社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2009－ARC－182 (24)
2009－HPC－119 (24)
2009／2／28

広域ファイルシステムにおける分散メタデータサーバの検討

平賀弘平[†1] 建部修見[†2]

本稿では，広域環境で動作する分散ファイルシステムのメタデータ管理を目的とした，分散メタデータサーバを提案する．高遅延ネットワークを含む環境では，クラスタ内での一般的なメタデータ管理手法では，リクエスト応答時間が増大する．そこで，各拠点にメタデータサーバを分散配置することで，高遅延ネットワーク間の通信をバックグラウンドで行い，さらに一貫性保証をある程度緩めることでリクエスト応答時間の削減を行う．広域分散計算機環境である InTrigger を利用し評価実験を行った結果，高遅延ネットワークをまたがる拠点間でのメタデータ管理において，書き込み応答時間が約 262μsec，読み込み応答時間が約 130μsec だった．評価実験により，提案システムが拠点間の RTT に依らないリクエスト応答性能を示すことを確認した．

Distributed Metadata management system for wide area filesystem

Kohei Hiraga[†1] and Osamu Tatebe[†2]

This paper proposes a distributed metadata server architecture for a wide-area distributed file system. Distributed metadata management in a cluster environment causes a long response time in such a wide-area network environment due to consistency management among metadata servers. The proposed architecture solves this problem by relaxing the consistency semantics among metadata servers located in distant places. Performance evaluations show the comparable performance even in a geographically distributed metadata servers in the InTrigger platform, and achieves 262 μsec and 130 μsec in write and read response time, respectively.

1. はじめに

素粒子物理学，天文学，生命科学などの科学技術分野におけるデータインテンシブコンピューティングの需要の増加に伴い，広域に散在する計算資源，記憶装置，観測装置を効率的に扱うグリッド関連技術の発展と標準化が進んでいる．その一つに分散ファイルシステムがある．分散ファイルシステムは分散するストレージ資源への透過的かつスケーラブルなアクセスを提供する技術で，これまでに様々な研究がなされている．[1)2)3)]

分散ファイルシステムではファイルのパス名や所有者，タイムスタンプ，複製場所，ノードの負荷やストレージ使用状況等，様々なメタデータを管理する．分散ファイルシステムを高速化するために重要なのは，大容量のデータを扱うために 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，このメタデータの操作のための処理時間と，リクエスト応答時間を短縮することが挙げられる．これはメタデータに対する操作がファイルシステム全体のワークロードの半分を占める[4)] ためである．

通常グリッドのように Round Trip Time (RTT) の長いネットワークを内包した環境では，リクエスト応答速度が増大する．地理的に離れた拠点にあるメタデータへ読み書きを行うことや，データ一貫性保証のために広域でロックを取得することは，ネットワークの遅延時間により深刻なリクエスト応答時間の増加につながるため好ましくない．従来クラスタ環境で利用されているネットワークファイルシステムには，NFS, Ceph[5)], Google File System[6)] などがある．これらを広域で運用すると，メタデータ操作の際に高遅延ネットワークを使う通信が発生するため，リクエスト応答時間が増大するという問題がある．

そこで本研究では，グリッド上の広域分散ファイルシステムにおいて，RTT の長い通信路によるリクエスト応答時間の増加を抑止する手法を提案する．提案システムは，ファイルシステムへの操作によって発生するメタデータの読み込みや書き込みを，グリッドの各拠点に分散配置したメタデータを管理するノード (以後メタデータサーバ) が処理する．メタデータは緩い一貫性保障の仕組みによって，グリッドの各拠点間で共有されるため，メタデータの一貫性保証やロックの取得のために，拠点間の通信を行うことなくクライアントの要求を処理することができる．

†1 筑波大学第三学群情報学類
College of Information Science, Third Cluster of Colleges, University of Tsukuba
†2 筑波大学大学院システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba

