムはない．アプリケーションやユーザ，管理者が，ア

プリケーションのメタデータスキーマを意識することなく利用可能なメタデータ管理の実現が必要となる．

4. データの鮮度を考慮したメタデータ収集・管理
   パラメータサーベイアプリケーションでは，データの鮮度が必要となるケースが多い．そのため，データの鮮度を考慮したメタデータの収集・管理機能が必要となる．広域分散環境でメタデータの複製を行う場合，結果整合性を用いるシステムが多い．しかしながら，パラメータサーベイアプリケーションでのデータの鮮度を考慮すると，結果整合性では不十分である．弱い一貫性を保証しながら，データの鮮度を考慮したメタデータ複製機能の実現が必要となる．このメタデータ複製機能を実現することで，ユーザ側から見た際に高速なメタデータ I/O も実現可能となる．

4，5 章にてこれらの問題を解決するシステムについて提案する．

第4章　MetaFa: 広域分散環境におけるメタデータ収集・管理システム

本章では，3.5 節で述べた，I/O パフォーマンスへの影響を低く抑えることが可能なメタデータ収集機能，アプリケーションがもつメタデータスキーマの差異を吸収してメタデータを統一的に管理する機能，データの鮮度を考慮したメタデータ収集・管理機能を有する広域分散環境におけるメタデータ収集・管理システム MetaFa を提案する．

4.1. MetaFa の概要

MetaFa (Metadata administration Factory) は，広域分散環境でメタデータを収集・管理するためシステムである．想定するグリッド実行環境を図 4.1 に示す．MetaFa は，グリッド環境で実行するパラメータサーベイアプリケーションを対象とする．ユーザは，図 4.1 のグリッド環境上で，パラメータサーベイアプリケーションを実行する．ユーザは，ジョブをマスタージョブスケジューラへ投入する．マスタージョブスケジューラは，VO (Virtual Organization) 上のワーカーノードへジョブを配布する．VO とは，ユーザの目的を実現するために必要な資源 (ネットワークやユーザを含む) から構成されるグループを意味する．ワーカーノードはジョブを実行し，シミュレーション結果をワーカーノード上のファイルシステムへ保存する．

パラメータサーベイアプリケーションの取り扱うデータは，WORM (Write Once Read Many) 型のデータであることが多く，一度書き出されたデータは，後にユー

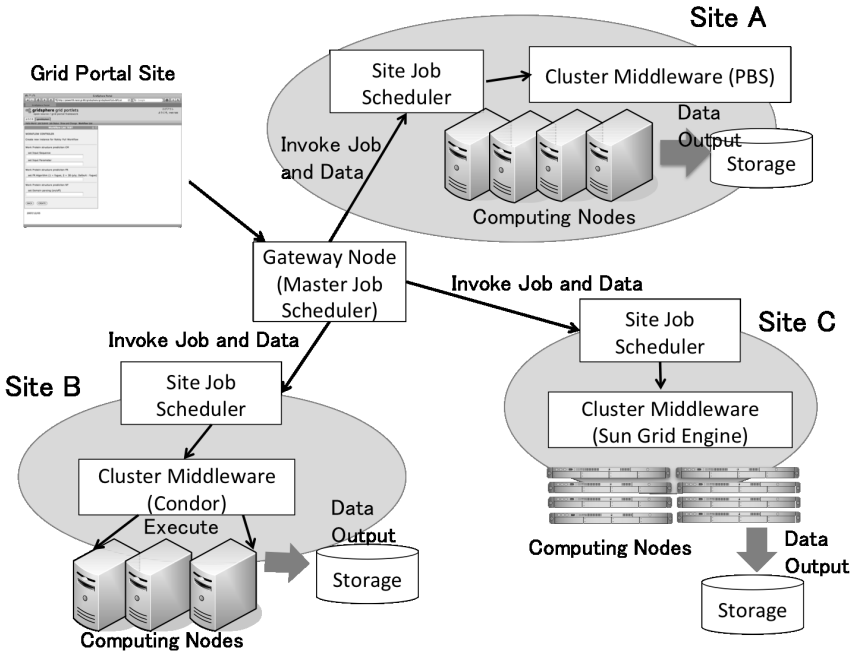

図 4.1 グリッドコンピューティングアプリケーション実行環境

ザやアプリケーションから参照のみされ，データの更新は発生しない．MetaFa は，WORM 型のデータを前提としてシステムを設計した．

4.2. MetaFa におけるメタデータの定義

本節では，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取り扱う．基本メタデータでは，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．パラメータサーベイアプリケーションで取り扱うデータの多くはファイル形式であるため，データを管理するために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有する該当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 4.1 に示す．基本メタデータの要素は，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．アプリケーションメタデータは，ユーザが定義したデータについてのアプリケーションレベルの特徴を表現する．画像データや文章データなどの一般的なデータ形式の場合は，アプリケーションメタデータのスキーマが決定している．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．一方，グリッド環境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決定したメタデータのスキーマを利用する．本研究においては，アプリケーションメタデータのスキーマは定義せず，ユーザが定義しているアプリケーションメタデータのスキーマを利用する．

MetaFa では，データ管理に必要な基本メタデータと，アプリケーションに特有な属性を管理するためのアプリケーションメタデータを扱う．MetaFa では，基本メタデータで取り扱うデータをファイルに限定し，ファイルシステムに関連する機能を利用することで次節以降で述べる自動的なメタデータ収集，データ鮮度を考慮したメタデータ収集を実現している．アプリケーションメタデータに関しては，MetaFa において属性と属性の型，メタデータ収集プログラムが存在するだけで利用可能である．SRB や iRODS においては，ユーザレベルメタデータが MetaFa におけるアプリケーションメタデータにあたる．このユーザレベルメタデータは，ユーザが定義したメタデータ (XML 形式)，メタデータ収集プログラムが必要である．SRB や iRODS，MetaFa でのメタデータの表現は同等の機能を有する．途中で，アプリケーションメタデータの属性を追加・削除しようとした場合，多くのメタデータ管理システムは，データベースのテーブルの再定義や XML の再定義など必要となる．MetaFa は，次節以降で説明するメタデータのスキーマレス管理手法を採用しているため，特にメタデータ管理データベースのテーブルの再定義は不要である．

表 4.1 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
protocol	データアクセスプロトコル

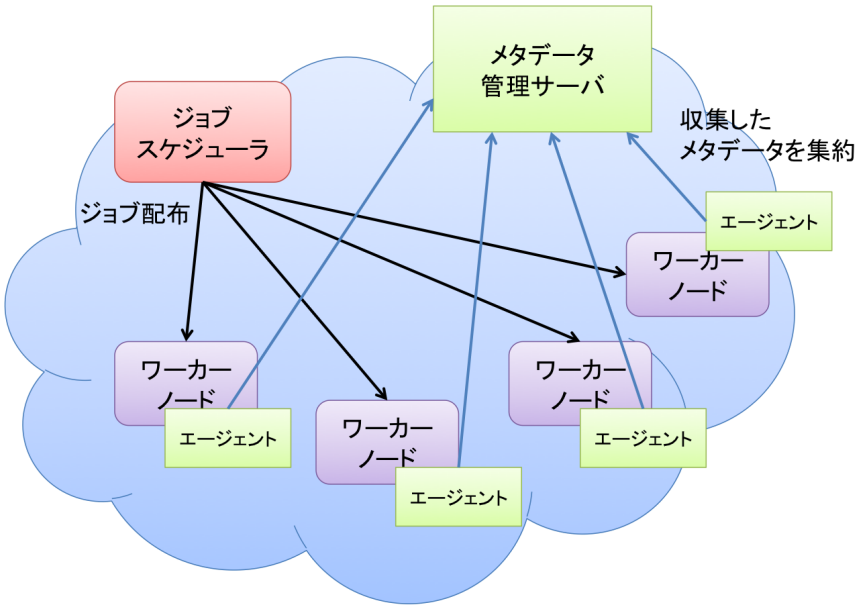

図 4.2 MetaFa システムアーキテクチャ

4.3. MetaFa の設計方針

本節では，広域分散環境でのメタデータ収集・管理における要求要件や関連研究の問題点を踏まえた MetaFa の設計方針を述べる．

MetaFa システムのアーキテクチャを図 4.2 に示す．メタデータを取得するエージェントは，計算処理を実行するワーカーノード上に配置する．ワーカーノードのようにデータを保持するノード上で，メタデータを分散することは可能である [65]．しかし，ワーカーノードの計算資源を最大限に活用するため，アプリケーションプログラム以外の処理を極力実行しない方針を採用した．これはワーカーノードに対して，メタデータ収集・管理機能が負荷を与えないためである．そのため，ワーカーノードで収集したメタデータを集約して管理するメタデータ管理サーバを導入する．MetaFa は，1 つの組織において，ワーカー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，メタデータを管理するメタデータサーバから構成されるクライアント・サーバモデルを採用した．

4.3.1 メタデータ収集の自動化手法の方針

本項では，MetaFa におけるメタデータ収集の自動化手法についての方針を述べる．グリッド環境上で実行するパラメータサーベイアプリケーションが出力するデータからメタデータを自動的に収集する手法を実現するにあたり，解決すべき問題としては，以下の 2 点である．

1. ワーカーノードの I/O 性能に与える影響が小さなメタデータ収集機能

2. データの鮮度を考慮したメタデータ収集機能

メタデータを自動的に収集する既存手法として，3 種類の方法がある．1 つめは，Google Desktop ツールなどの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一致するファイルを提示するシステムである．その特徴は，全文検索用インデックスの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．また，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．しかし，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，通知されると即座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．

3 つめは，open()/close() などのファイル I/O に関わるシステムコール関数に対して，メタデータを取得する処理を加えた open()/close() などのシステムコール関数を実装し，置き換えるライブラリプリロード方法である．この方法は，ファイル生成時に即座にメタデータを取得することができるが，アプリケーションメタデータ取得処理によって，I/O スループットが遅くなる可能性がある．

表 4.2 3つのメタデータ収集方法の比較

	クローリング (Google Desktop)	ファイルシステムイベント (Spotlight)	ライブラリプリロード
(1) 負荷	定期的に発生	イベント発生時のみ	システムコール呼び出しの度
(2) 即時性	なし	あり	あり
(3) アプリの改変	不要	不要	ライブラリの読み込みのみ

3つの手法を (1) ワーカーノードに与える負荷，(2) メタデータ収集の即時性，(3) アプリケーション改変の有無という観点から比較した．表 4.2 に比較結果を示す．Spotlihgt のように，ファイルシステムイベントを取得し，イベントに応じてメタデータ取得する手法が，ワーカーノードに与える影響も小さく，アプリケーションの改変も不要である．MetaFa では，Spotlight のようにファイルシステムイベントを取得し，メタデータを収集する方法を採用した．

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．メタデータ収集機能は，グリッド環境で用いられる Linux で動作するように設計，実装する必要がある．また，ライブラリプリロード手法についても設計し，実装した．

4.3.2 広域分散環境下におけるメタデータの管理方針

本項では，1つのアーキテクチャ上で，異なる性質のメタデータを管理氏，データの鮮度を考慮したメタデータ管理，および，メタデータ I/O の高速化を実現するための3つの方針，(1) マルチテナントアーキテクチャの採用，(2) スキーマレ

スメタデータ管理，(3) メタデータ I/O 高速化のためのメタデータ複製について述べる．

管理方針 (1)–マルチテナントアーキテクチャ

1 つめのメタデータ管理方針は，複数のアプリケーションメタデータをひとつのシステムアーキテクチャを管理するために，マルチテナントアーキテクチャを採用することである．マルチテナントアーキテクチャは 3 つのレベルに分けることができる．

レベル 1 全てのアプリケーションメタデータが同一データベース，同一スキーマを共有

全アプリケーションメタデータがひとつのインフラを共有し，運用も共通化できるため，最も効率がよく，システム構築が容易なアーキテクチャである．

レベル 2 全アプリケーションメタデータが同一データベースを共有，スキーマは個別

データベース，インフラは共有することが可能であるが，スキーマ管理や運用管理，システム構築は個別に行う必要があるため手間がかかる．

レベル 3 アプリケーションメタデータ毎に個別のデータベース，個別のスキーマを利用

アプリケーションメタデータからの個別の要求に柔軟に対応可能であるが，データベースの数が増えて管理が複雑になる．また，運用の手間も増える．インフラなど共通部分が減り，インフラの利用効率が下がる．

MetaFa では，レベル 1 の全アプリケーションメタデータが同一データベース，同一スキーマを共有するマルチテナントアーキテクチャを採用し，メタデータ管理サーバ側で，アプリケーションメタデータをメタデータスキーマの個別対応を行わずに共通のデータベース，テーブルへ格納する．異なる種類のアプリケーションメタデータを共通のデータベース，テーブルで管理するためにメタデータの保存方法に工夫が必要となる．

管理方針 (2)–メタデータのスキーマレス管理

2つめのメタデータ管理方針としては，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ことである．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．メタデータのスキーマは，RDB のテーブル構造に該当する．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，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．グリッド環境では，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，実行するアプリケーションの数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．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採用した．

メタデータの記述形式についての設計方針を述べる．メタデータの記述方法は，RDF (Resource Description Framework)[18] や XML (eXtensible Markup Language) による表現や，JSON (JavaScript Object Notation)[40] のようにデータをシリアライズした表現方法がある．最近では，Protocol Buffers[17] や，MessagePack[62] のように，構造化データをバイト列にシリアライズして管理するバイナリシリアライズなどのデータ構造もある．XML/RDF と比べて，Protocol Buffers などのバイナリシリアライズデータは，よりシンプルなデータフォーマットにできる．

バイナリシリアライズ手法は，通信時のデータサイズがXMLを用いた手法より小さくできる．Procotol Buffersなどは，プログラム側から使用する場合にも，扱いやすいデータクラスを生成することが可能である．しかし，バイナリシリアライズデータ形式は，ソフトウェアコンポーネント間で通信し，処理するために，事前に送受信するデータの仕様であるIDL(Interface Definition Language)を定義しなければならない．アプリケーションメタデータは，アプリケーション毎に要素が異なるため，アプリケーション毎にデータの属性を決定し，IDLを記述し事前にワーカーノードに配布することは困難である．MetaFaでは，JSON等のシリアライズしたデータを用いてメタデータを表現する．

管理方針 (3)–メタデータI/O高速化のためのメタデータ複製方針

3つめのメタデータ管理方針として，メタデータI/O高速化を図るため，VOに存在する全てのメタデータ管理サーバに全てのメタデータを複製する．これは，ユーザがどの組織に所属したとしても，自組織のメタデータ管理サーバへ問い合わせることで，同一の結果を得られることを目的としている．これは，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強化される．グリッド環境では，VOに参加しているノードの中から，ジョブスケジューリングポリシーに従って，異なる組織のワーカーノードへジョブが割り当てられる．ユーザ側からは，どのノードへ投入されジョブが実行されているかを意識する必要はない．MetaFaは，各組織に設置されワーカーノードからメタデータを収集する．ユーザ側からは，目的とするデータがどの組織のワーカーノードやファイルサーバに存在するか確認することはできない．そのため，ユーザはどのメタデータ管理サーバへ問い合わせても利用可能なデータが得られることが重要となる．

メタデータの複製処理では，メタデータの一貫性が問題となる．RDBでデータを複製する場合，強い一貫性制御が用いられる．メタデータの複製中は，データベースがロックされ，ユーザによる問合せができない可能性がある．しかし，パラメータサーベイアプリケーションの取り扱うデータは，WORMであるため，強い一貫性制御を必要としない．MetaFaでは，メタデータ複製処理において弱

い一貫性を採用し，複製処理時にも，ユーザからの問合せに返答可能にする．

また，データ鮮度を考慮したメタデータ管理において，メタデータが利用可能な状態になるまでの時間は，パラメータサーベイアプリケーションの結果が全て揃い，後処理が開始されるまでの時間となる．パラメータサーベイアプリケーションにおいて，データ鮮度を考慮したメタデータ管理を実現するために，メタデータのインデックスを作成後に，他のメタデータ管理サーバへ収集したメタデータをプッシュ配信する方針を採用した．このような手法を採用することにより，メタデータが利用可能になる時間を短くする．

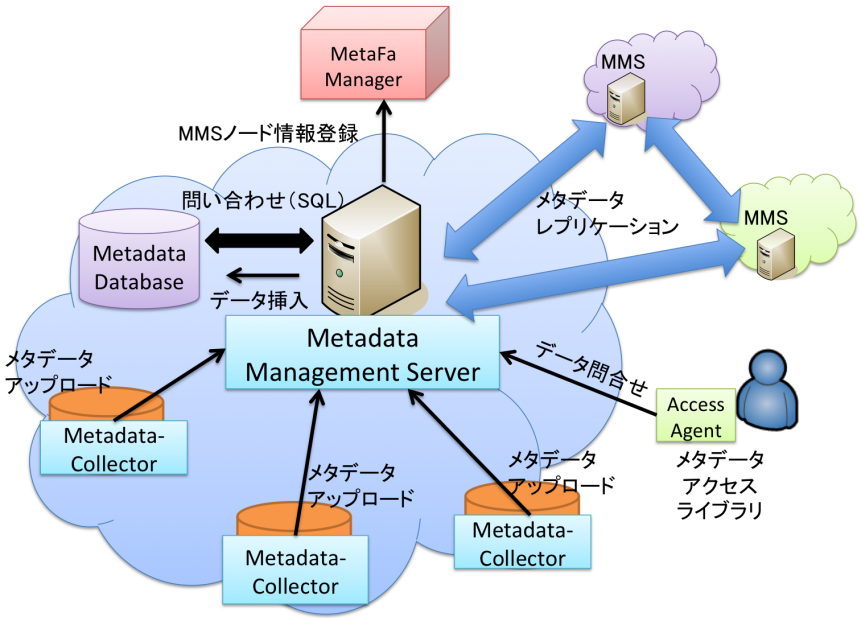

図 4.3 MetaFa コンポーネント

4.4. 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 プロトタイプシステムを Linux 上に Python を用いて実装した．

4.4.1 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，Access Agent (MetaFa-AA) から構成される．MetaFa システムの基本コンポーネントを図 4.3 に示す．次節以降にて，各コンポーネントの詳細について述べる．

4.4.2 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し，メタデータを収集する．MetaFa-MC は，デーモンとして動作し，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 でも保存する．

MetaFa-MC は，4.3.1 節で述べた設計方針に基づき実装した．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 は，ファイルシステムイベントを inotify[39] 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 がファイルシステムイベントを取得し，メタデータを収集する一連の流れを以下に示す (図 4.4).

