Vol.2010-HPC-126 No.29
2010/8/4

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

平賀弘平[†1]　建部修見[†1]

広域環境下の複数の組織間で，大容量のデータを効率よく共有が年々増加している．地理的に分散した PC クラスタ間でデータを効率的に共有するためには，広域ファイルシステムが有用であるが，高遅延通信路の影響により，ファイルシステムのメタデータ操作がシステムのボトルネックになるという問題がある．本稿では，広域環境下での効率的な大規模データ処理を目的とした，広域ファイルシステム HGFS を提案し，2-level Consistency と，Soft Client-centric Consistency という 2 種類のファイルシステムセマンティクスについて述べる．HGFS の分散メタデータサーバである HGMDS の実装を行い，広域分散計算機環境である InTrigger を利用し性能評価を行った結果，クライアントリクエスト応答時間は書き込みが 145μsec，読み込みが 107μsec であった．また，複数拠点を利用した，HGMDS のリクエスト処理性能の評価を行い，HGMDS が広域環境においても高いスループットであることを確認した．

Distributed Metadata Management System for Global File System HGFS

Kohei Hiraga[†1] and Osamu Tatebe[†1]

Efficient sharing of large-scale data has been required among multiple organizations in wide area. Global file system is useful to easily share data, but it has a problem of long response time when accessing from a distant location. This paper presents a implementation and performance evaluation of distributed metadata server for HGFS global file system. It provides 2-level Consistency and Soft Client-centric Consistency. Performance evaluations show the comparable performance even in a geographically distributed metadata servers named HGMDS in the InTrigger platform, and achieves 145μsec and 107μsec in write and read response time, respectively. Moreover, HGMDS shows a high perfromance by the performance evaluation of throuput using two or more sites in spite of the large area environment.

1. はじめに

近年コンピュータの性能向上や，価格低下により，大規模なデータセンターを構築し，多数のコモディティコンピュータを並べて，大量のデータを解析する技術が確立されてきた．それに伴い，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分野では，年々扱うデータ量が増加の一途を辿っている．また，これらの分野では，広域に散財する計算資源，記憶装置，観測装置を効率的に扱う，複数の拠点による広域分散データ解析の要求が高まっている．

広域分散データ解析の例として，Avian Flu Grid プロジェクト[1)] では，鳥インフルエンザ等の感染メカニズム，薬物耐性の解明のために，世界規模に分散した組織間で大量のデータ共有と分析を分担している．

また，天文学の分野では，地理的に離れた場所にある観測機器が生成する膨大なデータを，いかに効率よくデータ解析できるかが重要である．これらの大量の天文データを，広域通信路を含む環境で，効率的にデータ処理を行う手法の提案がなされている[2)]．

このように，地理的に分散した複数の拠点間で，大量のデータを共有するためには，Gfarm[3)] などの広域ファイルシステムの利用が注目されている．

しかしながら，既存のファイルシステムが持つ問題点として，広域環境では Round Trip Time (RTT) の長い通信路の影響で，ファイルシステムのメタデータへの操作の応答性が低下し，システムのボトルネックとなる問題がある．例えば，日米間の RTT は 130~ 250ms，日欧間は 250~ 300ms なので，広域ファイルシステム操作に対するリクエスト応答性能は，高遅延ネットワークを介した通信が何往復するかによって，大部分が決定するからである．

我々はこれまでに，RTT の長い通信路に依存しない，Key-Value の連想配列をベースにメタデータを管理する，広域ファイルシステムのための分散メタデータサーバについて研究を行って来た[4)]．特に，Key-Value でツリー構造であるファイルシステムの inode 構造体を分散管理するための手法の提案と，システムの設計について研究を行って来た[5)]．本稿では，広域ファイルシステムに適したファイルシステムセマンティクスを明確にし，これまでの提案を元に，広域ファイルシステム HGFS の分散メタデータサーバ，HGMDS の実装と性能評価を行い，広域環境においてファイルシステムのメタデータ管理手法としての有効性を確

†1 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

認する．これにより，現在の分散ファイルシステムで問題となっている，広域ネットワークの遅延によるファイルシステムのメタデータアクセスの応答性能低下の問題を解決する．

2. HGFS のファイルシステムセマンティクス

本節では，HGFS の提供するファイルシステムセマンティクスについて述べる．広域環境下で効率的なデータインテンシブコンピューティングを行う要求がある．既存の POSIX 準拠のファイルシステムや，ACID を満たすストレージシステムが提供する，Strong Consistency の広域環境での実現には，ステートフルなコンポーネントの集中化，もしくは，一貫性維持のための高遅延通信路をまたいだ通信が必要になる為，多大なパフォーマンスの犠牲は避けられない．