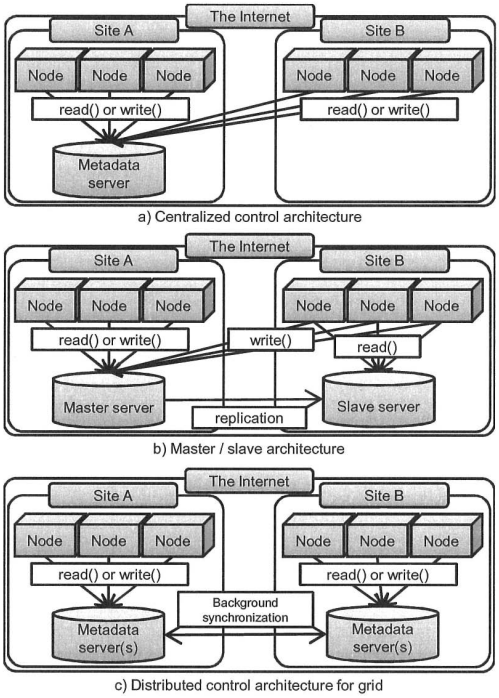

図 1 メタデータ管理方式の違いと広域環境への適用例
a) 集中管理方式 b)Master/slave 方式 c) 本提案方式

2. 分散メタデータサーバの設計

2.1 概　　要

本研究では，分散ファイルシステムは基本的に

- ストレージノード
- メタデータ管理層
- クライアントノード

から構成されると想定する．提案システムの位置づけとしては，このような構成からなる分散ファイルシステムのメタデータ管理層に置き換わるものとする．

提案システムが扱うメタデータとは，ファイルやディレクトリのパス名，所有者，アクセス制御リスト，タイムスタンプ等のファイルに対するメタデータや，ストレージ資源を持つノードのアドレス，負荷状況，ストレージ使用状況，ファイルの複製位置等の分散ファイルシステムを構築するために共有しなければならない情報，その他広域で共有しなければならない設定情報などが考えられる．メタデータの個々のデータサイズは数 KB 程度の大きさで，分散ファイルシステムのコンポーネントを含む多数のクライアントから頻繁に読み書きされると想定する．また，本研究は物理的に離れた複数の拠点で構成されるグリッド環境を対象とする．

既存の分散ファイルシステムが採用している典型的なメタデータ管理方式を広域に適用した場合と，本提案手法との違いを**図 1** に示す．SiteA，SiteB はそれぞれ地理的に分散した異なる組織を表し，複数のノードを保有する．拠点内は低遅延なネットワーク，拠点間は広域の高遅延なネットワークで接続されている．

集中型メタデータサーバを広域で運用した場合，メタデータサーバを保有していない拠点のノードからのメタデータアクセス要求は，すべて広域ネットワークを通過することになる．

Master/slave 型の分散管理方式を行う場合，広域では Slave サーバを各拠点内に配置することで，メタデータ読み込み要求に対する負荷分散と広域ネットワーク間の通信の排除が可能になる．書き込み要求はメタデータの一貫性保証のために Master サーバと通信を行う必要がある．より厳密なデータ一貫性，データの永続性のために Two-Phase Commit を用いると，1 回の書き込み要求を処理するために広域ネットワーク間の通信を複数回行う必要がある．Quorum-like なプロトコル[7] を用いる場合，読み込み要求を行った場合でも広域ネットワーク間の通信が発生する場合がある．

拠点間の通信回数を減らす必要がある．提案システムでは，メタデータサーバを各拠点に一台以上分散配置し，ノードから発行された書き込み，読み込み要求の両方に，拠点内のメタデータサーバが答える．拠点間はある程度緩い一貫性モデルで，バックグラウンドで非同期的にメタデータの交換を行う．

広域環境にデータインテンシブコンピューティングでは，地理的に離れた場所での更新はしばらくたってから反映されても問題ないようなケースが多い．Avian Flu Grid[8] では，鳥インフルエンザ等の流行病や他の新興感染症の感染メカニズムや薬物耐性の解明のために，世界規模に分散したグリッド環境でデータの共有と解析を行っている．ある拠点での解析結果のデータは，直ちに他の拠点で必要になるわけではなく，数時間あるいは数日後に利用可能になっていれば十分であるケースがほとんどである．この要件から，厳密な一貫性を保証することにより，広域でのメタデータアクセスの応答時間が全体的に遅くなることを避ける設計を採用する．