1. アプリケーションプログラムがファイルシステムにファイル書き込み
2. Linux カーネル側の inotify が，MetaFa-MC にイベント通知
3. イベント通知メッセージは，inotify_event 構造体を含む
4. MetaFa-MC が発生したイベントに応じて，メタデータ収集処理

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可能である．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．MetaFa-MC が inotify から受信可能なイベント一覧を表 4.3 に示す．inotify インスタンスでで，inotify からどのイベントを受け取るか指定できる．MetaFa-MC では，IN_CREATE, IN_MODIFY, IN_MOVE_FROM, IN_MOVE_TO, IN_CLOSE_WRITE を受信する．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,

表 4.3 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

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．

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[21] を採用した．sqlite3 は，メモリ上にデータベースを配置可能である．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，これによって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・更新することで，ファイルシステムをクロールせ

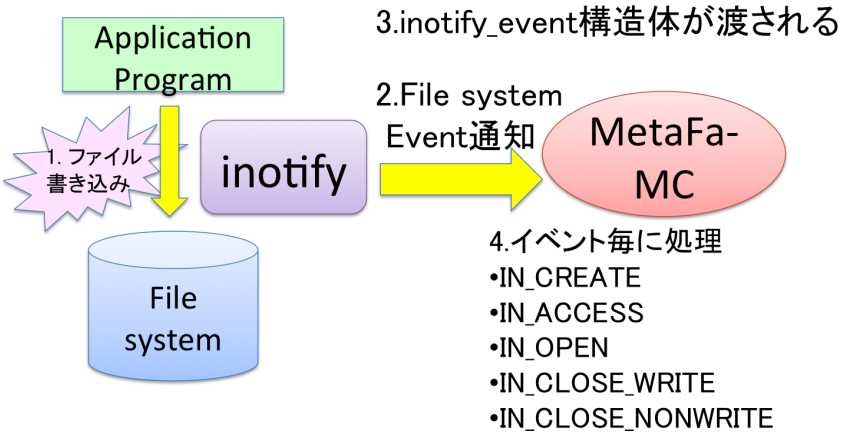

図 4.4　ワーカーノード上での MetaFa-MC の挙動

ずに，メタデータデータベースの構築・維持が可能である．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データの拡張子を確認する．MetaFa-MC は，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．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．拡張子が JPEG など一般に広く利用されている場合で，メタデータ抽出プログラムが存在するものはあらかじめ MetaFa-MC に拡張子とメタデータ抽出プログラムを登録する．また，ユーザが独自に定義し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．現在の実装では，異なるアプリケーションで出力するファイルの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．この他にも，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．

また，MetaFa-MC の別手法として，ライブラリプリロード方式でメタデータ収集するライブラリを実装した．実装した関数は，open_hook，close_hook，get_metadata，put_metadata である．これらの関数は，ライブラリ libmetafa-mc.so としてワーカーノードに配布する．アプリケーションを利用する前に，環境変数 LD_PRELOAD に libmetafa-mc.so を設定する．アプリケーション実行時に，open()/close() 関数の代わりに，open_hook()/close_hook() 関数が呼ばれる．open_hook は，ファイルのオープン時に，新規作成 (O_CREAT) フラグを検知すると，ファイルのクローズ時に get_metadata を呼び出してメタデータを収集する．しかしながら，この手法は open()/close() の度に，メタデータ収集するチェックを行うため，オーバーヘッドが大きく実際には使用しなかった．また，本ライブラリは，C 言語による実装のためアプリケーションメタデータの抽出プログラムを変更する度にライブラリのコンパイルが必要となる問題がある．

4.4.3 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 は，起動時に MetaFa-Manager 接続をする．このとき，MetaFa-MMS はホスト名と XML-RPC[28] サーバのポート番号 (デフォルト: 12000 番ポート) 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，ノードリストに掲載されている MetaFa-MMS に対して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一定回数失敗 (デフォルト: 2 回) すると，新しいアップロード先を MetaFa-Manager へ問い合わせる．このとき，MetaFa-Manager は，代替の MetaFa-MMS を MetaFa-MC へ通知する．

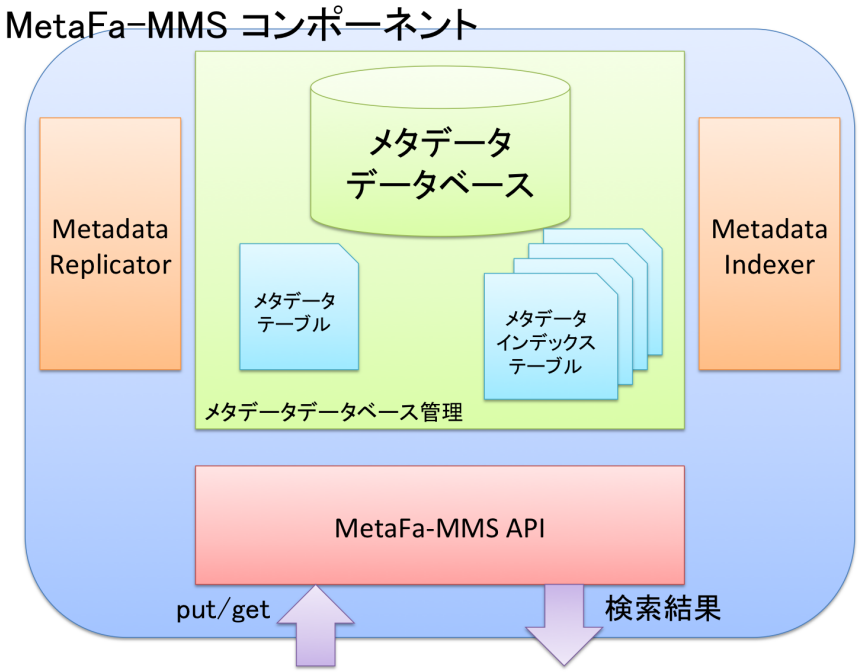

図 4.5 MetaFa-MMS コンポーネント

4.4.4 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 4.5 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，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 を提供する．MetaFa-MMS は各組織に 1 台以上設置する．

以降，RDBMS によメタデータのスキーマレス管理および，Metadata Indexer，Metadata Replicator について説明する．

(1) RDBMS によるメタデータのスキーマレス管理

RDBMS で，データを取り扱うためには，テーブルを作成するためのデータのスキーマが必要である．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取り扱う．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．RDBMS は，メタデータテーブルの 1 レコードに id，dataid，hostname，metadata_item を保持する．id は，MetaFa システムにおける主キーである．dataid は各ワーカーノード上でデータを識別するための id である．hostname は，ワーカーノードのホスト名，もしくは，IP アドレスを格納する．metadata_item に，シリアライズされたメタデータを保持する．RDBMS は，metadata_item を文字列として扱うため，データベースに格納する際，個々の要素のデータ型を意識する必要はない．MetaFa-MMS では，RDBMS に PostgreSQL を用いた．MetaFa-MMS を設置するノードにルート権限がなく，PostgreSQL がインストールできない環境なども考えられるため，SQLite を用いた実装もした．MetaFa-MMS の実行時に，データベースを PostgreSQL か SQLite か選択可能である．MetaFa-MMS では，デフォルトで PostgreSQL を使用する．

(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保持する．MetaFa では，メタデータに対して SQL による属性名と属性値を用いた検索はできない．そのため，シリアライズしたメタデータは属性名と属性値に基づいて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検索用インデックスを作成するためのプログラムである．Metadata-Indexer は，定期的に追加されたメタデータが存在しているかどうかを確認する．Metadata-Indexer は，追加されたメタデータの 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，キー毎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．

(id, dataid, hostname, {'key1':'value1', 'key2':'value2', ..., 'key$N$':'value$N$'})

図 4.6 メタデータの構造

表 4.4 メタデータインデックス

テーブル名	インデックス構造
key1	'value1': id
key2	'value2': id
...	...
key$N$	'value$N$': id

MetaFa で管理するメタデータは，図 4.6 の構造をしている．図 4.6 のメタデータから，MetaFa は表 4.4 に示す検索のためのインデックスを作成する．

(3) Metadata Replicator

MetaFa-MMS は，同一組織にある 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．MetaFa システムでは，全ての MetaFa-MMS が保持するメタデータを全て保持する．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 へ複製する．オリジナルのメタデータをもつ MetaFa-MMS は，XML-RPC プロトコルを用いてメタデータを複製する．リアルタイムにメタデータを複製するため，メタデータのインデックスを作成した後，Metadata Indexer から MMS-Replicator に対して，メタデータインデックスを作成した id の範囲を通知する．MMS-Replicator は，受信した id の範囲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する．MetaFa-MMS がメタデータを複製する流れを図 4.7 に示す．プッシュ配信にて，メタデータを複製した場合は，MetaFa-MMS は複製したメタデータをデータベース内の replica テーブルに格納する．replica テーブルに格納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．Metadata-Replicator は，Python の Multiprocessing モジュールを用いて，並列に複製処理する．

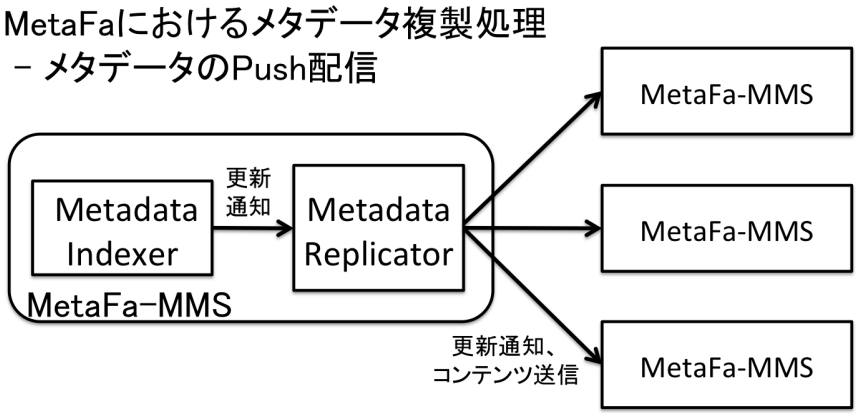

図 4.7 MetaFa-MMS のメタデータ複製機能

4.4.5 MetaFa-AA

MetaFa-AA(Access Agent) は，メタデータを用いたデータアクセスのためのライブラリである．C/C++向けの MetaFa-AA ライブラリを設計した．リスト 4.1 に，MetaFa-AA のヘッダファイルの一部，リスト 4.2 に MetaFa-AA ライブラリを用いた C プログラムの例を示す．アプリケーションプログラムは，findbymd 関数に属性名と属性値，演算子を渡すことで，該当するメタデータの一覧を取得し，データ操作可能である．

Listing 4.1 ”MetaFa-AA’s header file”

```
struct MetaFa-Data{
  char *filename;
  char *hostname;
  long inode;
  long dataid;
  struct MetaFa-Data *next;
}
// prototype declaration
struct MetaFa-Data findbymd();
```

Listing 4.2 ”Usage of API”

```
#include"metafa-aa.h"
int main(int argc, char *argv[])
{
  struct MetaFa-Data metafa;
  metafa = findbymd(argv);
  while((metafa->next) != NULL){
     fd = open(metafa->filename);
     close(fd);
     metafa->next++;
  }
}
```

4.5. MetaFaの実験・評価

MetaFa システムをキャンパスネットワーク，PlanetLab 上に展開し，実験した．MetaFa の有効性を示すため，以下の 5 つの実験を実施した．次項以降，それぞれの実験の詳細と，その結果について述べる．

実験 (1) ファイルシステムベンチマークによる MetaFa がファイル I/O スループットへ与える影響の分析

実験 (2) メタデータ収集速度の計測

実験 (3) 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

実験 (4) メタデータインデックス作成処理時間の計測

実験 (5) 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

4.5.1 実験 (1) – ファイル I/O スループットへの影響

MetaFa では，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即時に収集する．アプリケーションの評価指標のひとつは，アプリケーションのファイル I/O スループットである．本実験では 3 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ツールを用いて，MetaFa-MC の有無に応じて，書き込み処理のファイル I/O スループットを計測し，どのような影響があるかを調査した (実験 (1–1))．2 つめは，NFS 環境上で，MetaFa-MC の有無に応じて，書き込み処理のファイル I/O スループットを計測し，どのような影響があるのかを調査した (実験 (1–2))．3 つめは，アプリケーションが出力するデータからアプリケーションメタデータを取得する際のファイル I/O スループットを計測し，MetaFa-MC がファイル I/O に与える影響を調査した (実験 (1–3))．

実験 (1) では，パラメータサーベイアプリケーションが取り扱うデータサイズ，データ量において，MetaFa-MC によるメタデータ収集が，ファイル I/O 性能に対して影響を与えていないことを確認する．パラメータサーベイアプリケーションが取り扱うデータの特性は，以下の通りである．実験 (1–1)，(1–2) はデータサイズの観点，実験 (1–3) は生成するデータ数の観点から実験した．

- データサイズ: 1 KB から 1.4MB 程度
- データ数: 1,000 個 から 10,000 個程度

実験 (1–1)iozone によるファイルシステムベンチマーク

はじめに，ワーカーノード上で MetaFa-MC を動作させた場合，動作させていない場合で，iozone[8] によるファイル書き込みベンチマークを実行した．本実験は，アプリケーションが取り扱うデータのファイルサイズにおいて，ファイル書き込み処理のオーバーヘッドを調査した．

アプリケーションメタデータの収集方法は，アプリケーション毎に異なり，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．

表 4.5 ワーカーノードのスペック

ノード A	Intel Pentium 4 3.40GHz, Mem: 2GB, OS: Ubuntu 10.04 HDD: HITACHI HUA722020ALA330 2TB, 7200rpm, FS: ext4
ノード B	Intel Core i7 3.2GHz, Mem: 12GB OS:Ubuntu 9.04(x86_64) HDD: HITACHI HUA722020ALA330 2TB, 7200rpm, FS: ext4
ノード C	Intel Xeon 3.6GHz, Mem: 2GB, OS: Ubuntu 9.04 HDD: Maxtor 6L160M0 160GB, 7200rpm, FS: ext3
ノード D	Sun UltraSPARC T2 1.4GHz, Mem: 64GB, OS: Solaris 10 HDD: SAS 146GB, FC: SAS300GB, 150000rpm, FS: ZFS
ノード E	Intel Pentium 4 3.06GHz, Mem: 2GB, OS: Ubuntu 8.10 HDD SEAGATE ST3120026A 120GB U100 7200rpm, FS: ext3

そのため，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．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収集した．

キャンパスネットワーク上に設置した 2 台の性能の異なるワーカーノード上で iozone ベンチマークを実行した．iozone のパラメータは，生成したファイルサイズ，レコード長である．ファイルサイズは，64KB から 4GB の範囲で生成した．最小ファイルサイズは，iozone ベンチマークで実行可能なファイルサイズの下限である 64KB を設定した．ファイルサイズの上限は，ノード A のメモリの 2 倍としスワップが起きやすい状況にするために 4GB とした．また，レコード長は 16KB から 16MB の範囲で生成した．レコード長は，アプリケーションプログラムの作り方によって変化する．そのため，レコード長が MetaFa-MC がメタデータを収集する際に，ファイル書き込み処理に対して影響を与えるかどうかを確認した．なお，本実験時には，メタデータのアップロードはせずに，ワーカーノードの sqlite3 データベースへ蓄積した．本実験で利用したワーカーノード A，B のスペックを表 4.5 に示す．

本実験は，異なるスペックのノード A, B で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．ベンチマーク結果を図 4.8，4.9 に示す．図 4.8 はレコード長が 64KB のときの実験結果，図 4.9 はレコード長が 1024KB のときの実験結果である．グラフの横軸は，生成したファイルサイズ (KB)，縦軸は，ファイルを生成するときのスループット (MB/sec) を示す．図 4.8，4.9 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．また，MetaFa-MC が各ファイルサイズにおいてメタデータを 100%取得していることを確認した．つまり，メタデータの取りこぼしを発生させず，かつ，ファイル書き込み性能に大きな影響を与えずに MetaFa-MC ではメタデータの取得が可能であることを示した．

パラメータサーベイアプリケーションの例として，武宮らが構築した気象予報シミュレーションシステム [74] において，シミュレーションプログラムが出力するデータは，約 1.4MB である．本実験では，これよりも小さいデータサイズ 64KB，大きなデータサイズ 4GB までの範囲でファイルを書き込み，どのデータサイズにおいても MetaFa-MC はファイル書き込み I/O 性能に影響を与えていないことを示した．また，MetaFa-MC によるメタデータ収集は，ファイル書き込み時のレコード長に依存することなく，ファイル書き込み I/O 性能へ影響を与えないことを示した．

実験 (1–2) NFS 環境での iozone を用いたファイルシステムベンチマーク

2.5.2 項で述べたように，各組織でのデータグリッドの構成方法は異なる．本実験では，NFS サーバが提供する領域をワーカーノードがマウントし，NFS 上で iozone ベンチマークを実施し，MetaFa-MC がファイル書き込み I/O 性能に与える影響を調査した．iozone ベンチマークの条件は，実験 (1–1) と同じである．同時に，Gfarm で iozone にてファイルシステムベンチマークを実施した．

実験 (1–2) で用いた実験環境を図 4.10，4.11 に示す．また，実験 (1–2) で用いた各ノード間の RTT を表 4.6 に示す．

NFS サーバとして，表 4.5 のノード C，ノード D を用いた．ノード C は，/data ディレクトリをエクスポートした．ノード D では，/zfs-FC ディレクトリをエクスポートした．ノード A，B にて，ノード C，D がエクスポートした領域を NFS マ

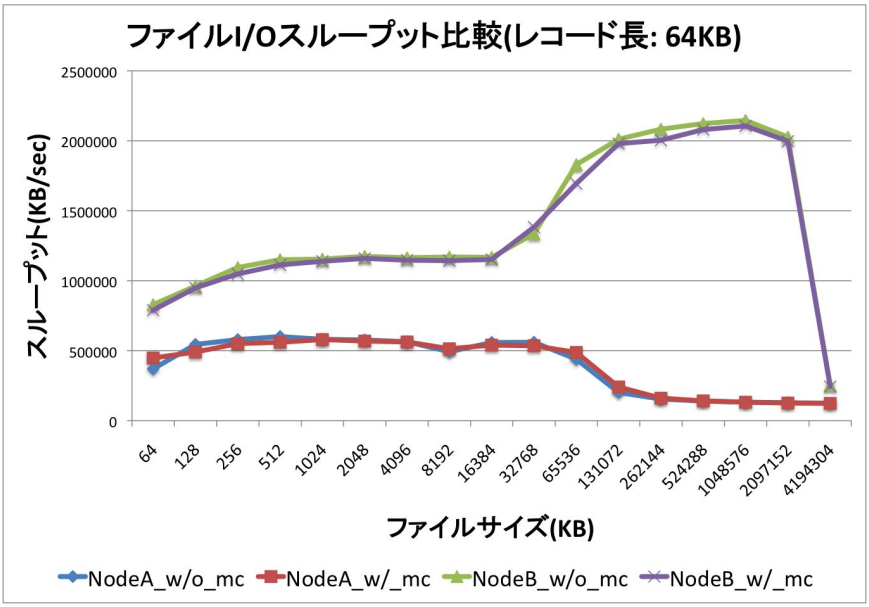

図 4.8 MetaFa の有無によるファイル書き込み性能の比較 (レコード長:64KB)

ウントして iozone で MetaFa-MC を動作させた場合と動作させない場合でベンチマークを実施した (図 4.11)．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 にて，Gfarm のメタデータサーバ (gfmd) を起動し，ノード C にて 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 A，B 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 (図 4.11)．

これらのベンチマーク結果を図 4.12，図 4.13 に示す．それぞれの図はレコード長が 64KB，1024KB の結果である．図 4.12，図 4.13 に示される値からも，大きな差は見られなかった．このことから，NFS においても MetaFa-MC が動作することを確認した．

本実験よりパラメータサーベイアプリケーションが生成するデータの書き込み先が，NFS 環境であっても，MetaFa-MC は書き込み I/O 性能へ大きな影響を与えずにメタデータ収集可能であることを示した．また，MetaFa-MC が各ファイルサイズにおいてメタデータを 100%取得していることを確認した．つまり，メ

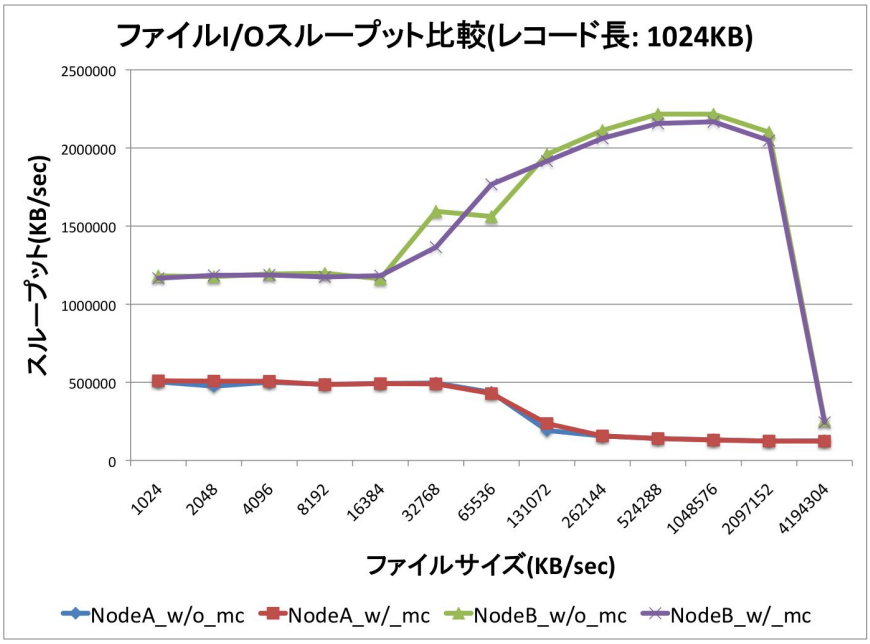

図 4.9 MetaFa の有無によるファイル書き込み性能の比較 (レコード長:1024KB)