2.2 一貫性モデル

ここでは，分散メタデータサーバの一貫性モデルについて述べる．メタデータサーバを分散化させる時の問題点として，

- システムの可用性
- 強い一貫性保証
- ネットワーク分割への対応

この 3 項目のうち，2 項目までしか達成することができないという制約がある．[9] 提案システムでは，このトレードオフの選択を拠点内と拠点間の 2 層の領域に分割して考える．拠点内はネットワーク分割への対応と強い一貫性保証を優先し，拠点間はネットワーク分割への対応とシステムの可用性を選ぶ．分散サーバモデルにおいて，ネットワーク分割への対応は避けら

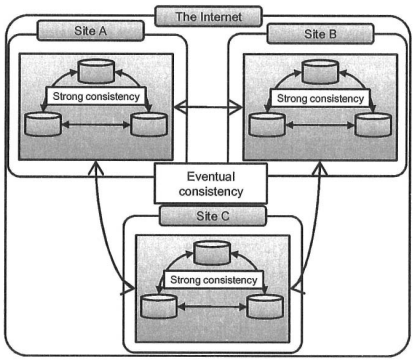

図 2 ハイブリッド型一貫性モデル

れないものとし，拠点間はある程度緩めた一貫性保証をすることでリクエスト応答時間を短縮する．拠点内は強い一貫性を保証しつつ，障害時はその拠点内のみ可用性が損なわれる．

本提案手法では**図 2** に示すように，拠点内は Strong consistent，拠点間は Eventual consistency[10] を保証する．Strong consistent は，あるクライアントの書き込みが完了したら，続くすべての読み込みは最新のデータを得られる事を保証する．Eventual consistency は，クライアントは常に最新のデータにアクセスできるとは限らず，最終的には最新のデータにアクセスできるようになる事を保証する．

拠点間でのメタデータの交換は，バックグラウンドで非同期的に行うことを許している．これにより，クライアントからの読み込み，書き込み要求に，広域ネットワークを介す処理を行わずに応答を返すことができる．また広域ネットワークに障害が起こった場合や，いくつかの拠点が応答しなくなった場合でも，メタデータサーバの可用性を維持することができる．拠点間通信の障害が復旧したら，それぞれのメタデータサーバはお互いの変更点を交換し合い，最終的にはすべてのメタデータサーバが最新のメタデータを共有する．

複数クライアントによる同時書き込みやネットワーク分割，メタデータサーバの障害で，同じメタデータへの変更が衝突する場合がある．提案システムでは Vector clocks[11] を用いて，拠点間で同じメタデータの内容が食い違う場合，衝突が起こったのか，まだ変更が伝わってきていないだけなのかを判断する．メタデータの衝突が起こった場合，同じメタデータに対して 2 種類以上違うバージョンが現れることがある．クライアントはメタデータの衝突を解決するか，もしくは他拠点で行われたメタデータへの変更をひとまず無視して，自拠点のメタデータのみに対して読み書きをすることができる．読み込み要求時は衝突が起こったすべてのメタデータを受け取るか，自拠点のメタデータのみをうけとるか指定する．書き込み要求時には，どのバージョンのメタデータに対して書き込みを行うのか指定する．自拠点のメタデータのバージョンを指定すれば，衝突を解消せずにメタデータを更新する．衝突の起こっているすべてのバージョンのメタデータを指定すれば，メタデータの衝突を解決することができる．

分散ファイルシステムをこのような一貫性モデルの下で構築する場合，アプリケーションのファイルシステムへの要求仕様（POSIX 準拠等）によっては拠点間で分散ロックを行わなければ実現が難しい可能性がある．その場合 POSIX 仕様の拡張を行うか，一貫性モデルを緩くすることで，本提案システムの一貫性を生かした分散ファイルシステムを実現できる．

2.3 メタデータサーバの分散配置

クライアント-メタデータサーバ間のリクエストにおいて高遅延ネットワークを通る通信を行うと，リクエスト応答時間が増大する．本提案システムでは，メタデータサーバを各拠点に分散配置し，クライアントは自拠点のメタデータサーバに対してリクエストを発行することでこの問題に対処する．

クライアントからメタデータ書き込みリクエストが発行されると，メタデータサーバは他拠点のメタデータサーバに対して，リクエストをバックグラウンドで転送する．そのため，メタデータサーバは他の全てのメタデータサーバとコネクションを維持しなければならない．

メタデータサーバが他のメタデータサーバの存在を知るために，メタデータサーバの参加・脱退履歴情報を gossip-based protocol[12] で交換する．交換した履歴情報から，現在所属しているメタデータサーバのホスト名とポート番号を取得し，コネクションを維持する．メタデータサーバの参加・脱退は，管理用コマンドラインツールを使ったマニュアル操作で明示的に行われる．グリッドに新たな拠点がオープンした場合には，その拠点に動的にメタデータサーバを追加することができる．

2.4 分散メタデータサーバのインターフェース

ファイルシステムのメタデータを管理するための，汎用的なインターフェースを設計する．本提案システムでは，シンプルな key と value からなる連想配列機能を提供する．これは，メタデータ管理のための最もプリミティブなインターフェースである．より複雑なメタデータの管理を行いたい場合は，クライアント側で可用性，信頼性，一貫性，リクエスト応答時間のトレードオフを選択し，それらを保証する必要がある．

本提案システムは以下のクライアント用インターフェースを提供する．

- connect(host, port)
  指定した host 名と port 番号でメタデータサーバに接続する．
- close()

メタデータサーバから切断する．

- [list of value, context] = gread(key)
  指定した key に対応する value と context を読み込む．value が衝突している場合，異なるバージョンの value が複数返される．context は，読み込んだ value のバージョン情報を含む．
- gwrite(key, value, context)
  value を key の位置に書き込む．指定した context によって，value の妥当性を検証し，上書き，衝突，失敗の判断をすることができる．衝突した場合，既存の value と新しい value の両方が保存される．
- value = read(key)
  gread() の拠点内版．指定した key に対応する value のうち，拠点内で一番最近に書き込まれた value を読み込む．
- write(key, value)
  gwrite() の拠点内版．value を key の位置に配置し書き込む．すでに key に対応する value が書き込まれていた場合，それが拠点内からの書き込みである場合は上書きされる．そうでない場合は上書きせず，既存の value と新しい value の両方が保存される．

拠点全域で共有すべきメタデータは，gwrite() と gread() を使って読み書きする．また，拠点内でのみ管理したいメタデータについては，write() と read() を使うことで，一般的な連想配列と同じように読み書きすることができる．write() で書き込まれたデータは，gread() を使うことによって，別拠点で読み込むことができる．

3. 評　　価

3.1 実験環境

実験には，InTrigger[13] を利用した．InTrigger は日本国内の様々な教育・研究機関にクラスタを設置した広域分散計算機環境である．InTrigger の各拠点は物理的に離れた場所にあるため，拠点間のネットワークは高遅延である．この実験では，InTrigger の各拠点を，他のユーザと共用して実験を行っているため，実験結果には誤差が含まれている可能性があるが，他のユーザが利用していないことを監視しながら実験を行った為，影響は少ないと考えられる．実験に使用した InTrigger の拠点とハードウェアスペックを**表 1** に示す．拠点間の RTT は約 55msec だった．

3.2 ベンチマーク

提案システムの各 API のリクエスト応答時間を測定するため，実験用ベンチマークを作成した．gwrite(), gread(), write(), read() の各 API を呼び出しリクエスト応答時間を測定する．また比較として，図 1 に示す Master/slave 型メタデータサーバを実装し，クライアント API として put(key, val), get(key), out(key) を準備した．put() は key と value を書き込む．get() は key を読み込む．out(key) は key の削除を行う．put() と out() は，厳密な一貫性保証のため，Master サーバ対して行われる．Master サーバは受け取ったリクエストを非同期的に Slave サーバに転送する．ベンチマークプログラムは，"00000001"～"00010000" の 8 バイトの文字列を key，"abcdef...." という適当な 50 バイトの文字列を value として，それぞれの API を 10000 回ずつ呼び出し平均時間を測定する．あらかじめ 1 度ベンチマークプログラムを動かし，メタデータサーバにデータが入っている状態で計測を行う．純粋にネットワークの遅延時間と，メタデータサーバで処理にかかった時間を計測するため，クライアント側での処理時間は計測時間から除いてある．