タデータの取りこぼしを発生させず，かつ，ファイル書き込み性能に大きな影響を与えずに MetaFa-MC ではメタデータの取得が可能であることを示した．

実験 (1–3) アプリケーションメタデータ取得時のオーバーヘッド計測

次に，アプリケーションメタデータ取得時における MetaFa-MC がファイル I/O 性能に与えるオーバーヘッドを調査した．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 (EXIF Version 2.21) を取得した．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．EXIF は，JPEG データの中に埋め込まれているタイプのメタデータである．本実験は，データに埋め込まれているメタデータをプログラムを用いて取得する際に，MetaFa-MC がファイル書き込み I/O 性能へ与える影響を調査するために実施した．

実験のパラメータとして，生成するデータのデータ数を変化させた．まず，80KB

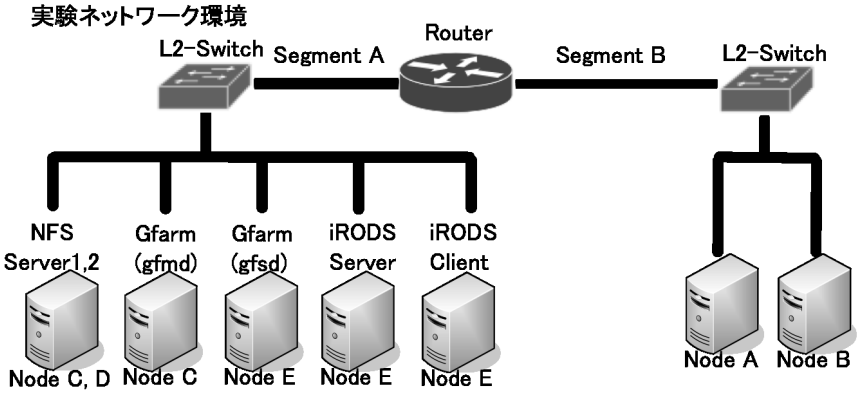

図 4.10 ファイルシステムベンチマーク実験環境 (1)

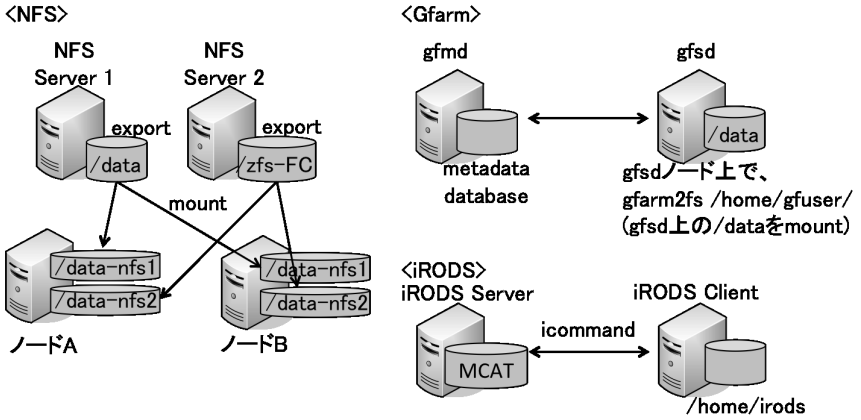

図 4.11 ファイルシステムベンチマーク実験環境 (2)

の画像ファイルを $10^N (N = 1 - 6)$ 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．本実験は，データサイズが 80KB と，比較的小さなデータを最大 10^6 個生成することで，性能劣化を引き起こしやすい状況で実験した．

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．実験環境は，表 4.5 に示すノードを用いた．NFS，Gfarm を構築した環境は，4.5.1 節で述べた実験構成 (図 4.10) と同様である．iRODS は，

表 4.6 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

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

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 4.11 に示す．本実験の結果を図 4.14 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 操作する，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 処理をする．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，スループットに差はあるものの，ノード B では MetaFa-MC を動作させない場合とほぼ変わらないファイル書き込み I/O 性能を実現した．パラメータサーベイアプリケーションの 1 例である

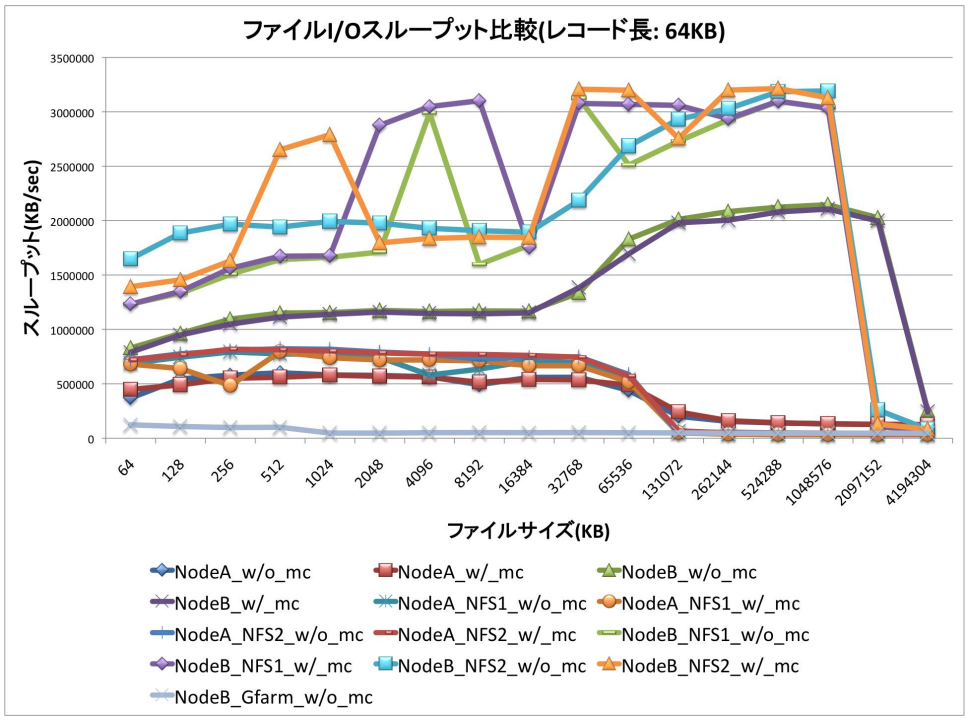

図 4.12　MetaFa と NFS，Gfarm によるファイル書き込み性能の比較 (レコード長:64KB)

気象シミュレーションの実行時間は，最短 180 秒間である．この間に生成されるデータ量はおよそ 1,000 個である．MetaFa-MC は，180 秒間に生成されるデータ量の 10 倍の 10,000 個程度のデータ量を取り扱った場合でも，ファイル I/O 性能へ大きな影響を与えず，アプリケーションメタデータを含めメタデータ収集可能であった．

実験 (1) の考察

実験 (1) で得られた結果から，MetaFa が I/O パフォーマンス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へ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ハードディスク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を確認した．これは 2

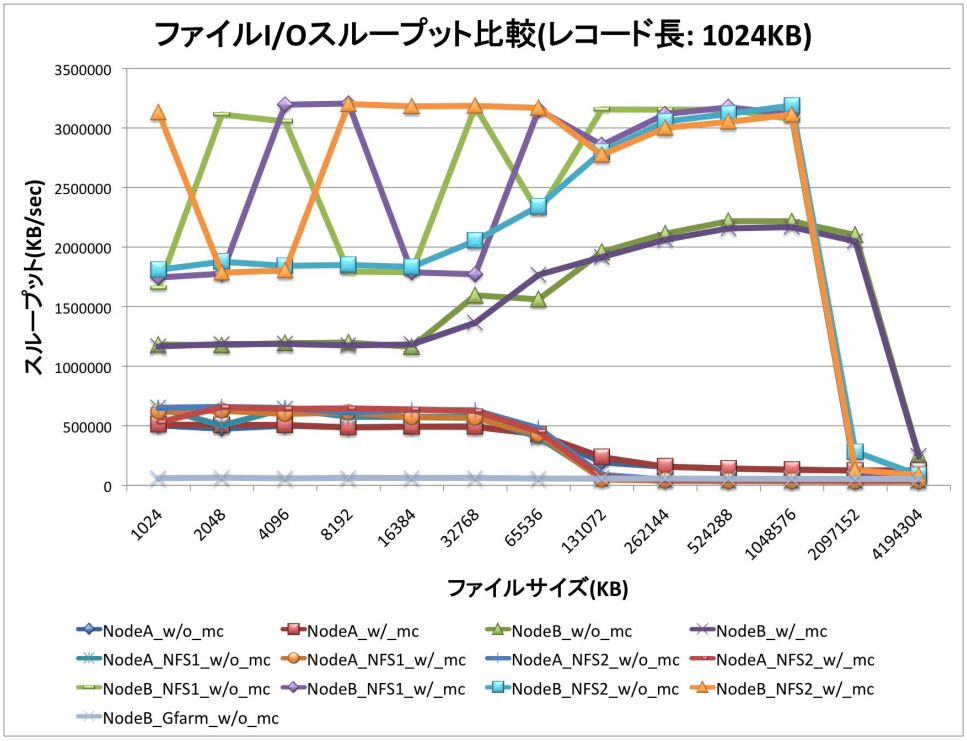

図 4.13　MetaFa と NFS，Gfarm によるファイル書き込み性能の比較 (レコード長:1024KB)

つの方針，ワーカーノード上でファイルシステムイベントを取得し，メタデータをデータ生成直後に即座に生成する方針，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する方針が有効であることを示すことができた．さらに，NFS サーバ上への書き込み処理に対しても本提案は有効であることを示した．Gfarm については，マシンの性能が異なる環境に Gfarm 環境を構築したため，直接の比較はできないが，Gfarm ではメタデータ管理サーバをファイルサーバノードとは別に設置するため，メタデータアクセスがネットワーク経由となるためボトルネックとなる可能性が高い．Gfarm と比較して，本提案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するため一定の I/O パフォーマンスを維持することができる．これは両者のメタデータの取り扱い方の違いによる．Gfarm では，メタデータ生成・操作において，POSIX ファイ

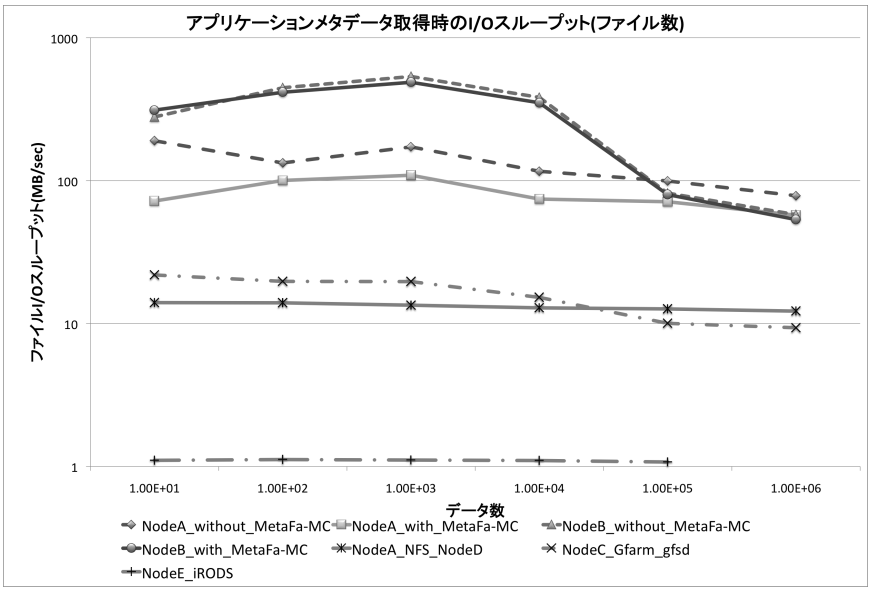

図 4.14 アプリケーションメタデータ取得時のスループットの比較 (ファイル数)

ルシステムの制約を受けるが，MetaFa システムでは POSIX ファイルシステムの制約を受けない.

4.5.2 実験 (2) – メタデータ収集に関する実験

MetaFa では，open()/close() などのシステムコールが呼ばれた際に，ファイルシステムイベントを受信し，即座にメタデータを収集する．本実験では，メタデータ収集時間を計測した．2 種類のワーカーノード上でデータを出力し，MetaFa-MC を動作させてメタデータを収集した．実験に用いたワーカーノードは表 4.5 のノード A，B である．パラメータサーベイアプリケーションでは，ワークフローと連携してジョブを実行する．ジョブを並行実行し，後処理として最終的な結果を得るために統計処理する．このとき，後処理では必要なデータを収集する．データの鮮度を考慮したメタデータ収集・管理を実現する場合，各ジョブが終了し，後処理のためにデータを転送する時間までにメタデータが利用可能になっている必要がある．気象シミュレーションの例では，1 サンプルシミュレーションの実行でデータ転送までの待ち時間が 10 秒程度である．本実験では，MetaFa-MC によるメタデータ収集において，データの鮮度を考慮できているかどうかを確認する．

本実験では，アプリケーションメタデータとして画像データから EXIF 情報を収集した．80KB の画像ファイルを，$10^N (N = 2 – 5)$ 個数分 MetaFa-MC が監視しているディレクトリにコピーし，メタデータの取得時間を 3 回計測した．実験結果を，図 4.15 に示す．図 4.15 の横軸がコピーしたデータ数，縦軸がメタデータの平均収集時間 (ミリ秒) である．ノード B の場合，データ生成後 0.04 ミリ秒でメタデータを収集している．1 シミュレーションで取り扱うデータ数が 10^5 個の場合でも，メタデータ収集の合計は 4 秒から 18 秒であった．ワーカーノードの性能に依存するが，後処理のためのデータ転送時間以内で，メタデータ収集可能である．

実験 (2) の結果からメタデータの収集では，各ワーカーノード上で inotify を用いてファイルシステムイベントを受け取り，データ発生から 10^5 個のデータに対して，トータル 4 秒でメタデータ収集可能とした．メタデータ収集部分に関しては，データの鮮度を考慮したメタデータ収集が実現できている．

しかしながら，アプリケーションメタデータの取得方法によっては，より多くのメタデータ取得時間になる．今回の実験では画像データから EXIF 情報を取得する形でメタデータを収集した．これは，画像データに EXIF 情報を付与している

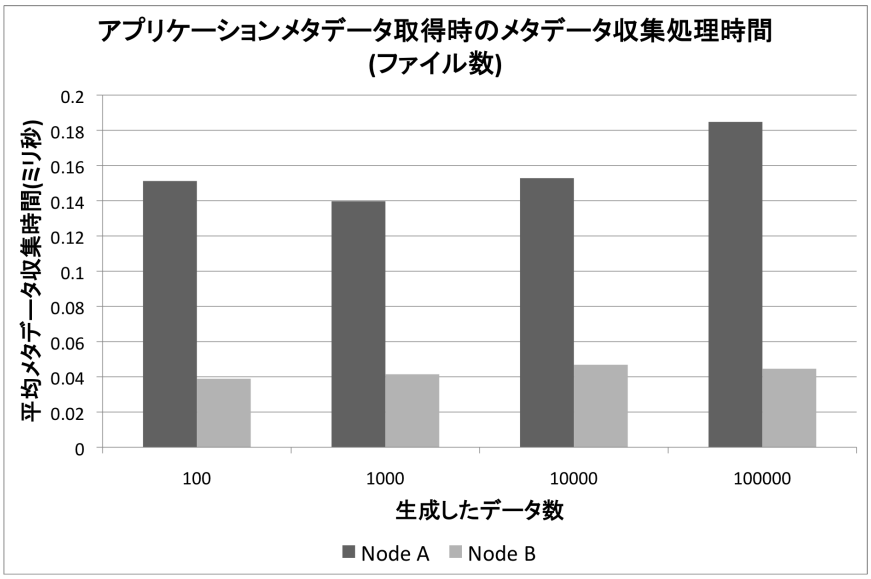

図 4.15 EXIF 情報取得時のメタデータ収集時間 (ファイル数)

領域があり，その領域を読み込んでメタデータを取得している．メタデータの種類によっては，データ全体を読み込んで取得しなければならないものも存在する．そのため，MetaFa システムでは，パラメータサーベイが取り扱う全てのデータに対して，完全にデータの鮮度を考慮したメタデータ収集をサポートしているとは言えない．これは，データからのメタデータ収集方法に依存するため，MetaFa 側でサポートすることは難しいと考える．

4.5.3 実験 (3) – スキーマレスメタデータ管理手法

本実験では，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メタデータの平均アップロード処理時間を比較した．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 へメタデータをアップロードした．本実験時には，他の組織に存在する MetaFa-MMS へメタデータを複製しなかった．MetaFa-MMS 上に，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毎にテーブルを定義して作成する．実験には，キャンパスネットワークに設置した 40 台のワーカーノードと 1 台のメタデータ管理サーバを用いた．実験に用いたワーカーノードとメタデータ管理サーバのスペックを表 4.8 に示す．キャンパスネットワークに設置した 10，20，30，40 台の MetaFa-MC からキャンパス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する．実験したノードの台数は，2.4.5 項で示した，VO における 1 サイトに存在するワーカーノードの数である．MetaFa-MMS は，各サイトに最低 1 台ずつ設置する前提である．

メタデータ管理サーバとワーカーノード間は，3 つのルータを経由し，RTT は 0.227 ミリ秒であった．本実験で扱うアプリケーションメタデータは，画像の EXIF 情報 (メタデータ属性数: 27)，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 (メタデータ属性数: 18)，Web サーバのログ (メタデータ属性数: 20) と基本メタデータのみ (メタデータ属性数: 15) 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 1000，2000，3000，4000 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．気象シミュレーションでの 1 サンプルシミュレーションの実行時間である 180 秒の間に生成されるデータ量がおよそ 1000 個程度である．1000 個のメタデータの中に，異なる 4 種類のメタデータを混ぜてメタデータのスキーマレス管理手法によるメタデータ管理の有効性を検証した．

メタデータのスキーマを管理しないデータベースのテーブル構造と，スキーマを管理するデータベースのテーブル構造を図 4.16 に示す．メタデータのスキーマ

を管理する方法では，データベース上に基本メタデータ用テーブル，アプリケーションメタデータ用に3つのテーブルと，それぞれのアプリケーションと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上でメタデータの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．そのため，MetaFa-MCで，アプリケーションメタデータのスキーマを管理する実装を行った．MetaFa-MCは，メタデータスキーマを管理するために，メモリ上に設置した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を管理するためのテーブルを用意した．

まず，MetaFa-MC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込む．本実験では，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込処理速度を計測した．メタデータデータベースからは4種類のメタデータをランダムに読み込む．それぞれの読込処理速度を表4.7に示す．スキーマを管理する手法では，4,000個のデータ読込時間は，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個のデータ読込時間は平均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込み可能である．提案手法は，データベースからのメタデータ読み込みにおいてスキーマを管理しない手法が高速に処理可能であり，ワーカーノードに与える負荷も低く抑えることが可能である．

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がメタデータデータベースから読み出した全てのデータをアップロードする時間を計測する．本実験の結果を図4.17に示す．横軸は，MetaFa-MCからアップロードするメタデータ

表 4.7 MetaFa-MC 上におけるスキーマありなしでのメタデータ読込処理速度比較

	メタデータ スキーマを管理	メタデータ スキーマを管理しない
平均読込処理時間	11.65 秒	4.64 秒

(4000 個のメタデータをランダムに読み込んだ場合の処理時間の比較)

表 4.8 ワーカーノードとメタデータ管理サーバのマシンスペック

ワーカーノード F	Intel Xeon Quad-core X5570 (2.93GHz) × 2, 24GB DDR3-1333, CentOS 5.4(x86_64), HITACHI H103030SCSUN300G SAS 300GB 10000rpm × 4
メタデータ管理サーバ	Intel Xeon L5520 (2.26GHz), 12GB DDR3-1333/PC3-10600, Ubuntu 9.04(x86_64) HITACHI HUA7250S SATA 500GB 7200rpm

のレコード数，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．塗りつぶした棒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，網掛けの棒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 4.17 より，メタデータのスキーマを管理しない提案手法がメタデータの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．

次に，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．スキーマレスでメタデータを管理することのメリットは，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，ユーザはグリッド環境において，どこのワーカーノードで処理を実行するかを事前に特定できない．そのため，ユーザが処理する前にメタデータのスキーマをワーカーノードに配布しておくことは困難である．スキーマレスでメタデータを管理することにより，MetaFa-MC

◆スキーマレスメタデータ管理におけるテーブル構成

id	dataid	item	item_num
1	23456	{'key1':'value1', 'key2':'value2'}	2
2	23589	{'key1':'value2', 'key3':'value3', 'key4':'value4'}	3
3	56789	{'key1':'value5', 'key5':'value1'}	2

◆スキーマ有りメタデータ管理におけるテーブル構成

•基本メタデータテーブル

id	dataid	...	appid
1	23456		-1
2	56789		1

•appid管理テーブル

appid	application name
-1	basic metadata only
1	protein structure prediction
2	EXIF

appid 1 table

id	parameter	start	end
2	...	...	...

appid 2 table

id	version	date	model
3	...	...	...