リクエストの送信から応答を受け取るまでの処理を，

- network
- deserialization
- vector clocks
- storage
- serialization
- replication
- misc

という単位に分割し，経過時間の内訳を計測した．network は，クライアント (ベンチマークプログラム) とメタデータサーバ間の，リクエスト送信とレスポンス受信にかかるネットワーク往復時間を表す．deserialization はバイト列をリクエストデータに変換する時間．vector clocks は緩い一貫性制御に関わる時間．storage はデータ書き込み，読み込みにかかる時間で，書き込むデータをバイト列に変換する時間，読み出したバイト列からデータに変換する時間等も含んでいる．serialization は，レスポンスデータをバイト列に変換する時間を示す．replication はメタデータサーバ間の通信準備にかかる時間を示す．misc は (クライアント側経過時間-misc 以外のサーバ側経過時間) を示す．

3.3 拠点内での各 API のリクエスト応答時間

mirai 拠点内での各 API のリクエスト応答時間を計測し，Master/slave 型メタデータサーバと提案システムを比較した．実験には，InTrigger の mirai 拠点を使用し，拠点内の 2 ノードにメタデータサーバを配

表 1　実験に使用した InTrigger の拠点とスペック

mirai	
設置機関	はこだて未来大学
domain	intrigger.jp
CPU	Xeon E5410 2.33GHz
Memory	16GB
OS	Linux 2.6.18-6-amd64 SMP
Compiler	GCC version 4.1.2
kyutech	
設置機関	九州工業大学
domain	isc.kyutech.ac.jp
CPU	Xeon E5410 2.33GHz
Memory	32GB
OS	Linux 2.6.18-6-amd64 SMP
Compiler	GCC version 4.1.2

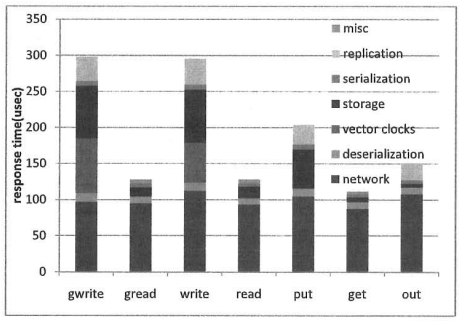

図 3 mirai 拠点内でのリクエスト応答時間

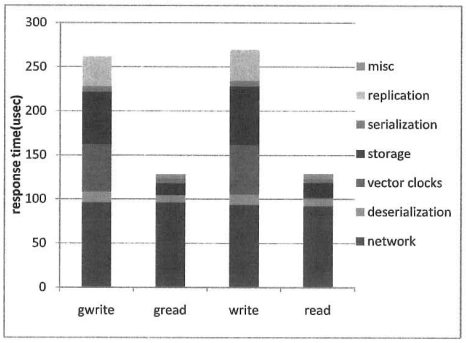

図 4 mirai-kytech 拠点間でのリクエスト応答時間 (提案手法)

置した．ベンチマークプログラムは拠点内の他のノードで動かし，メタデータサーバに対してリクエストを送信した．

実験の結果を**図 3** に示す．network はどのリクエストも約 100μsec かかっている．deserialization，serialization の時間もほぼ一緒である．gwrite()，write()，put() の書き込み系リクエストの storage の方が，gread()，read()，get() の読み込み系リクエストより処理時間が長い．vector clocks の処理のオーバーヘッドが，gwrite()，write() リクエストの約 27〜28 %を占めている．これは一度 DBM から key に対応する vector clocks を読み込む処理と，vector clocks の比較処理，vector clocks 作成のための mutex ロック処理がオーバーヘッドになっていると考えられる．

3.4 拠点間での各 API のリクエスト応答時間

mirai-kyutech 拠点間での，各 API のリクエスト応答時間を測定し，ネットワーク遅延時間のリクエスト応答時間への影響を確かめた．kyutech 拠点と mirai 拠点にそれぞれ 1 台ずつメタデータサーバを配置した．ベンチマークプログラムは mirai 拠点に配置した．本提案システムの評価結果を**図 4** に示す．提案システムは mirai 拠点内で計測した結果拠点内でメタデータサーバを動かした結果とほぼ同じリクエスト応答時間になった．読み込み，書き込みに関わらず，拠点間でメタデータを共有した場合でも，ネットワーク遅延時間による影響は出ていないことが確かめられた．**図 5** は，Master/slave 型，集中型で拠点間でメタデータを共有した場合を示している．kyutech 拠点にあるメタデータサーバを Master サーバとみなし，put と out は Master に，get は mirai 拠点内の Slave に対してリクエストを発行した．remote_get は集中型メタデータサーバを想定し，get を拠点外の Master サーバに発行した場合の結果である．拠点外のメタデータサーバにリクエストを送信する put，get，remote_get は，拠点間の高遅延ネットワークの影響でリクエスト応答時間が増加している．

4. 関連研究

Coda[1]，Antiquity[14]，Dynamo[12] は，本研究と同

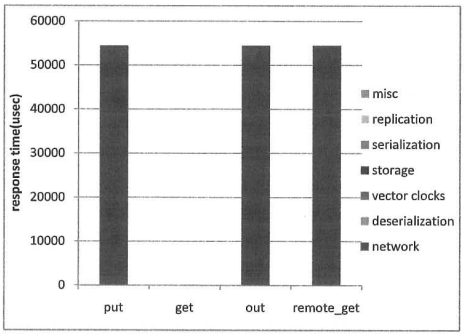

図 5 mirai-kytech 拠点間でのリクエスト応答時間 (Master/slave，集中型)

様に緩い一貫性モデルを採用した分散システムである．

Coda はメタデータとファイル両方をクライアントにキャッシュすることで，ネットワーク分断の際にも可用性を損なわない分散ファイルシステムである．キャッシュを利用することで，クライアントとサーバ間の通信回数を削減している．書き込みとキャッシュミスした場合の読み込みの際には，存在する全てのサーバと通信を行う必要がある．

Antiquity は広域環境を想定したストレージシステムで，ノードの参加・脱退が多く不安定な環境下でも，Quorum-like なプロトコルによってデータの品質と永続性を担保することができる．メタデータは単一のサーバが管理している．

Dynamo は high available な key-value のハッシュストレージシステムである．key のハッシュ値を元にストレージノードを決定し，データを分散管理する．広域で運用する場合は拠点外にあるストレージノードが選択される可能性もあり，その場合リクエストの応答時間が長くなると考えられる．

5. おわりに

本稿では，広域環境で動作する分散ファイルシステムのメタデータ管理を目的とした，分散メタデータサー

バを提案した．提案システムは，各拠点にメタデータサーバを分散配置することで，高遅延ネットワーク間の通信をバックグラウンドで行う．さらに緩い一貫性保証を行うことでリクエスト応答時間の削減を行った．広域分散計算機環境である InTrigger を利用し評価実験を行った．高遅延ネットワークをまたがる拠点間でのメタデータ管理において，書き込み応答時間が約 262μsec，読み込み応答時間が約 130μsec だった．このことから，提案システムが拠点間の RTT に依らないリクエスト応答性能を示すことを確認した．