図 4.16 メタデータスキーマの有無に応じたテーブル構成

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．個別のメタデータスキーマを管理することは，データベースが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，データベースの動的分割機能などは，商用データベースなどに限られており，全てのMetaFaで商用データベースを利用することは困難である．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．スキーマレスでメタデータを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．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れば，処理の高速化を図れば，特に大きな問題とはならないと考える．

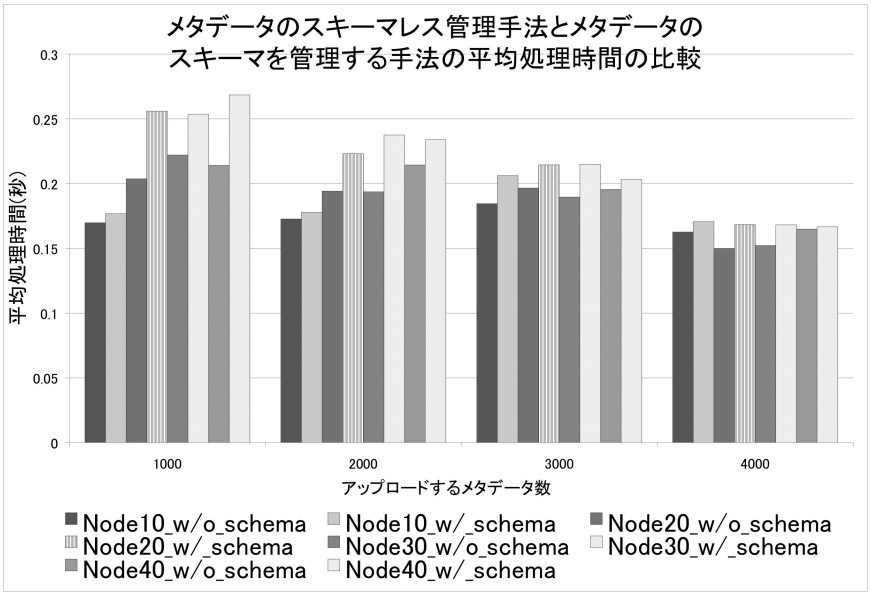

図 4.17 メタデータのスキーマレス管理とスキーマを管理するシステムの比較

実験結果より，メタデータをスキーマレスでシンプルに管理することにより，パラメータサーベイアプリケーションが 1 度に取り扱うデータの量 (1,000 個から 10,000 個程度) では，データベースに与えるオーバーヘッドはないことが判明した．ワーカーノードが 10 台程度で，気象シミュレーションで，50 サンプルシミュレーションを実行した場合，実行時間は 1,200 秒程度で，1 ワーカーノードあたり 1,000 個から 4,000 個のデータが出力された場合でも，全てのメタデータが利用可能になるまでの時間は，700 秒程度であり，シミュレーション実行時間内に利用可能となるため，データの鮮度を考慮したメタデータのアップロードが可能である．

4.5.4 実験 (4) – メタデータインデックス作成に関する実験

本実験では，MetaFa-MMS 上で検索に用いるメタデータインデックスを作成する処理時間を計測する．MetaFa-MMS が MetaFa-MC からのアップロードされるメタデータを 120 秒おきに確認し，メタデータインデックスを作成する．このと

き，インデックスを作成するメタデータのレコード数を 1,000，10,000 と変化させた．180 秒間のシミュレーションで，1,000 から 100,000 個のデータが出力された場合を想定している．また，メタデータの持つ属性の数によって，インデックス作成時間に違いがあるかを確認するためにメタデータの属性数を変化させた．本実験で用いたメタデータは，基本メタデータのみ (属性数: 15)，基本メタデータと画像の EXIF 情報 (属性数: 27)，基本メタデータとタンパク質立体構造予測のアプリケーションメタデータ (属性数: 18)，基本メタデータと Web サーバのログのアプリケーションメタデータ (属性数: 20) である．実験結果を図 4.18 に示す．図 4.18 の横軸が，処理するメタデータのレコード数，縦軸がインデックス作成処理時間 (秒) である．各棒グラフは，メタデータの持つ属性数に対してのインデックス作成時間となっている．インデックス作成時間は，データ数に比例して増加している．インデックス作成時間は，1,000 個のデータに対して，およそ 25 秒であった．180 秒のシミュレーション実行時間のうち，後処理に対してデータを転送する時間が 10 秒程度である．後処理は，10 秒程度の待ち時間が許容されるが，それに対してインデックス作成時間は 25 秒かかっている．そのため，メタデータが利用可能になるまでの時間が後処理の開始するまでの待ち時間に影響を与えている．現在の実装では，データの鮮度に対して，インデックス作成処理が MetaFa において，最大のボトルネックとなっている．

4.5.5 実験 (5) – 広域分散環境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，全ノードへ複製が完了するまでの収束時間を計測した．本実験では，広域分散環境において，データの鮮度を考慮したメタデータ複製が実現できているかどうかを検証した．本実験は，キャンパスネットワーク，PlanetLab，Dummynet を用いたエミュレーション環境で実施した．

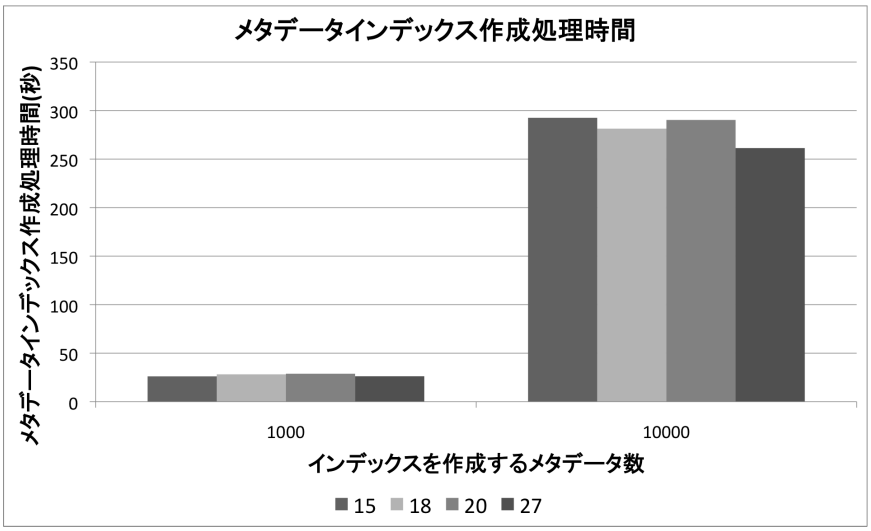

図 4.18 メタデータインデックス作成処理時間

実験 (5–1) キャンパスネットワーク上でのメタデータ複製処理実験

N 台の MetaFa-MMS が存在するとき，オリジナルのメタデータをもつ 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．本実験では，実験 (3) で用いた表 4.8 中のワーカーノード F である．メタデータ複製元ノードは，実験 (3) で用いた表 4.8 中のメタデータ管理サーバである．複製するメタデータのレコード数を $10^N (N = 2, 3, 4, 5, 6)$ 個，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．MetaFa-MMS は，各サイトに 1 台以上設置する前提であるため，複製するノード数は，VO 上のサイト数と同じとなる．最低限満たすべきノード数は，2.4.5 項で示した VO におけるサイト数 5 から 10 である．

メタデータ複製元の MetaFa-MMS とメタデータ複製先の MetaFa-MMS 間の RTT は，0.227 ミリ秒であった．このとき，実験に用いたメタデータは基本メタデータ (属性数: 15) のみである．1 台の MetaFa-MMS から 10，20，30，40 台の MetaFa-MMS に対して，メタデータをプッシュ配信する．MetaFa-MMS は，Python の Multiprocessing 機能を用いて，CPU 個分だけ複製処理を並列実行する．このとき用いた複製元ノードの CPU は 8 コアであるため，同時に 8 台の

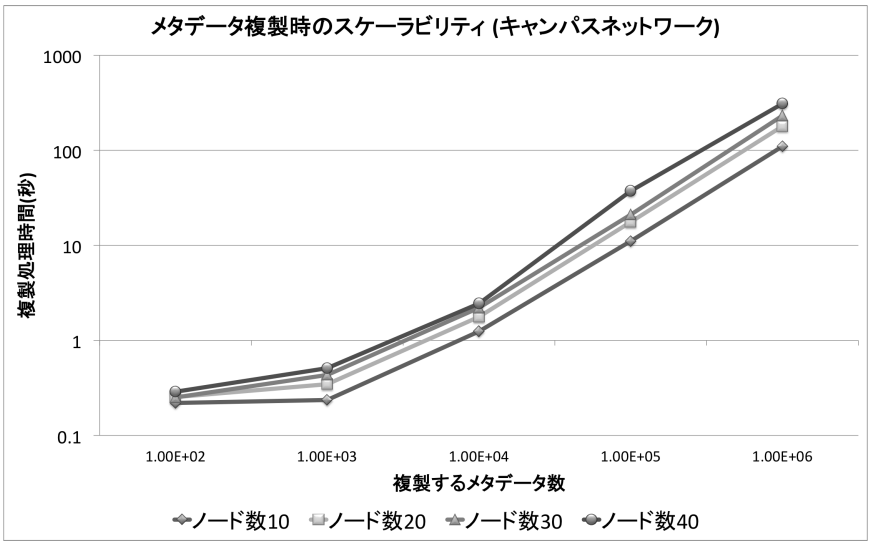

図 4.19 メタデータ複製時のスケーラビリティ(キャンパスネットワーク)

MetaFa-MMS に対して，プッシュ配信した．

実験結果を図 4.19 に示す．横軸は複製したメタデータのレコード数，縦軸は複製処理時間 (秒) を示している．縦軸は，対数表示である．図 4.19 より，実験結果より，メタデータのレコード数，ノード数の増加に伴い，複製時間が増加している．本研究で対象とするパラメータサーベイアプリケーションが取り扱うデータの数は，1,000 個から 10,000 個程度である．これらのデータ量に対するメタデータを，どのノード数においても 1,000 個の時 1 秒以内，10,000 個の時 5 秒程度で複製処理を完了している．気象シミュレーションの最短のシミュレーション実行時間は，180 秒であり，その内，後処理のためのデータ転送待ち時間が 10 秒程度ある．そのデータ転送待ち時間内に，1,000 個のメタデータが複製処理によって，全ての MetaFa-MMS にて利用可能になる時間はおよそ 1 秒である．データ転送待ち時間の 10 秒程度に充分収まる時間内に 1,000 個のメタデータを最大 40 台の MetaFa-MMS へ複製可能である．

表 4.9 PlanetLab ノードのスペック要件

CPU	Intel Cores @ 2.4GHz × 4
RAM	4GB
Disk	500GB

PlanetLab テストベッド上でのメタデータ複製処理実験

組織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した．PlanetLab[16] とは，プリンストン大学を中心としたプロジェクトで，次世代のインターネット，その上で動作する分散アプリケーションを研究するため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毎に，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，日本国内の最大 20 台の PlanetLab ノードに対してメタデータを複製した．

実際の VO では，サイト間でのメタデータが複製処理される．組織間は，インターネットを経由して接続されているため，メタデータ複製処理において，ネットワーク遅延が影響する可能性がある．本実験では，インターネット上に配置された PlanetLab ノードを用いて，メタデータ複製処理に対してネットワーク遅延が与える影響を調査した．

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 5，10，15，20 台を選択した．PlanetLab がノードに求めるがスペックの目安を表 4.9 に示す．オリジナルのメタデータをもつ MetaFa-MMS から PlanetLab ノードに対する平均 RTT を表 4.10 に示す．先の実験と同様に，複製するメタデータのレコード数を 1,000 個から 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，1GB 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ためである．

[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．

表 4.10 PlanetLab ノードへの平均 RTT

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

実験結果を，図 4.20 に示す．横軸は複製するメタデータのレコード数，縦軸は複製処理時間 (秒) を示している．縦軸は，対数表示である．キャンパスネットワーク上での実験結果と同様に，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 を用いた本実験の結果は，1,000 個のメタデータ，10,000 個のメタデータの複製処理がともに 5 秒程度の複製処理時間であった．しかし，図 4.20 の中で，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．実験時の 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表 4.11 に示す．5 分間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 ノードの方が負荷が高かった．多くのユーザが利用しているため PlanetLab ノードは常時，負荷が高い．20 台で実験を実施した際の PlanetLab ノードのロードアベレージを表 4.11 に示す．5 分間のロードアベレージの平均は，8.52 であった．5 分間の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 上で実験を実施したが，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影響を与えているとは判断できない．

Dummynet によるインターネットエミュレーションした環境におけるメタデータ複製処理実験

負荷が低い PlanetLab ノードを利用することは難しい．本実験では，キャンパスネットワークに設置したノードの間に，Dummynet[2] を設置し，擬似的にイン

表 4.11 PlanetLab ノードのロードアベレージ

5 台での実験時のロードアベレージ (平均)	10.21
10 台での実験時のロードアベレージ (平均)	6.70
20 台での実験時のロードアベレージ (平均)	8.52
20 台での実験時のロードアベレージ (最大)	29.64
20 台での実験時のロードアベレージ (最小)	0.41

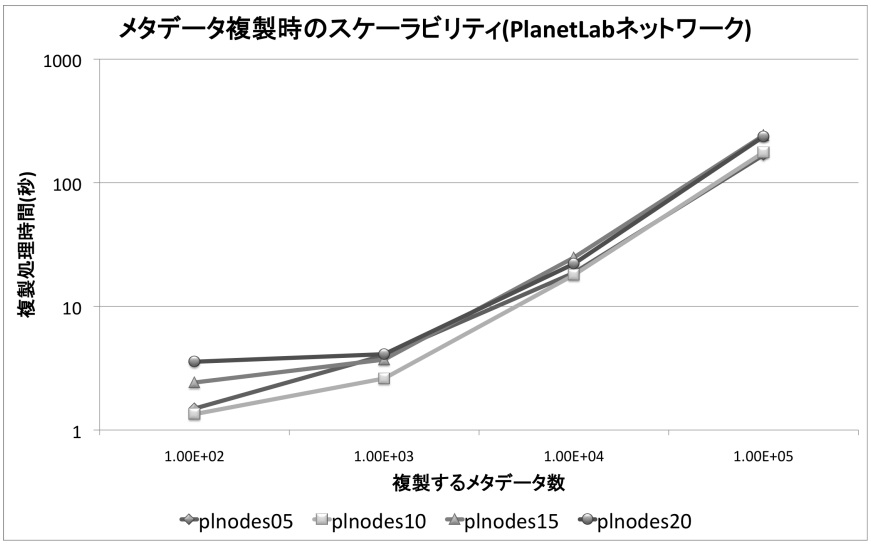

図 4.20 メタデータ複製時のスケーラビリティ (PlanetLab ネットワーク)

ターネットを用いた実験環境を構築した．Dummynet とは，FreeBSD 上で動作し，帯域，遅延時間，パケット損失率などを制御可能なツールである．この実験では，負荷が低いノード間で，RTT を増加させた場合でのメタデータの複製処理を実施した．本実験は，ネットワークの遅延時間がメタデータ複製処理時間に与える影響を調べるために，RTT を増加させた．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．本実験の構成を図 4.21 に示す．メタデータの複製元の MetaFa-MMS から 8 台の MetaFa-MMS の間に，Dummynet を設置した．Dummynet は，2 つの NIC をブリッジ接続として設定した．Dummynet では，ipfw コマンドにて em0 から em1，em1 から em0 を通過するパケットに対し

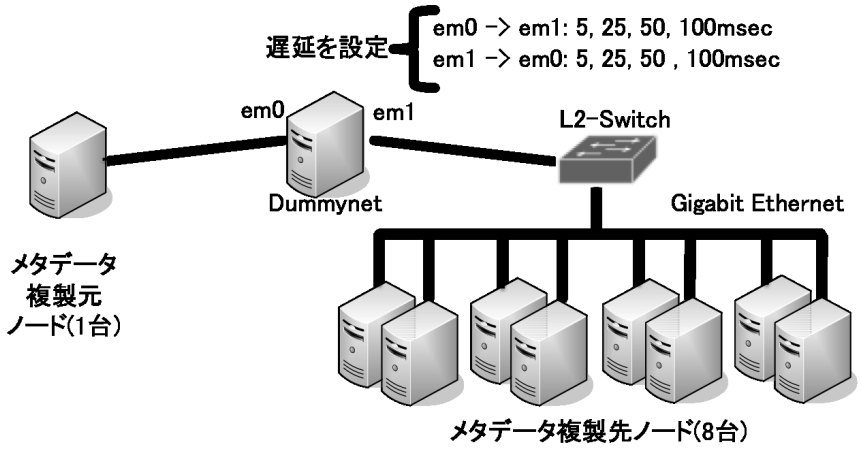

図 4.21 Dummynet を用いた実験構成

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．このように設定することで，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．RTT は，それぞれ東京–大阪間，北海道–沖縄間，東京–東南アジア間，東京–米国西海岸間でのメタデータ複製を想定して設定した．複製するメタデータの数は，$10^N (N = 2, 3, 4, 5, 6)$ 個とした．

実験の結果を，図 4.22 に示す．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，ネットワーク遅延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，ネットワーク遅延の影響をほとんど受けていない．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，ディスク I/O がボトルネックとなり，ネットワーク遅延の影響をほとんど受けていないと考えられる．メタデータが 1,000 個から 10,000 個で，40 台のノードへ複製処理した際に，およそ 4 秒から 8 秒の複製処理時間である．データ転送待ち時間が 10 秒であった場合，後処理の開始に対して多少影響は起きる．ユーザ側からすると，メタデータによってデータ検索したタイミングによっては，多少のデータの取りこぼしが発生する可能性もある．

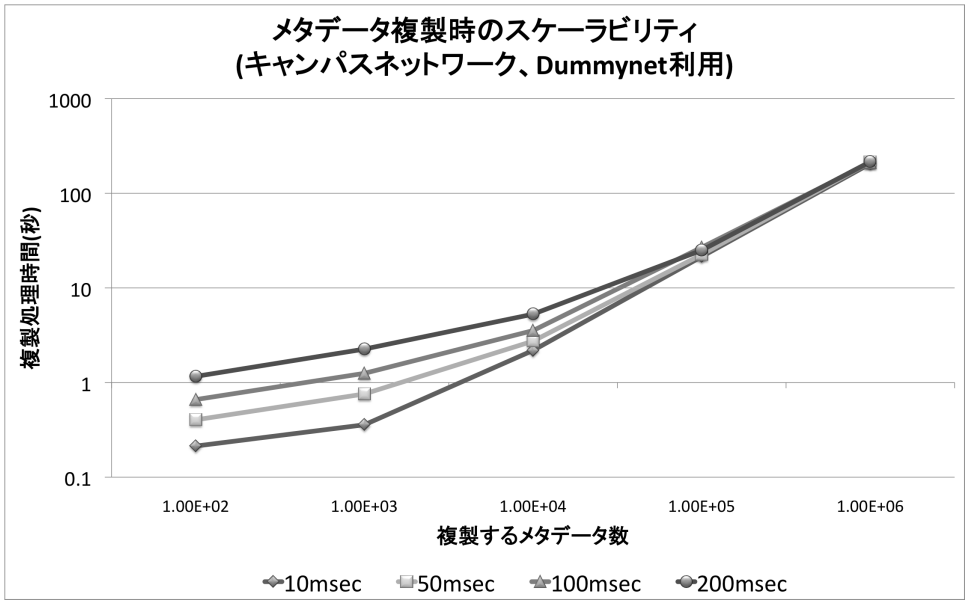

図 4.22 メタデータ複製時のスケーラビリティ(Dummynet)