今後の課題として，次の点が挙げられる．

- メタデータの永続性保証
  サーバのディスク障害や操作ミスによるメタデータファイルの消失が起こると，メタデータの永続性が保証されない．現状ではメタデータを複製する前に，クライアントに応答を返すので，メタデータの消失を避けるため，複数個所にメタデータを複製してから応答を返すべきである．電源障害や自然災害による拠点全体の障害に対処するためには，他拠点へ複製するの望ましいが，他拠点へ確実に複製されたことを確認するには拠点間通信を行わなければならないため，リクエスト応答時間が増大する．そのため他拠点への複製は行うが応答確認はせず，拠点内への複製にのみ応答確認をすることで，単一サーバ障害への永続性を保証する複製の仕組みが必要だと考える．
- メタデータサーバ性能の動的なスケールアウト
  現状ではサーバの動的な追加は可能となっている．クライアントの読み込み要求は，各拠点のメタデータサーバ対してのみ向かうため，性能はスケールアウトする．しかし書き込み要求はすべてのメタデータサーバに転送されるため，スケールアウトしない．この問題に対処するため，各拠点でメタデータの key-value のセットを全部保持しつつ，拠点ごとに key によってメタデータの管理を担当するメタデータサーバを振り分ける，という方法がある．

謝辞 本研究の一部は，情報爆発時代に向けた新しい IT 基盤技術の研究，文部科学省科学研究費補助金「特定領域研究」(課題番号 19024009) および文部科学省次世代 IT 基盤構築のための研究開発「e-サイエンス実現のためのシステム統合・連携ソフトウェアの研究開発」，研究コミュニティ形成のための資源連携技術に関する研究 (データ共有技術に関する研究) による．

参 考 文 献

1) Satyanarayanan, M.: Coda: A Highly available File System for a Distributed Workstation Environment, *IEEE Transactions on Computers*, Vol.39, pp.447–459 (1990).
2) 亀嶋徳哉，角川裕次，山下雅史：世界規模分散ファイルシステム SKINNY，情報処理学会研究報告. [システムソフトウェアとオペレーティング・システム]， Vol.95, No.79, pp.1–8 (1995).
3) 修見建部，哲之曽田：広域分散ファイルシステム Gfarm v2 の実装と評価 (グリッド I)，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, No.122, pp.7–12 (20071207).
4) Roselli, D., Lorch, J.R. and Anderson, T.E.: A Comparison of File System Workloads, *In Proceedings of the 2000 USENIX Annual Technical Conference*, pp.41–54 (2000).
5) Weil, S., Brandt, S. A., Miller, E. L., Long, D. D. E. and Maltzahn, C.: Ceph: A Scalable, High-Performance Distributed File System, *Proceedings of the 7th Conference on Operating Systems Design and Implementation (OSDI '06)*, Vol.7, USENIX (2006).
6) Ghemawat, S., Gobioff, H. and Leung, S.-T.: The Google File System, *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pp.20–43 (2003).
7) Alvisi, L. and Dahlin, M.: Byzantine quorum systems, *Distributed Computing*, Vol. 11, pp. 569–578 (1998).
8) : Avian Flu Grid. http://avianflugrid.pragma-grid.net/.
9) Gilbert, S. and Lynch, N.: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services, *SIGACT News*, Vol.33, No.2, pp.51–59 (2002).
10) Vogels, W.: Werner Vogels "Eventually Consistent" (2007). http://www.allthingsdistributed .com/2007/12/eventually_consistent.html.
11) Lamport, L.: Time, clocks, and the ordering of events in a distributed system, *Commun. ACM*, Vol.21, No.7, pp.558–565 (1978).
12) DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P. and Vogels, W.: Dynamo: amazon's highly available key-value store, *SIGOPS Oper. Syst. Rev.*, Vol.41, No.6, pp.205–220 (2007).
13) 秀雄斎藤，良和鴨志田，省吾澤井，健　弘中，慧　高橋，岳史関谷，楠　頓，剛志柴田，大作横山，健次朗田浦：InTrigger：柔軟な構成変化を考慮した多拠点に渡る分散計算機環境 (HPC-14：分散処理)，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, No.80, pp. 237–242 (20070801).
14) Weatherspoon, H., Eaton, P., Chun, B.-G. and Kubiatowicz, J.: Antiquity: exploiting a secure log for wide-area distributed storage, *SIGOPS Oper. Syst. Rev.*, Vol. 41, No. 3, pp. 371–384 (2007).