4.6. MetaFaの考察

本節では，MetaFaのデータ鮮度を考慮したメタデータ収集・管理について考察する．気象シミュレーションの1サンプルシミュレーションにおいて，1,000ファイル生成された場合，シミュレーション時間180秒のうち，データ転送待ちは10秒程度である．このとき，メタデータが利用可能になるまでのステップおよび各処理時間は，以下の通りである．

1. メタデータ収集 (0.4秒)
2. メタデータアップロード (15秒)
3. メタデータのインデックス作成 (25秒)
4. メタデータの複製処理 (1秒)

データの鮮度を考慮したメタデータ収集・管理に対して，メタデータアップロードとメタデータのインデックス作成処理がボトルネックとなっている．メタデータアップロード部分に関しては，1つずつ送信していたが，オーバーヘッドが大きいため，まとめてアップロードするように実装を変更した．アップロード処理時間は1,000個のメタデータであれば，1秒程度でアップロード可能となっている．残るボトルネックは，インデックス作成処理である．この部分が短縮できれば，データ転送待ち時間の10秒以内にメタデータが利用可能になるまでの時間を抑えることが可能となる．

MetaFaをグリッド環境で運用した場合の考察する．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，全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．

4.7. 広域分散環境におけるメタデータ収集・管理システムのまとめ

本章では，以下の特徴を持つ広域分散環境におけるメタデータ収集・管理システムMetaFaを提案した．

- ワーカーノードのファイルI/Oへの負荷を抑え，アプリケーションの実行性能へ影響を与えないメタデータ収集
- 異なるアプリケーションメタデータスキーマを統一的に管理するメタデータ管理手法
- メタデータI/O高速化のためのメタデータ複製配置
- データアクセスインタフェースに対するデータ検索のためのメタデータ提供

MetaFaにおける課題としては，メタデータのスキーマレス管理をRDBMSではなくドキュメント指向DBであるMongoDB[11]にて実装し，既存の実装方法と比較する．MongoDBは，JSONをベースとしたBSON(Binary-encoded serialization of JSON-like documents)を用いて，スキーマレスなデータモデルにてデータを管理する．MongoDBにはカラムという概念が存在せず，データのスキーマを定義せずに，柔軟にデータ管理できる．また，B-treeインデックスをサポートした高

速なクエリ，Master-Slave Replication に加え，Replica Pairs によるフェイルオーバー構成を選ぶことができる．

2 つめの課題としては，MetaFa-MC で収集したメタデータを MetaFa-MMS へのアップロード戦略である．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 秒毎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 秒毎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ー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．

3 つめの課題としては，MetaFa の性能向上を図る必要がある．2.4.5 項で述べたアプリケーションが動作する性能要件を満たすことを示したが，広域分散環境においてノード数などのシステムの規模は現在も拡大し，各ノードの性能も向上し続けている．また，アプリケーションの性能要件も向上している．そのため，今後は MetaFa が満たすべき性能についても向上していく必要がある．

第5章　広域分散環境におけるメタデータを用いたデータアクセスのための仮想ファイルシステムPVN-FS

広域分散環境では，データは個々の計算機のファイルシステムに蓄積されていく．そのため，ユーザはファイルシステムの名前空間を把握しておく必要がある．SRB や iRODS などのシステムでは，メタデータを用いて検索することで，名前空間を把握せずにデータを利用できる．これらのシステムで検索結果からデータへアクセスする際には，システム専用のインタフェースを利用しなければならず，既存アプリケーションの改変が必要であった．

本章ではメタデータを用いたデータアクセスインタフェースの 1 つの手法として，広域分散環境でのデータアクセスを可能とし，既存アプリケーションの再利用が可能なユーザ空間でメタデータによる検索結果とディレクトリ構造をマップし，名前空間をユーザ・コミュニティ毎に構築する仮想ファイルシステム *"Personal View Namespace File System (PVN-FS)"* を提案する．ユーザは，PVN-FS を用いて 4 章で述べた MetaFa で収集・管理するメタデータを用いてデータ検索可能である．

5.1. メタデータを用いた検索可能なデータアクセスシステムの機能要件

本節では，3.5 で述べた問題を解決するために，メタデータを用いた検索可能なデータアクセスシステムの機能要件について述べる．メタデータを用いたデータ検索可能なデータアクセスシステムの機能要件を以下に示す．

- メタデータを用いてデータ検索可能
- 既存アプリケーションが再利用可能
- メタデータの記述性

データグリッドシステムでは，メタデータを用いてデータ検索可能，広域ネットワーク環境で動作するが，POSIX ファイルシステムの API との互換性を持たず，アプリケーションプログラムの修正が必要であった．また，Semantic File System(SFS) は，メタデータを用いてデータ検索可能，POSIX ファイルシステムの API と互換性を持つが，グリッドやクラウド環境のような広域ネットワーク環境では動くことを想定してない．SFS では，LAN 上に設置した NFS 環境をベースにして構築されている．また，メタデータのインデックスをローカルホスト上で管理することで，問い合わせやファイルアクセスの性能を担保している．例えば，SFS をグリッドなどの広域分散環境で動作させた場合，メタデータインデックスへの問い合わせやファイルアクセスがネットワーク遅延による影響を大きく受け，応答時間が増大し，性能低下が生じることが容易に想像可能である．

2.4.5 項で述べたようにパラメータサーベイアプリケーションは，数 KB から数 MB のデータに対してアクセスすることが多い．また，試行錯誤を考慮したワークフローシステムとデータアクセスインタフェースが連携する場合，データはメタデータを経由して，すぐにデータアクセスできる必要がある．パラメータサーベイアプリケーションでは，ユーザが出力したデータを解析処理した後に，アーカイブされることはあるが出力直後にデータの移動・複製が行われることはほとんどない．そのため，メタデータによってデータ検索したデータは，移動・複製が発生することは少ない．

データは，VO 上のサイトのどこかに存在しているので，リモートアクセス手法が必要となる．他のリモートアクセスと同様のデータアクセス性能が必要となる．取り扱うデータは，WORM 型であるため，データの編集機能は必要としない．

5.2. メタデータを用いたデータ検索可能なデータアクセスシステムの設計

本節では，前節で述べた要求事項を満たすシステムの設計方針について述べる．システムの設計方針は，以下の 3 点である．

1. メタデータを用いたデータ検索可能
2. POSIX ファイルシステム API との互換性
3. 広域分散するデータに対してアクセス可能

1 つめの設計方針は，MetaFa が提供するメタデータを用いて，検索広域に分散するデータを検索し，データアクセス可能にすることである．提案するデータアクセスシステムは，MetaFa が提供するメタデータに記述されているデータの識別子を保持する．

2 つめの設計方針は，既存ファイルシステムとデータアクセスの互換性を持たせる点である．つまり，本提案でのデータアクセスインタフェースは，POSIX ファイルシステム API をサポートする．これは，アプリケーションの利用者であるユーザの利便性のためである．**??**項で述べた要求要件にもあるが，既存アプリケーションをグリッド向け，もしくは特定のデータグリッドシステム向けに改変することは非常に手間となる．従来，クラスタコンピューティング環境等で動作していたアプリケーションをそのまま利用することができれば，ユーザやアプリケーション開発者の手間を削減することが可能となる．また，手元に存在する既存のデータと広域分散環境におけるデータの比較など行う際に，データアクセスインタフェースが既存ファイルシステムと互換性があることで，HDFS などの

POSIX ファイルシステムと互換性のないシステムで必要となるデータのインポートやエクスポートが不要となる．

3 つめの設計方針は，ワーカーノード等に保存され，広域分散するデータに対してアクセスするために SSH プロトコルを用いる．SSH プロトコルは，多くのノードでリモートアクセスのために利用されているため，利用できる可能性が高い．また，ファイル共有を行う場合には，ファイルを提供するサーバ側に何らかの設定が必要となる．データにアクセスするユーザが，ワーカーノードに対してルート権限を持つとは限らない．そのため，設定不要で利用可能な SSH プロトコルを採用した．

5.3. PVN-FS の概要

PVN-FS (Personal View Namespace File System) は，MetaFa の提供するメタデータを利用したデータ検索可能な仮想ファイルシステムである．PVN-FS は，メタデータによってユーザ毎に名前空間を構築可能で，既存の名前空間に組み込み可能である．PVN-FS では，メタデータをディレクトリに付与し，指定されたメタデータにマッチするデータがディレクトリ内に保持する．

PVN-FS の概要を図 5.1 に示す．PVN-FS は，指定されたメタデータを MetaFa-MMS へ問合せる．PVN-FS で扱うメタデータは，ファイルが持つ属性の名前，属性の値，演算子を意味する．MetaFa-MMS は，指定されたメタデータに合致するデータの一覧を PVN-FS へ返す．PVN-FS は，MetaFa-MMS からの検索結果を受け取り，ファイル属性データベースへ格納する．検索結果のデータは，ディレクトリへマッピングする．ファイルが参照された際は，該当するレコードをファイル属性データベースから読み込む．PVN-FS は，仮想的なファイルシステムであり，PVN-FS を実行したクライアント側では，データを保持しない．広域に分散して保存されているデータは，データを保有するワーカーノードへ SSH プロトコル経由でデータアクセスする．

メタデータを用いて検索した結果をデータベースに格納し，仮想ディレクトリ構造にマッピングすることで，従来のファイルシステムと同様なデータの指定方

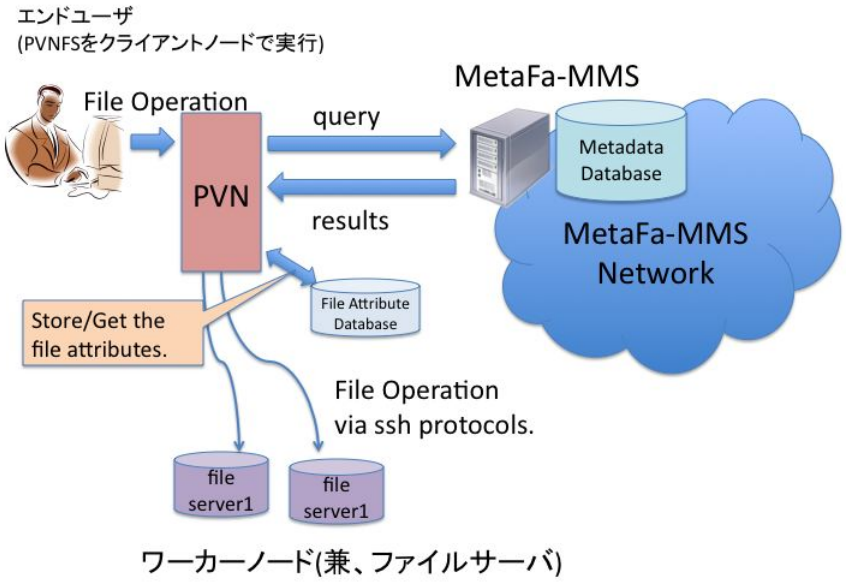

図 5.1 PVN-FS の構築，データアクセスフロー

法が可能になる．

既存のアプリケーションを再利用するために，POSIX ファイルシステム API と互換性を持つ必要がある．そのため，FUSE (Filesystem in Userspace)[4] をベースに実装した．FUSE は，Unix 系オペレーティングシステムのローダブル・カーネル・モジュールの一種で，一般ユーザがカーネルコードを修正することなく独自のファイルシステムを作成できる機能を提供する．これは，ファイルシステムのコードをユーザ空間で実行する際に，FUSE モジュールはカーネル側とのインタフェースを提供する．PVN-FS は，MetaFa と同様に Python で実装した．PVN-FS では FUSE のインタフェースを利用するため，fusepy モジュール [6] を用いた．

PVN-FS で変更したインタフェースを表 5.1 に示す．表 5.1 のインタフェースを実装することにより，ファイルシステムが提供しているインタフェースをオーバーラップし，ファイル属性をファイル属性データベースからの読み込みや，メタデータの検索，SSH 経由でのデータの読み込みを可能とした．PVN-FS の仮想ファイルシステム (/pvnfs) は，ディレクトリ構造をメモリ上に保持し，実態の

表 5.1 PVN-FS の提供するインタフェース

ファイル操作インタフェース	
read	pre-mount したデータを open し，read する．
readdir	指定されたディレクトリを read する． ディレクトリ内のファイルを表示する．
メタデータ操作インタフェース	
getattr	ファイルの属性情報をデータベースから取得する．
その他のインタフェース	
init	mount する際に呼び出される関数． ルートディレクトリ (/pvnfs) を設定する．
del	unmount する際に呼び出される関数． pre-mount したディレクトリを unmount する．

ないファイルシステムである．PVN-FS 上のディレクトリに対してファイル書き込みを行っても，ファイルが正しく生成できることを保証していないため，現在の実装では，PVN-FS 上でのデータの読み込み (read) のみをサポートしている．PVN-FS 上でアプリケーションを実行し，その出力結果をユーザのホームディレクトリ等へ書き込むことは可能である．

5.4. PVN-FS 上でのメタデータによるデータ検索

PVN-FS 上でのメタデータを用いたディレクトリの構成手法について説明する．

PVN-FS では，ディレクトリを作成するコマンドとして，pvn_mkdir.py を提供する．pvn_mkdir.py を用いてディレクトリを作成する場合，メタデータを指定する．pvn_mkdir.py の使用方法，メタデータの指定方法を図 5.2 に示す．pvn_mkdir.py の第 1 引数に，ディレクトリ名，第 2 引数に，属性，演算子，値を指定する．利用可能な演算子は，= , < , > , <= , >= などである．pvn_mkdir.py は，指定されたメタデータを MetaFa-MMS へ問い合わせ，問い合わせの結果をファイル属性データベースへ格納する．pvn_mkdir.py は，MetaFa-MMS が提供する get API

```
python pvn_mkdir.py directory_name 'attribute operator value'
```

図 5.2 メタデータの指定方法の例

を RPC で呼び出す．MetaFa-MMS は，引数のメタデータに該当するデータのリストを PVN-FS へ返す．PVN-FS は，受信したデータのリスト (シリアライズしたメタデータの集合) をファイル属性データベースへ格納する．PVN-FS は，ファイル属性データベースに格納する際，シリアライズされたメタデータを Python の辞書型データに変換し，基本メタデータとアプリケーションメタデータを分離する．基本メタデータは，i-node などファイル属性と拡張属性に分けて格納する．

PVN-FS での仮想ディレクトリは，ユーザが指定したメタデータを保有するデータの集合を意味する．PVN-FS では，図 5.2 のようにメタデータを指定することにより，指定されたメタデータを持つデータを作成したディレクトリ以下に収集する．PVN-FS では，指定したメタデータの属性値に完全一致する場合か，指定したメタデータの属性値の範囲に一致するかどちらかの検索が可能である．指定できる演算子は，SQL と同様の演算子が指定可能である．パラメータサーベイアプリケーションの一例であるタンパク質立体構造予測では，指定した塩基配列に一致するデータ，ジョブの開始時間と終了時間を指定して，その間に出力されたデータを検索するなどして，ユーザは利用することができる．

PVN-FS では，子ディレクトリは親ディレクトリのメタデータを継承する．つまり，PVN-FS で，子ディレクトリを作成した場合は，親ディレクトリがもつメタデータと子ディレクトリのもつメタデータを併せ持つデータの集合を作成する．PVN-FS での子ディレクトリは，親ディレクトリのもつメタデータとの積集合となる．PVN-FS では，ディレクトリ階層を深くするほど，各ディレクトリに付与したメタデータと，親ディレクトリの持つメタデータによるデータの絞り込みが可能である．PVN-FS のルートディレクトリ (/pvnfs) 以下にディレクトリを新規作成する場合，ひとつのメタデータを指定する．PVN-FS が MetaFa-MMS へ問い合わせその結果を，sqlite3 のファイル属性データベースへ格納する．このとき，データベースのテーブル名はディレクトリ名と同じにする．さらに，ファイル属性データベースでは，PVN-FS 上でディレクトリを作成するプログラム

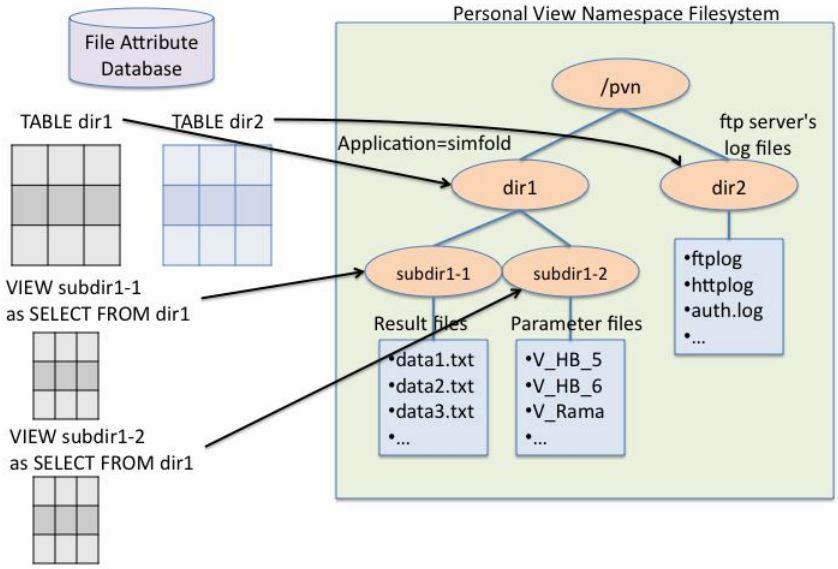

図 5.3 PVN-FS のディレクトリとファイル属性データベースのテーブルのマッピング

pvn_mkdir.py を実行した際の引数を保持するテーブル (ディレクトリテーブル) を持つ．pvn_mkdir.py では，子ディレクトリを作成する際に，まず，ディレクトリテーブルから親ディレクトリが持つメタデータ (属性名，属性値，演算子) を得る．子ディレクトリを作成する際に，指定された引数のメタデータと，ディレクトリテーブルから得られた親ディレクトリのもつメタデータを MetaFa-MMS へと送る．このとき，MetaFa-MMS からは，該当するデータの ID のリストのみを PVN-FS 側へ返す．得られた ID リストを用いて，親ディレクトリテーブルと AND 演算を行い，子ディレクトリ名でビューを作成する (図 5.3)．

5.5. PVN-FS のデータアクセスと名前空間

PVN-FS でのデータアクセス手法について述べる．PVN-FS でのデータアクセスプロトコルは SSH を利用している．ユーザは，メタデータを用いてデータ検索

した結果が保存されいているディレクトリにアクセスする．ディレクトリ内のデータは，PVN-FS を実行しているノード上では保持していない．実際のデータは，データを保有するホストへSSH プロトコル経由でデータアクセスを行う．SSH の公開鍵認証を行うことで，事前に ssh-agent を起動しておき，ssh-add で秘密鍵を登録することで，PVN-FS からは認証なしでデータへアクセスする．

PVN-FS の当初の実装では，read() が呼ばれる度に，データを保存しているノードに対して，SSH のセッションを開いて，ターゲットデータを open()/read() し，データの read() が終了すると，SSH のセッションを close していた．この実装方法では毎回 read する度に，SSH のコネクションを張ることになるため，fusepy のサンプルプログラムである sftp.py でのデータアクセスと比較して，オーバーヘッドが大きく約 2 倍の処理時間を要していた．さらに，同じファイルへアクセスしても，ファイルのキャッシュは残らないため，毎回オリジナルのデータへアクセスしなければならないという問題が存在した．そこで，あらかじめターゲットのデータがあるディレクトリをクライアント側でマウントする pre-mount 手法を実装した．pre-mount 手法では，pvn_mkdir.py を実行した際に，ターゲットとなるデータを保有するホストを SSHFS[22] で，クライアント側のファイルシステムにマウントする．SSHFS は，SSH 経由でリモートのサーバをマウントし，ローカルのファイルシステムと同じようにファイルを操作することができる，FUSE で作られたシステムである．pvn_mkdir.py では，SSHFS でターゲットデータをもつホストをマウントした後に，PVN-FS 上に作成したディレクトリ以下に，pre-mount したデータに対して，シンボリックリンクを作成する．さらに，SSHFS を実行する際に，kernel_cache を有効にした状態でマウントすることにより，クライアント側でファイルキャッシュが有効になるため，同じファイルを読み込んだ際には，ファイルキャッシュを利用するため高速に処理可能である．

PVN-FS の名前空間について述べる．PVN-FS は，”python pvnfs.py /pvnfs” と実行することにより，PVN-FS の名前空間 pvnfs を既存の名前空間に組み込み可能である．PVN-FS では，任意のポイントに独自の名前空間を構築可能である．現在の PVN-FS では，同一のファイル名が存在した場合，PVN-FS 内部では別のファイルとして識別可能であるが，同一ディレクトリ内では名前空間の衝突が発

生する.

表 5.2 PVN-FS 実験に用いたノードのスペック

PVN-FS クライアント	Intel Core i7 3.2GHz, Mem: 12GB OS:Ubuntu 9.04(x86_64) HDD: HITACHI HUA722020ALA330 2TB, 7200rpm, Filesystem: ext4
ファイルサーバ	Intel Xeon 3.6GHz, Mem: 2GB, OS: Ubuntu 9.04 HDD: Maxtor 6L160M0 160GB, 7200rpm, Filesystem: ext3

5.6. 実験 – PVN-FS 上でのデータ処理速度の計測

MetaFa で収集・管理しているメタデータを用いて PVN-FS を構築しデータ処理を実行し，I/O 性能（読込速度）を計測した．まずはじめに，PVN-FS による仮想名前空間が構築できているかどうか，UNIX コマンドを用いて動作確認を行った．本実験で用いたノードのスペックを表 5.2 に示す．

ファイルのオーナーが，ユーザ inet-lab のファイルを取得するように pvn_mkdir.py を実行した．コマンドの実行結果のスクリーンショットを図 5.4 に示す．pvn_mkdir.py の処理時間は，0.4 秒程度であった．ローカルファイルシステム上や，SSHFS でマウントしたディレクトリ上にディレクトリを作成した場合，2 ミリ秒から 7 ミリ秒程度であるから，およそ 200 倍の時間を要している．この 0.4 秒の間に，MetaFa-MMS へ問い合わせを行い，MetaFa-MMS からの結果をクライアント側にもつファイル属性データベースへ格納し，pre-mount を実行し，各ファイルへのシンボリックリンクを作成する．

1 つめの実験として，ファイルサーバにあるデータをクライアント側から読み込んで，UNIX のコマンド (cat，grep，sed，awk) を実行した．PVN-FS との比較のために，fusepy のサンプルプログラムである sftp.py，SSHFS でリモートホストのホームディレクトリをそれぞれ別のディレクトリにマウントし実験を行った．

```
minoru-i@blackjack:/pvn$ pvn_mkdir.py dir_1 'uid_name = inet-lab'
minoru-i@blackjack:/pvn$ ls
d2  d3  dir_1
minoru-i@blackjack:/pvn$ cd dir_1/
minoru-i@blackjack:/pvn/dir_1$ ls
cabin.c                     odtest.c    tchtest.c        villa.c
debian-501-i386-DVD-1.iso   registry.c  tctdb.c          vista.c
debian-501-i386-DVD-1.iso   relic.c     tctmgr.c         vltsv.c
debian-501-i386-DVD-1.iso   tcbtest.c   tcttest.c        xmlrpc_server_cgi.c
hvmgr.c                     tcfdb.c     tcucodec.c       xmlrpc_struct.c
md5.c                       tcfmgr.c    tcutest.c
odeum.c                     tcftest.c   tcutil.c
odidx.c                     tchmgr.c    timer_count.cpp
minoru-i@blackjack:/pvn/dir_1$
```

図 5.4　pvn_mkdir.py によるディレクトリ作成

また，ファイルサーバ上でマウントしているローカルディスク上でも同様のコマンドを実行した．計測方法としては，リモートホストのを sshfs/sftp.py でローカルマシンにマウントし，以下のコマンドを，それぞれ sftp.py，SSHFS，PVN-FS 上で 5 回ずつ実行し処理時間を計測した．

- cat ruby.c
- grep int array.c
- sed -e 's/rb/py/g' file.c
- awk 'BEGIN{sum=0}{sum+=$1}END{print sum/NR}' data8.txt

cat，grep，sed で利用したファイルは Ruby のソースコード，awk で利用した data8.txt ファイルは 10 万個の整数値が書かれたファイルである．実験に用いたファイルの行数，ファイルサイズを表 5.3 に示す．クライアントとファイルサーバ間のネットワークは，1 つのルータが設置され，Gigabit Ethernet で構成され，RTT は平均 0.32 ミリ秒であった．実験結果を図 5.5 に示す．

PVN-FS では，SSHFS を用いてターゲットとなるディレクトリをクライアント側で，pre-mount している．そのため，PVN-FS は SSHFS とほぼ同じ処理速度となっている．sftp.py は，read() が呼ばれると SSH プロトコル経由でリモートホ

表 5.3 実験に用いたファイル

ファイル名	行数	ファイルサイズ
ruby.c	1855 行	42KB
array.c	3887 行	95KB
file.c	4823 行	108KB
data8.txt	10 万行	370KB

ストのファイルを open し読み込む．sftp.py では読み込んだファイルのキャッシュを保有しないため，毎回 SSH プロトコル経由でリモートホストのファイルを読み込むことになるため，どのコマンドにおいても SSHFS，PVN-FS と比較して処理時間を要している．SSHFS，PVN-FS では 1 度読み込んだファイルはメモリ上にファイルキャッシュを保有しているため，2 回目以降のアクセスではファイルキャッシュを利用し，高速に処理可能であった．awk の場合は，ファイルサーバのローカルディスク上での結果より，SSHFS の方がわずかながら速い結果が得られている．SSHFS でのファイルキャッシュが有効に働き，CPU の差によるデータ処理速度に違いが出たと考えられる．本実験より，PVN-FS は SSHFS をベースとして実装し，pre-mount 手法であらかじめターゲットとなるディレクトリをマウントし，シンボリックリンクを作成しておくことで，SSHFS に遜色ない I/O 性能を実現したことを確認した．

2 つめの実験として，1 つめの実験と同様の構成で，RTT を増加させて同様のコマンドを実行し，処理時間を計測した．本実験では，PVN-FS を実行しているノードと，実際にアクセスするデータの間にネットワーク遅延を発生させることにより，広域分散環境でのデータアクセスを達成できているかどうかを確認する．

ファイルサーバを収容しているレイヤ 2 スイッチングハブとファイルサーバの間に，遅延・エラー障害発生器である Anue Systems 社製のネットワークエミュレータ GEM を設置し，ネットワーク遅延を発生させた．ネットワークエミュレータ側では，ネットワーク遅延を常に発生させるようにし，平均して遅延を 0.56 ミリ秒，1 ミリ秒，5 ミリ秒，10 ミリ秒，50 ミリ秒，100 ミリ秒，200 ミリ秒に設定した．実際にクライアントからファイルサーバへの RTT を ping で測定したと

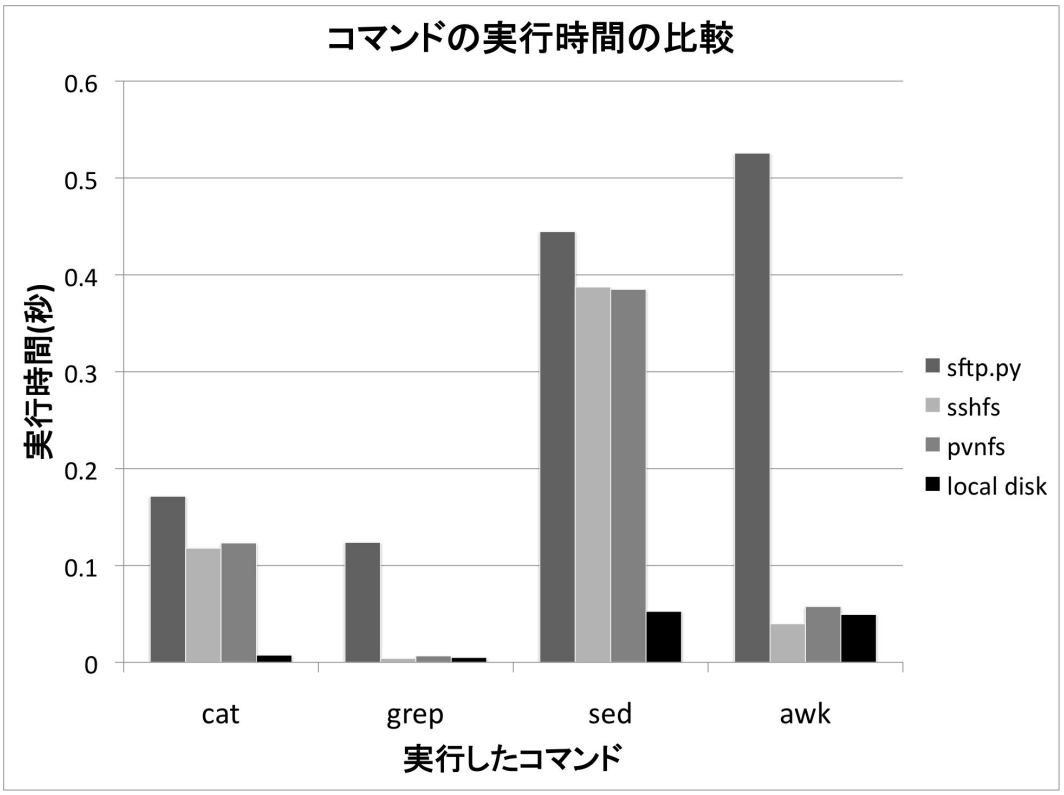

図 5.5 コマンドの実行時間の比較

ころ，それぞれ 0.7 ミリ秒，1.3 ミリ秒，5.3 ミリ秒，10.3 ミリ秒，50.9 ミリ秒，100.3 ミリ秒，200.3 ミリ秒であった．図 5.6，5.7，5.8，5.9 は，クライアントとファイルサーバの間の RTT を増加させた場合の cat，grep，sed，awk の実行時間を示す．また，同様の条件で ls の処理時間を計測した．図 5.10 にクライアントとファイルサーバ間の RTT を増加させた場合の ls の実行時間の変化を示す．図 5.10 の縦軸は，対数表示である．1 つめの実験と同様に，sftp.py ではファイルキャッシュがないために RTT の増加とともにコマンドの実行時間が延びている．SSHFS，PVN-FS では初回のデータアクセスのみが，RTT の増加に伴い処理時間も増加するが，2 回目以降のデータアクセスではファイルのキャッシュを利用するため，RTT に依存することなくデータ処理可能である．この傾向は cat，grep，sed，awk での全ての実験結果に現れている．SSHFS をベースとして構築してい

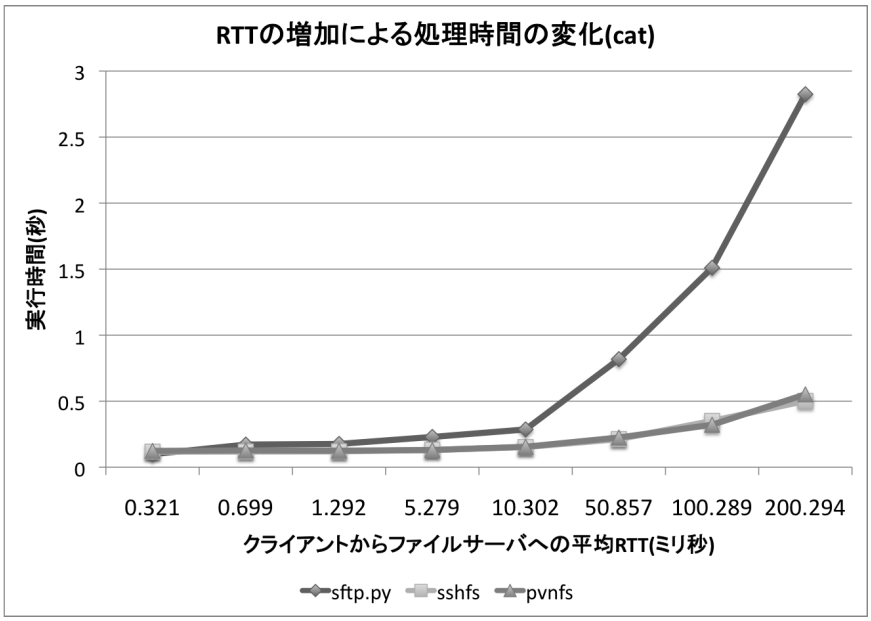

図 5.6 RTT を増加させた場合の実行時間の比較 (cat)

る PVN-FS のオーバーヘッドは，最大で 11.8 ミリ秒であり，ユーザの体感としてはほとんど無視できると考える．図 5.10 で示した ls の実験結果では，sftp.py はこれまでの実験結果と同様に RTT の増加とともに，処理時間が増大している．SSHFS では，実行時間は RTT の増加とともに緩やかに，実行時間が大きくなっていた．これは，getattr のようなファイル属性の取得を全てのファイルに対して行わなければならないため，SSHFS でも処理時間を要していることが考えられる．しかしながら，SSHFS では同じディレクトリを参照をしている限り，キャッシュが有効であるので実行時間を抑制可能である．一方，PVN-FS では PVN-FS 上で取り扱うデータのファイル属性は，すべてクライアント上にファイル属性データベースとして保持している．そのため，PVN-FS では ls において，RTT に依存せずに一定の実行時間でファイル属性参照可能である．

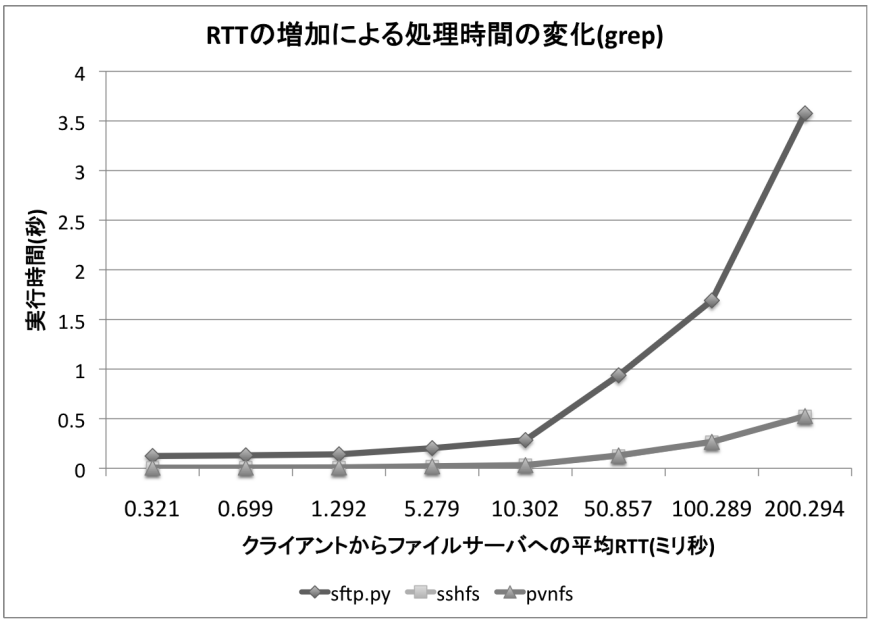

図 5.7 RTT を増加させた場合の実行時間の比較 (grep)

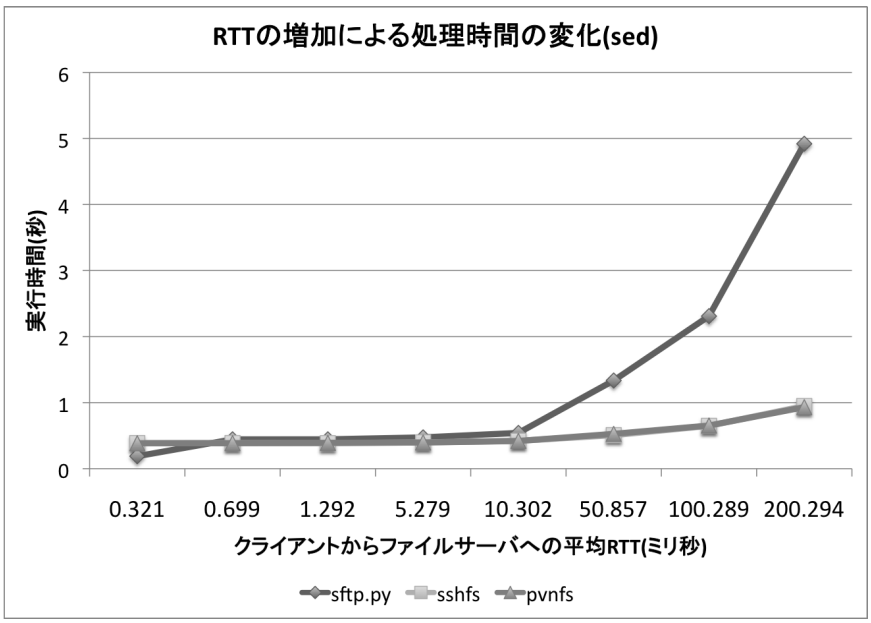

図 5.8 RTT を増加させた場合の実行時間の比較 (sed)

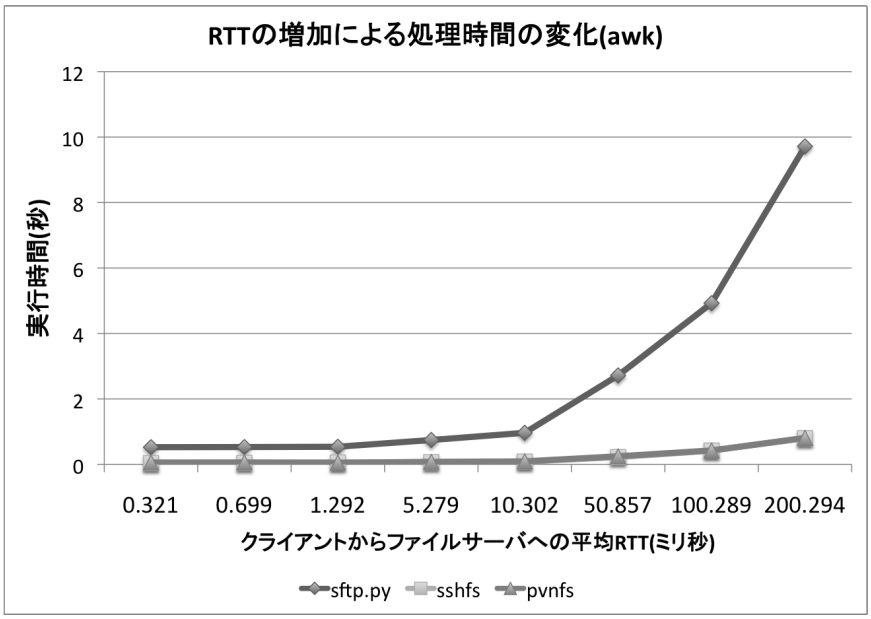

図 5.9 RTT を増加させた場合の実行時間の比較 (awk)

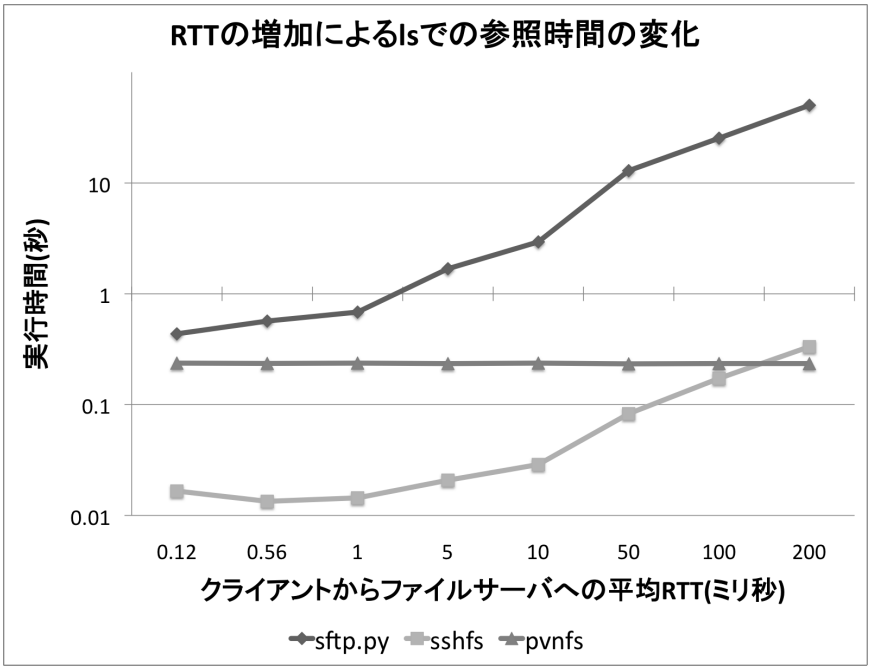

図 5.10 RTT を増加させた場合の実行時間の比較 (ls)

5.7. PVN-FSの考察

本節では，実験結果などからPVN-FSを考察する．

5.7.1 PVN-FSのI/O性能

メタデータを用いた検索可能なファイルシステムPVN-FSでは，I/O性能について検証を行った結果，SSHFSの特性を活かして，キャッシュを有効に利用できるためRTTの大きさに依存せず，SSHFSに遜色ないI/O性能を実現することができた．しかしながら，RTTが増大していくなかで，lsなどのデータ属性を参照するようなコマンドは，SSHFSでも実行時間が増加している．lsコマンドで，次々と別のディレクトリやデータを参照する場合は，SSHFSでもキャッシュの有効活用ができないため，処理時間がRTTの大きさに依存する．一方，PVN-FSではPVN-FS上で取り扱うデータのファイル属性は，すべてクライアント上にファイル属性データベースとして保持している．そのため，PVN-FSではlsなどのコマンドにおいて，RTTに依存せずに応答時間が決定する．そのため，単純にSSHFSを利用するだけでは得ることの出来ない，I/O性能をPVN-FSでは実現した．また，sftp.pyやSSHFSでは，ターゲットディレクトリ単位でマウントしなければならない．そのため，同じ属性をもつデータが別のホスト，別のディレクトリに存在した場合，プログラムにデータを渡す際はそれぞれのデータの絶対パスなどで指定しなければならない．PVN-FSでは，同一の属性をもつデータは指定されたディレクトリに収集することができるので，絶対パスなどを指定せずに一挙にデータ処理が可能となる．PVN-FSでは，SSHFSやsftp.pyのように，リモートのディレクトリをクライアントのファイルシステムにマウントするだけでは得られない機能を実現している．

ユーザの体感としてPVN-FSのデータアクセス性能は，ディレクトリ作成時に，0.4秒ほど待たされる．しかし，0.4秒という待ち時間はユーザの操作感からするとほとんど無視できるのではないかと考える．

5.7.2 PVN-FS でのデータにおける一貫性制御

PVN-FS は，ディレクトリを作成した段階で，ファイルが存在するノードのディレクトリを SSHFS であらかじめ pre-mount 手法によってクライアントマシン上にマウントため，実際のデータアクセスは SSHFS 経由で行われる．リモート側のデータが別のユーザによって更新された場合は，UNIX セマンティクス (アクセスの順によって，常にファイルの最新の値を返す) の考えに従い，データ更新後に PVN-FS でデータへアクセスした場合は，更新されたデータへアクセスする．

5.7.3 PVN-FS でのディレクトリにおける一貫性制御

PVN-FS 上でのディレクトリは NFS などのネットワークファイルシステムで採用されているセッションセマンティクス (ファイルの open から close までの間，ファイルの内容や状態が他のノードで行われた操作の影響を受けない一貫性制御) を採用している．PVN-FS においては，ユーザが PVN-FS 上で，メタデータを用いてディレクトリを作成した段階をセッションセマンティクスにおける open，ディレクトリを削除した段階をセッションセマンティクスにおける close として扱う．PVN-FS が保有するメタデータが，MetaFa-MMS に問い合わせて検索結果を取得した以降に更新するようにしていないためである．データの更新が発生した場合は UNIX セマンティクスに従っているため，データへアクセスすると最新の値が得ることが可能であるが，データの移動・削除が発生した場合には，PVN-FS 上のメタデータがセッションセマンティクスに従うため，当該データへアクセスできなくなる．

本研究で対象とする HPC アプリケーションで取り扱うデータは，Write Once Read Many(WORM) と呼ばれ，一度書き込まれたデータは，後から変更されることはなく参照のみが行われる．また，対象とする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上での HPC などでは，1 つのデータだけが削除されることは少なく，解析処理後，不要になったデータはまとめて移動や削除が行われることが多い．そのため，データの更新や移動，削除自体が頻繁に発生しないため，PVN-FS においてデータへアクセス不能になる可能性は低い．また，

シミュレーション途中にデータアクセスするため，データの移動はなく，想定しているパラメータサーベイなどではPVN-FSが保持するメタデータが古いため，ユーザがデータに対してアクセスできなくなることは発生しないと考える．

WORM以外のデータも対象とするため，PVN-FS側でメタデータデータベースの内容が最新かどうかを確認するようなコマンド (pvn_update) を提供している．

5.7.4 PVN-FSの利便性

PVN-FSの利便性を定性的に評価するため，ユーザがPVN-FSを実行するために必要なステップについて述べる．

ユーザが，PVN-FSを利用するためには，Linuxが入ったノードに対して，FUSE，fusepy，sqlite，SSHFSをインストールするだけで利用可能である．ユーザは，python pvnfs.py /mntとコマンドを実行し，pvn_mkdir.pyでディレクトリを作成するだけで，広域に分散するデータを仮想的に収集し，データ処理が可能である．

5.7.5 アクセス透過性

PVN-FSでは，メタデータで問合せを行い，仮想ディレクトリ上でアクセスした場合，オリジナルのデータが消失しない限り，同じ参照方法でアクセス可能である．PVN-FSでは，ファイル名によって，アクセス透過性を実現している．

5.7.6 アプリケーションの再利用性

PVN-FSは，ファイルシステムをユーザランドで実現するFUSEを用いて実装した．FUSEを用いて実装することにより，従来のPOSIXファイルシステムAPIをそのまま利用できる．cat，ls，awkなどのUNIXコマンドをPVN-FS上で実行できることを確認した．

5.8. 今後の課題

今後の課題は，PVN-FS において，自動的に pvn_update を実行することである．1 度構築した PVN-FS は，unmount を行い，新たにディレクトリを作成した場合か，pvn_update コマンドを手動で実行する必要がある．ユーザに対して，データはシンボリックリンクとして提供しているため，実際にプログラムがアクセスするまでデータが存在するかどうかは不明である．PVN-FS がファイルシステムとしての完成度をあげるためにも，さらなる開発が必要となる．

PVN-FS において，メタデータを用いてデータ検索する際の，適合率，再現率の検証を行う予定である．また，MetaFa システム側では，メタデータによる検索結果に対して順位付けを行うなど検索に対する改良を行う予定である．

現在は，PVN-FS 上でメタデータを記述し，完全一致か，範囲検索でデータを発見し，アクセスしている．今後は，指定したメタデータに該当するデータの平均を求めるなど，メタデータの記述能力の向上を図る．

5.9. まとめ

本提案 PVN-FS では，メタデータをデータアクセスインタフェース扱うための 1 つのインタフェースとして，階層型ファイルシステムの名前空間に組み込み可能な形でユーザへデータアクセスを提供する．POSIX ファイルシステム API をサポートし，既存アプリケーションの再利用が可能となっており，本提案では，広域に分散するデータに対して，メタデータによるデータアクセスシステムの 1 つを実現したと考える．

第6章　結論

この研究の考察，まとめを述べ，この研究の今後の展望について述べる．

6.1. 本研究の考察

本研究では，グリッドやクラウドなどの広域分散環境において，科学技術計算シミュレーション，特に，パラメータサーベイアプリケーションのデータに対して，メタデータを用いたデータアクセスを実現し，データ参照コストを削減することを目標としてきた．

MetaFa および，PVN-FS で実現したメタデータを用いたデータアクセスでは，メタデータによる検索があるため，データの識別子を得るために，通常のファイルシステムよりデータ I/O にかかるコストが大きくなる．本研究では，メタデータを用いることによって生じる I/O コストの増加に対して，以下の点で工夫を行った．

- MetaFa
  - ユーザ側からのメタデータ I/O を小さくするために，メタデータ管理サーバを VO 上の各サイトに配置した．このことより，メタデータを参照するためのコストを削減している．
- PVN-FS
  - 指定したメタデータに一致するデータをグループ単位でディレクトリに配置する．ディレクトリベースで検索結果を扱うことにより，データアクセス時にはメタデータが不要である．ファイルアクセス時には，

ファイル識別子を得るためにローカルに保持しているデータベースへアクセスするだけでよい．そのため，ファイルアクセス時に毎回メタデータを参照する必要はない．

6.2. 本研究で実現したこと

本研究では，広域分散環境におけるメタデータを用いたデータアクセス基盤，および，メタデータ収集・管理基盤を実現するために必要な以下の要求要件に対して，次節以降で示す点について実現した．

(1) データアクセスインタフェース

1. データ検索可能
2. 既存アプリケーションの再利用可能
3. メタデータ記述性

(2) データのメタデータ収集

1. メタデータ収集の自動化
2. メタデータ収集時にシステムに対する負荷を与えない
3. データの鮮度を考慮したメタデータ収集

(3) 広域分散環境におけるメタデータ管理

1. 1 つのメタデータ管理システムで複数のメタデータスキーマを管理
2. メタデータ I/O の高速化
3. メタデータ複製における一貫性制御
4. データの鮮度を考慮したメタデータ管理

6.2.1 メタデータを用いた検索可能なデータアクセスシステム

従来のメタデータを用いたデータ検索可能なデータアクセスシステムでは，POSIX 互換なシステムでは，広域ネットワーク上での動作を想定しておらず，非 POSIX 互換なシステムでは，広域ネットワーク上で動作するが，既存のアプリケーションの再利用ができない問題があった．

本提案では，MetaFa で管理しているメタデータでデータ検索可能であり，検索結果をディレクトリとして表現し，既存の階層型ファイルシステムの名前空間に組み込むことを可能とした．従来の階層型ファイルシステムに組み込むことで，PVN-FS 上に存在するデータに対して，ファイル名で何度もアクセスできるようにした．

また，FUSE を利用し POSIX 互換なシステムを実装することにより，既存のアプリケーションの再利用も可能である．PVN-FS は，メタデータを用いたデータアクセスシステムの 1 つを実現した．

6.2.2 広域分散環境でのメタデータ収集

従来のメタデータ収集方法は，ファイルシステムをクローリングするなど，ディスク I/O へ影響を与えるものが多かった．また，メタデータを自動的に収集するプログラムもメタデータのスキーマを含めて管理する．グリッドなどの環境では，どのワーカーノードで，ジョブが実行されるかわからないため，事前にどのノードにどのアプリケーションメタデータのスキーマを管理するべきかを割り当てることはできない．MetaFa では，ファイルシステムイベントを取得することにより，ワーカーノードに与える負荷が小さく，メモリ上にメタデータを保持することによりアプリケーションのファイル書き込み I/O 性能を維持することを可能とした．また，データの鮮度を考慮するために，メタデータをリアルタイムに収集することを可能とした．

6.2.3 広域分散環境でのメタデータ管理

従来のメタデータ管理方法では，メタデータのスキーマを管理するため異なるアプリケーションメタデータを管理することができなかった．MetaFa では，スキーマを管理せずに，シリアライズした形式でメタデータを管理することにより，複数の異なるアプリケーションメタデータを簡潔に管理することを可能とした．

また，各組織にメタデータ管理サーバを配置し，全てのメタデータ管理サーバが全てのメタデータを保持することにより，クライアント側からのメタデータI/Oを小さくなるようにメタデータ複製処理をする．

6.3. 本研究によって得られた知見

本研究を遂行するにあたり，得られた知見を述べる．

- データ管理におけるメタデータのあり方
  科学技術分野では，アプリケーションのデータに対してメタデータを記述し，データ管理，データアクセスに活用してきた．アプリケーション毎に，メタデータのスキーマが異なるため，メタデータ管理システムが構築されている．このメタデータ管理基盤を共通化することにより，データグリッドの構築コストを削減できると考えられる．

- データアクセスシステムとしての階層型名前空間のあり方
  従来の階層型ファイルシステムにおいては，ペタバイトスケールのデータ管理は困難である．本研究で提案した PVN-FS では，既存アプリケーションの再利用性を考慮して階層型ファイルシステムの名前空間を拡張し，検索結果をファイルシステムに組み込んだ．名前だけが唯一のアクセス手段である階層型ファイルシステムとメタデータを用いたデータアクセスシステムは，ユーザの利便性の観点からすると，圧倒的にメタデータを用いたデータアクセスシステムが有利である．しかし，現状では過去のプログラム資産を利用したいという要望があるため，階層型ファイルシステムを抜本的に変更することは難しい．そのため，階層型名前空間の中に，メタデー

タによる検索した結果を管理する確率的な名前空間を組み込むのが現段階では最適な解であると考える．

6.4. 本研究の貢献

本論文では，グリッドコンピューティング，クラウド環境を利用するユーザがメタデータを用いたデータ管理を行うために，以下の貢献を果たした．

メタデータを統一的に管理する基盤の実現

従来，複数のメタデータを管理するメタデータ管理システムでは，メタデータスキーマの管理が必要となり，メタデータ管理システムの内部が複雑となり，十分な性能が得られないことがあった．本研究で提案した MetaFa では，どのようなメタデータでもスキーマレスで統一的に管理することで，シンプルなメタデータ管理システムアーキテクチャとなり，十分な性能を得ることができる．

グリッド環境におけるメタデータを用いたパラメータサーベイアプリケーション向け階層型ファイルシステムの拡張

ユーザは，データを探すために，従来の階層型ファイルシステムにおいては，ディレクトリの中身を確認していた．本研究で提案した PVN-FS では，メタデータを用いて階層型ファイルシステムに存在するデータを検索可能とした．さらには，広域分散環境で動作するように設計・構築したため，遠隔にあるデータも，ユーザ側から操作可能となった．パラメータサーベイアプリケーションでは，数 KB と小さなファイルに対してアクセスする．ジョブ終了直後や，ジョブ実行中にでも出力されたデータに関するメタデータを数十秒単位で利用可能にし，メタデータによるアクセスを実現した．

6.5. 今後の展望

- メタデータのアクセス権，利用権制御
  MetaFa や PVN-FS では，グリッドコンピューティングミドルウェア Globus Toolkit や，グリッド認証基盤 Grid Security Infrastructure (GSI) を前提としている．プロトタイプ実装では，SSH による認証と各ノードにユーザがあることを前提としていた．これは，データに対するアクセス権限を UNIX におけるユーザ，所属グループによるファイルアクセス権限のみで制御していたことになる．

  メタデータを用いたデータアクセスシステムの場合，ファイルアクセス権限による制御だけで十分な場合と不十分な場合がある．メタデータにはデータの中身そのものは記述されていないが，内容を示すポインタや，実験パラメータなどが記述されている．他のユーザがメタデータを見ただけで，実験内容が判別できる可能性がある．これは，バイオサイエンスや生命科学分野での製薬実験などにおいて大きな問題となる．このため，メタデータを取り扱うミドルウェアでのメタデータに対するアクセスを制限するなどを行わなければならない．この問題を解決するために，MetaFa を VO へ対応させ，GSI を基盤として，メタデータに対するアクセス権の強化を図る．

- 検索率の向上
  現在は，バックエンドで管理するデータベースにて，SQL による完全一致，部分一致のみがメタデータ検索において結果として表れる．この時，ユーザに関係ないデータも検索結果に表れる．データとデータの関連性，ユーザとデータ，ユーザとアプリケーションの関連性などを取り入れて，検索率の向上を図ることが求められる．

- 異なるデータ形式へのアクセス
  現在は，ファイルを前提としたシステムであるが，グリッドやクラウド環境ではデータベースにデータを格納することも多い．そのため，現在のメタデータ収集手法では通用しない．ファイル形式やデータベース形式など形式を問わず，シームレスにアクセスできる必要がある．

- さらなるデータアクセスインタフェースの開発
  本研究では，SSHFS/FUSE をベースとした PVN-FS を提案した．しかしながら，PVN-FS は SSH でアクセス可能であることを前提としているため，SSH が利用できないサーバに存在するデータへはアクセスできない．グリッドコンピューティングの共通基盤ソフトウェアである Globus Toolkit などのグリッド環境で動作するメタデータを用いたデータ検索可能なユーザインタフェースの設計・開発が必要となる．write など POSIX ファイルシステム API を完全にサポートした検索可能なユーザインタフェースを Gfarm や GSI-SFS[75] などに組み込むことが必要となる．

  この他にも，センサネットワークアプリケーションでは，ユーザがアクセスするセンサデータには日時の粒度というパラメータが存在する．気象センサから出力される気温データであれば，ユーザは現在の気温を 10 分などといった粒度でアクセスし，同じ場所の気温でも去年の気温が知りたいという場合には，10 分といった粒度ではなく，1 日や 1 週間の平均気温といった粒度のデータが必要となる．データアクセスインタフェースとして，メタデータに演算処理を記述し，生のデータを得るのではなく，必要な形に加工されたデータをユーザに提供する機能が求められる．

- システム規模の拡張，性能向上に対する対応
  2.4.5 項で述べたアプリケーションの性能要件を満たすことを示したが，グリッド，クラウドにおいてノード数などのシステムの規模は現在も拡大し，各ノードの性能も向上し続けている．パラメータサーベイアプリケーションで取り扱うデータに関しては，各シミュレーションが取り扱うデータサイズは大きく変化はないが，グリッドのシステム規模，ワーカーノードの性能向上によって，データ量は今後 100 万個オーダー，それ以上になることが予想される．そのため，今後は MetaFa が満たすべき性能，特に取り扱うデータ量，メタデータの個数について MetaFa の性能向上を図る必要がある．

6.6. 本研究のまとめ

本研究では，グリッド環境における” メタデータ・ファイル I/O オペレーション” を実現することを目標に，そのための基盤技術を確立することにより進めてきた．

科学技術シミュレーションやグリッド環境におけるパラメータサーベイアプリケーションの調査を行い，データの特性などについて分類した．また，科学技術シミュレーションやグリッド，クラウドコンピューティング，および，既存のデータアクセス手法におけるメタデータについて調査し，要求要件をまとめ，問題点に基づき，システムの設計を行った．

4 章では，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，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．

5 章では，メタデータを用いたデータ検索・データアクセスシステム PVN-FS について提案した．PVN-FS では，MetaFa が管理するメタデータを用いてデータ検索可能であり，既存のファイルシステム上に，/pvnfs ディレクトリとして組み込み可能である．PVN-FS は，FUSE を用いて実装することにより，POSIX ファイルシステム API をオーバーラップし，既存アプリケーションの再利用可能とし

ている．SSHFS をベースとして実装し，pre-mount 手法においてファイルキャッシュを有効に活用するなどして，十分な I/O 性能を実現した．

本研究で提案したグリッド環境下で動作するメタデータ管理システム MetaFa，および，メタデータを用いた検索可能な仮想ファイルシステム PVN-FS において，目標とする” メタデータ・ファイル I/O オペレーション” をパラメータサーベイアプリケーションを対象として実現した．

謝辞

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，博士後期課程合わせて 7 年もの研究生活において，多くの方々に，研究指導などを受けてきました．本研究をまとめるにあたり，私の研究活動に関わった多くの人に感謝の意を述べたいと思います．本研究の主指導教員であり，研究に対しての有益なコメントを頂き，素晴らしい環境のなかで研究活動を行わせていただいた奈良先端科学技術大学院大学 情報科学研究科 インターネット・アーキテクチャ講座，慶應義塾大学 メディアデザイン研究科 砂原秀樹 教授に，感謝の意を表します．本研究の副指導教員であり，私の研究に対してたくさんの有益なコメントを頂いた奈良先端科学技術大学院大学 情報科学研究科 インターネット工学講座 山口英 教授に謝意を表します．本研究の副指導教員であり，私の研究テーマのきっかけを与えてくださり，研究活動のサポートをしていただいた奈良先端科学技術大学院大学 情報科学研究科インターネット・アーキテクチャ講座 藤川和利 准教授に深謝の意を表します．大阪大学 情報科学研究科 伊達進 准教授からは他大学ながら研究指導頂き，また有用な助言を頂き，深く感謝いたします．IT-Keys の傍ら，論文のチェックや研究活動のサポートをしていただいた奈良先端科学技術大学院大学 情報科学研究科インターネット・アーキテクチャ講座 猪俣敦夫 特任准教授に感謝の意を表します．私が奈良先端科学技術大学院大学で過ごした 7 年間の間に，様々な人にコメントを頂き，研究活動の励みになりました．九州工業大学 情報科学センター 中村豊 准教授，独立行政法人 情報通信研究機構 河合栄治氏，奈良先端科学技術大学院大学 インターネット・アーキテクチャ講座 和泉順子 助教，総合情報基盤センター 次世代システム研究グループ 垣内正年 助教，総合情報基盤センター 次世代システム研究グループ 寺田直美 助教，奈良先端科学技術大学院大学 インターネット・アーキテクチャ講座 松浦知史 特任助教に感謝いたします．

研究を進めていく上で，様々な事務処理においてサポートをして頂きました，奈良先端科学技術大学院大学 総合情報基盤センター 事務補佐員 呂悠妃 女史に感謝いたします．曼陀羅システムという研究環境を提供し，研究活動のサポートをして頂いた，総合情報基盤センター 情報基盤技術サービスグループの皆様に感謝いたします．

研究室生活のなかで，多くの人にお世話になったことを感謝します．多くの先輩方に，研究指導や研究室生活など様々なことを学びました．ここに感謝いたします．特に，同志社大学 工学部 理工学部研究員 島田秀輝氏，立命館大学 総合理工学研究機構 新井イスマイル氏，独立行政法人 産業技術総合研究所 情報技術研究部門 インフラウェア研究グループ 研究員 広渕崇宏氏に感謝の意を表します．また，一緒におしゃべりをしたり，ご飯を食べたり，研究生活を過ごす上での一時の清涼剤となり，苦しいときも，支えとなった後輩達に感謝いたします．特に，石橋賢一氏，洞井晋一氏，岡本慶大氏には感謝いたします．大学院生活の中で，楽しい同期に恵まれ修士時代から現在まで様々な面で支えになりました．インターネット・アーキテクチャ講座 2004 年度入学の同期である，青木太一氏，大橋純氏，笠藤麻里女史，斎田佳輝氏，森本健志氏に感謝いたします．最後に，7 年間という大学院生活において，両親や家族のサポートややりたいことをしなさいという励ましの言葉がなければ，無事に修了することはできませんでした．暖かく見守ってくれた家族や実家に帰ったときに暖かく迎え入れてくれた姉夫婦，兄夫婦，5 人の甥・姪に感謝いたします．

参考文献

[1] Dublin Core. http://dublincore.org/.

[2] Dummyet home page. http://info.iet.unipi.it/~luigi/dummynet/.

[3] e-Science. http://www.rcuk.ac.uk/escience/news/firstphase.htm.

[4] File System in userspace. http://fuse.sourceforge.net/.

[5] FITS Documents. http://fits.gsfc.nasa.gov/documents.html.

[6] fusepy: Python bindings for FUSE with ctypes. http://code.google.com/p/fusepy/.

[7] Hadoop distributed file system. http://hadoop.apache.org/hdfs/.

[8] Iozone Filesystem Benchmark. http://www.iozone.org/.

[9] Lustre a Network Clustering FS. http://www.lustre.org/.

[10] memcached – a distributed memory object caching system. http://memcached.org/.

[11] MongoDB. http://www.mongodb.org/.

[12] MSDN Blogs, What's in Store. http://blogs.msdn.com/b/winfs/.

[13] NetCDF (network Common Data Form). http://www.unidata.ucar.edu/software/netcdf/.

[14] Numerical Data – GRIB2 Format. http://www.weatheroffice.gc.ca/grib/index_e.html.

[15] Open Grid Forum. http://www.ogf.org/.

[16] PlanetLab. http://www.planet-lab.org/.

[17] Protocol Buffers – Google's data interchange format. http://code.google.com/p/protobuf/.

[18] RDF – Semantic Web Standards. http://www.w3.org/RDF/.

[19] Search engine strategies conference 2006. http://www.searchenginestrategies.com/sew/summer06/.

[20] Spotlight. http://www.apple.com/macosx/features/300.html#spotlight.

[21] SQLite. http://www.sqlite.org/.

[22] SSH Filesystem. http://fuse.sourceforge.net/sshfs.html.

[23] Storage Resource Broker. http://www.sdsc.edu/srb/index.php.

[24] Top 500 Supercomputing Sites. http://www.top500.org/.

[25] UCAR. http://www.ucar.edu/ucar/.

[26] Unidata. http://www.unidata.ucar.edu/.

[27] WMO. http://www.wmo.int/.

[28] XML-RPC. http://www.xmlrpc.com/.

[29] SC10: The International Conference for High Performance Computing, Networking, Storage and Analysis. http://sc10.supercomputing.org/, 2010.

[30] A. Finkelstein, C. Gryce and J. Lewis-Bowen. Relating requirements and architectures: a study of data-grids. *Journal of Grid Computing*, 2(3):207–222, 2004.

[31] A. Rajasekar, M. Wan, R. Moore and W. Schroeder. A Prototype Rule-based Distributed Data Management System. In *HPDC workshop on Next Generation Distributed Data Management*, May 2006.

[32] A. Traeger, E. Zadok, N. Joukov and C. P. Wright. A nine year study of file system and storage benchmarking. *ACM Transactions on Storage*, 4:51–56, May 2008.

[33] A. Verma, S. Venkataraman, M. Caesar and R. Campbell. Efficient metadata management for cloud computing applications. Technical report, University of Illinois, 2010.

[34] B. Salmon, S. W. Schlosser, L. F. Cranor and G. R. Ganger. Perspective: Semantic data management for the home. In *Proccedings of the 7th conference on File and storage technologies (FAST'09)*, pp. 167–182, Feb 2009.

[35] C. Baru, R. Frost, R. Marciano, R. Moore, A. Rajasekar and M. Wan. Metadata to support information-based computing environments. In *Proceedings of the 2nd IEEE International Conference on Metadata97*, Sep 1997.

[36] C. Baru, R. Moore, A. Rajasekar and M. Wan. The SDSC storage resource broker. In *Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative research*, pp. 5–17, Nov 1998.

[37] C. Yeh, T. Tseng and Y. Hsu. A reliable dht-based metadata server cluster. In *The 2008 International Computer Symposium (ICS2008)*, pp. 273–278, Nov 2008.

[38] CIPA. Exif Print. `http://www.cipa.jp/exifprint/index_j.html`.

[39] corbet. Watching filesystem events with inotify. `http://lwn.net/Articles/104343/`.

[40] D. Crockford. The application/json Media Type for JavaScript Object Notation (JSON). RFC 4627 (Informational), July 2006.

[41] D. K. Gifford, P. Jouvelot, M. A. Sheldon, and W. O'Toole, Jr. Semantic file systems. In *Proceedings of 13th ACM Symposium on Operating Systems Principles*, pp. 16–25. ACM SIGOPS, Oct 1991.

[42] D. Roselli, J. R. Lorch and T. E. Anderson. A comparison of file system workloads. In *Proceedings of the 2000 USENIX Annual Technical Conference*, pp. 41–51, Jun 2000.

[43] I. Foster and C. Kesselman. *The Grid2 Blueprint for a New Computing Infrastructure*. Morgan Kaufman, 2003.

[44] I. Foster, Y. Zhao, I. Raicu and S. Lu. Cloud Computing and Grid Computing 360-Degree Compared. In *Proceesings of IEEE Grid Computing Environments 2008 (GCE08)*, pp. 1–8, Dec 2008.

[45] J. Greenberg, K. Spurgin and A. Crystal. Functionalities for automatic metadata generation applications: a survey of metadata experts' opinions. *International Journal of Metadata, Semantics and Ontologies 2006*, 1(1):3–20, 2006.

[46] J. Nielsen. The Death of File Systems. `http://www.useit.com/papers/filedeath.html`, 1996.

[47] J. Thomson, D. Adams, P. J. Cowley and K. Walker. Metadata's Role in a Scientific Archive. *IEEE Computer Magazine*, 36:27–34, 2003.

[48] J. Ma J. Xing, J. Xiong and N. Sun. Memory Based Metadata Server for Cluster File Systems. pp. 287–291, Los Alamitos, CA, USA, 2008. IEEE Computer Society.

[49] K. Smith, L. Seligman and V. Swarup. Everybody share: The challenge of data-sharing systems. *IEEE Computer Magazine*, 41(9):54–61, Sep 2008.

[50] K. Veeraraghavan, J. Flinn, E. B. Nightingale and B. Noble. quFiles: The Right File at the Right Time. In *Proccedings of the 7th conference on File and storage technologies (FAST'09)*, pp. 65–68, Feb 2010.

[51] L. M. Vaquero, L. Rodero-Merino, J. Caceres and M. Lindner. A Break in the Clouds: Towards a Cloud Definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50–55, Jan 2009.

[52] M. Pereira, O. Tatebe, L. Luan and T. Anderson. *Resource Namespace Service Specification*. OGF, 2007. GFD.101.

[53] M. Seltzer and N. Murphy. Hierarchical file systems are dead. In *Proceedings of the 12th conference on Hot topics in operating systems (HotOS'09)*, pp. 1–1, May 2009.

[54] Medical Imaging & Technology Alliance. Dicom homepage. `http://medical.nema.org/`.

[55] Namazu Project. Namazu: a Full-Text Search Engine. `http://www.namazu.org/`.

[56] B. Nowicki. NFS: Network File System Protocol specification. RFC 1094 (Informational), March 1989.

[57] O. Tatebe, Y. Morita, S. Matsuoka, N. Soda and S. Sekiguchi. Grid Datafarm Architecture for Petascale Data Intensive Computing. In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, pp. 102–110, May 2002.

[58] Oracle. Oracle grid engine. `http://www.oracle.com/us/products/tools/oracle-grid-engine-075549.html`.

[59] P. A. Chirita, S. Ghita, W. Nejdl and R. Paiu. Beagle++: Semantically enhanced searching and ranking on the desktop. In *Proceedings of 3rd European Semantic Web Conference 2006 (ESWC2006)*, pp. 348–362, Jun 2006.

[60] R. P. Tyer, P. A. Couch, K. K. V. Dam, I. T. Todorov, R. P. Bruin, T. O. H. White, A. M. Walker, K. F. Austen, M. T. Dove and M. O. Blanchard. Automatic metadata capture and grid computing. In *Proceedings of 5th UK e-Science All Hands Meeting (AHM 2006)*, pp. 381–384, Sep 2006.

[61] S. A. Weil, K. T. Pollack, S. A. Brandt and E. L. Miller. Dynamic Metadata Management for Petabyte-Scale File Systems. In *Proceedings of the ACM/IEEE SC2004 Conference*, pp. 4–15, Nov 2004.

[62] S. Furuhashi. The MessagePack Project. `http://msgpack.org/`.

[63] S. Takada. Protein Folding Simulation With Solvent-Induced Force Field: Folding Pathway Ensenble of Three-Helix-Bundle Proteins. *Proteins: Structure, Function and Genetics*, pp. 85–98, 2001.

[64] S. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long and C. Maltzahn. Ceph: A Scalable, High-Performance Distributed File System. In *Proceedings of the 7th Conference on Operating Systems Design and Implementation (OSDI '06)*, pp. 307–320, Nov 2006.

[65] T. D. Thanh, S. Mohan, E. Choi, S. Kim and P. Kim. A Taxonomy and Survey on Distributed File Systems. pp. 144–149. IEEE Computer Society, 2008.

[66] W. Liao, X. Shen and A. N. Choudhary. Meta-data Management System for High-Performance Large-Scale Scientific Data Access. In *Proceedings of the 7th*

International Conference on High Performance Computing (HiPC '00), pp. 293–300, 2000.

[67] 蟻川 浩, 増田 慎吾, 古田 忠臣, 金 文珍, 朴 聖俊, 高田 彰二, 藤川 和利, 砂原秀樹. 対話的動作を考慮したタンパク質立体構造予測システム. 46(SIG 12 (ACS 11)):407–419, Aug 2005.

[68] 金田行雄, 笹井理生 (監修) 笹井理生 (編). **計算科学講座 超多自由度系の新しい科学**. 共立出版, 2010.

[69] 建部 修, 森田 洋平, 松岡 聡, 関口 智嗣, 曽田 哲之. 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**, 43(SIG 6):184–195, Sep 2002.

[70] 古橋貞之. kumofs. `http://kumofs.sourceforge.net/`.

[71] 小西史一. 巨大データの扱いと解析. **情報処理**, 50(9):845–852, Sep 2009.

[72] 小野 功, 水口 尚亮, 中島 直敏, 小野 典彦, 中田 秀基, 松岡 聡, 関口 智嗣, 楯真一. Ninf-1/Ninf-G を用いた NMR 蛋白質立体構造決定のための遺伝アルゴリズムのグリッド化. **情報処理学会論文誌 コンピューティングシステム**, 46(SIG 12 (ACS 11)):396–406, Aug 2005.

[73] 滝田裕, 多田好克. 全文検索エンジンを利用したファイルシステムの名前空間拡張. **情報処理学会論文誌. コンピューティングシステム**, 47(3):16–26, 2006.

[74] 武宮 博, 首藤 一幸, 田中 良夫, 関口 智嗣. Grid 環境上における気象予報シミュレーションシステムの構築. **情報処理学会論文誌 コンピューティングシステム**, 44(SIG 11 (ACS 3)):23–33, Aug 2003.

[75] 武田 伸吾, 伊達 進, 下條 真司. GSI-SFS: グリッドのためのシングルサインオン機能を有するセキュアファイルシステム. **情報処理学会論文誌 コンピューティングシステム**, 45(4):223–233, Apr 2004.

[76] 平賀弘平, 建部修見. 広域ファイルシステム HGFS のための分散メタデータサーバの実装と性能評価. In **情報処理学会 研究報告** *HPC-126 (SWoPP2010)*, pp. 1–9, Aug 2010.

研究業績

論文誌

- 池部実，猪俣敦夫，藤川和利，砂原秀樹: 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, 情報処理学会 論文誌 Vol.52 No.2 pp.488–506 2011 年 2 月
- 池部実，猪俣敦夫，藤川和利，砂原秀樹: 広域分散環境におけるデータセマンティクスを用いた柔軟なデータアクセスシステムの提案と評価, 日本ソフトウェア科学会 コンピュータソフトウェア 2011 年 5 月 (掲載予定)

国際会議 (査読付)

- Minoru Ikebe, Atsuo Inomata, Kazutoshi Fujikawa, Hideki Sunahara: "MetaFa: Metadata Management Framework for Data Sharing in Data-Intensive Applications", International Symposium on Distributed Computing and Artificial Intelligence 2009, LNCS 5518, pp.665–668, Jun, 2009
- Minoru Ikebe, Atsuo Inomata, Kazutoshi Fujikawa, Hideki Sunahara: "Distributed Data Access/Find System with Metadata for Data-Intensive Computing", Proceedings of the 2008 9th IEEE/ACM International Conference on Grid Computing (Poster Session), pp.361–366, Sep, 2008

国内研究会 (査読付)

- 池部実, 猪俣敦夫, 藤川和利, 砂原秀樹: ”広域分散環境におけるデータ管理のためのメタデータ管理フレームワークの提案と実装”, ソフトウェア科学会 第 10 回 インターネットテクノロジーワークショップ WIT2009, 2009 年 6 月, http://wit.jssst.or.jp/2009/

- 池部実, 藤川和利, 砂原秀樹: ”広域分散ネットワーク環境におけるデータのセマンティクスを用いたデータ管理・アクセス手法の提案”, ソフトウェア科学会 第 8 回 インターネットテクノロジーワークショップ WIT2007, pp.97–104, 2007 年 6 月

国内研究会 (査読なし)

- 池部実, 藤川和利, 砂原秀樹: ”広域分散ネットワーク環境におけるデータのセマンティクスを用いたデータ管理・アクセス手法の提案と評価”, 情報処理学会研究報告 (ハイパフォーマンスコンピューティング), Vol.2007-HPC-111, pp.115–120, 2007 年 8 月

- 池部実, 増田慎吾, 藤川和利, 砂原秀樹: ”動的なデータ発生・移動に対応した一意にアクセス可能なデータグリッドシステムの提案”, 情報処理学会研究報告 (ハイパフォーマンスコンピューティング), Vol.2006-HPC-105, pp.133–138, 2006 年 2 月

共著

- 森川泰揮, 猪俣敦夫, 池部実, 岡本慶大, 宇多仁, 藤川和利, 砂原秀樹: ”クラウドを考慮した Web サーバ負荷に応じた動的リソース割当て機構の提案”, 平成 21 年度 第 8 回 電子情報通信学会 信学技報 (インターネットアーキテクチャ), IEICE-IA2009-103, pp.113–118, 2010 年 3 月

- 岡本慶大, 森川泰揮, 野口悟, 池部実, 猪俣敦夫, 河合栄治, 藤川和利, 砂原秀樹: ”クラウド環境を想定した仮想計算機リソース管理におけるセキュリティフレームワークの提案”, 情報処理学会 コンピュータセキュリティシンポジウム 2009 論文集, Vol.2009, No.11, pp.129–134, 2009 年 10 月

- Hideki Shimada, Minoru Ikebe, Yuki Uranishi, Masayuki Kanbara, Hideki Sunahara, Naokazu Yokoya: ”Design and Implementation of Wireless LAN System for Airship”, International Symposium on Distributed Computing and Artificial Intelligence 2009, LNCS 5518, pp.659–662, Jun, 2009

- 藤樫淳平, 池部実, 洞井晋一, 藤川和利, 砂原秀樹: ”P2P システムにおける実ネットワークトポロジを考慮したオーバレイネットワークトポロジ構築に関する一考察”, 情報処理学会 マルチメディア，分散，協調とモバイル (DICOMO2008) シンポジウム論文集, pp.925–931, 2008 年 7 月

- 増田慎吾, 池部実, 藤川和利, 砂原秀樹: ”グリッドコンピューティングにおける科学技術計算を支援するワークフローシステム”, 情報処理学会 マルチメディア，分散，協調とモバイル (DICOMO2007) シンポジウム論文集, pp.1065–1073, 2007 年 7 月

- 板谷諭, 池部実, 藤川和利, 砂原秀樹: ”計測内容のシナリオ記述とタスクの集約を用いた分散アクティブ計測基盤”, 情報処理学会研究報告 (分散システム／インターネット運用技術), Vol.2007-DSM-044, pp.47–52, 2007 年 3 月

- Shingo Masuda, Minoru Ikebe, Kazutoshi Fujikawa, Hideki Sunahara: ”A Gridfied Protein Structure Prediction System Rokky-G and its Implementation Issues”, STUDIES IN HEALTH TECHNOLOGY AND INFOMATICS 120, pp.158–166, Jun, 2006