HGFS は，広域環境下での効率的にデータ共有と解析が行えるプラットフォームを目指すため，Strong Consistency の実現よりもむしろ，現在の広域分散計算機環境のユースケースにマッチした，2-level Consistency と，複数拠点でのデータ連携を可能にする，Soft Client-centric Consistency という，2 つのファイルシステムセマンティクスを提供する．

2.1 2-level Consistency

2-level Consistency は，拠点間の効率的なデータ共有の実現と，既存のデータ解析システムとの親和性を重視した一貫性モデルである．

広域分散計算機環境の典型的なユースケースとして，データ解析処理等の計算自体は，ネットワークが低遅延で，利便性の良い 1 拠点内の計算機のみを使って実行し，計算のための入力データと，計算結果を広域で共有する形態が挙げられる．このようなユースケースの場合，通常の Strong Consistency である POSIX 準拠のファイルシステムと同じファイルシステムセマンティクスを備えており，既存のデータ解析プログラムが動作することと，計算処理に必要な入力データや，解析結果データが，いずれ全ての拠点から利用可能になることが，ファイルシステムに求められる．

HGFS では，このようなユースケースに答えるため，2-level Consistency という一貫性モデルの下でファイルシステムセマンティクスを設計した．ファイルシステムを用いる既存のプログラムの再利用のため，1 拠点内の複数計算機からのファイルシステムアクセスは，Strong Consistency を保ち，通常の POSIX 準拠ファイルシステムと同じように使用できる．そして，ある拠点で生成された入力データや，解析結果データは，Eventual Consistency[6] で各拠点に伝播し，いずれ利用可能となる．

2.2 Soft Client-centric Consistency

近年，大規模データ解析を行うフレームワークとして，MapReduce[7] をはじめとするワークフローエンジンが盛んに利用されている．大規模分散データ解析を行う方法として，有効なこのフレームワークを，広域環境においても複数拠点を用いて効率的に分散実行する手法が提案なされている[2),8),9)]．

広域環境下でワークフローエンジンを利用可能とするためには，ワークフローエンジンにおけるワーカー計算機が正しく動作するために，タスク開始時に入力ファイルが存在することを保証すればよい．したがって，ワークフローエンジンの，タスクスケジュールを行っているコンポーネントに対して，Client-centric Consistency[10] を満たす，ファイルシステムセマンティクスを保証できればよい．

HGFS では，広域環境の複数拠点を用いた場合の，ファイルシステムセマンティクスとして，Soft Client-centric Consistency を提案する．

例えば，あるクライアントがファイルを作成し，別の拠点のクライアントが同じファイルを読み込むことを保証したい時，クライアント自身がファイルシステム操作を行った後，hgsync 操作を行うことで，別の拠点にファイル作成の通知が正しく伝播したことを必要ならば保証できる．

hgsync 操作は，Soft Client-centric Consistency を保証したい拠点のサーバ識別子を指定できる．もし，指定した拠点で別のクライアントが同じファイル操作を行っていた場合，ファイルシステム操作が重複し，Client-centric Consistency が満たせなくなる場合がある．その時は，hgsync はクライアントに失敗を返す．

3. 広域ファイルシステム HGFS と，分散メタデータサーバの概要

広域ファイルシステム HGFS は，複数の拠点からなる広域分散計算機環境下で，大規模データ処理を行うためのファイルシステムを目指している．

ファイルシステムのファイルやディレクトリに関するメタデータは，inode と呼ばれる構造で管理する．inode は，Key-Value ストアをベースとした HGMDS で管理する．ファイルシステムメタデータ操作では，複数の inode エントリをアトミックに更新する必要があるが，我々はこれまでに，それを可能とする仕組みを提案した[5]．

HGFS は，2-level Consistency と，Soft Client-centric Consistency と呼ぶ 2 種類のセマンティクスを提供し，必要に応じて選択できる．

Vol.2010-HPC-126 No.29
2010/8/4

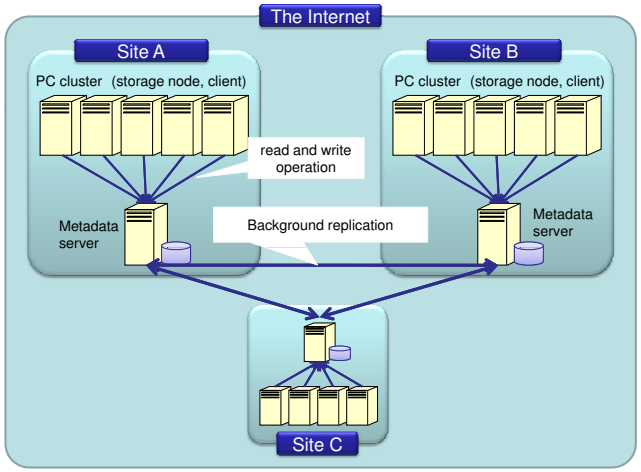

図 1 HGFS の構成．SiteA,SiteB,SiteC は PC クラスタを保有している地理的に分散した異なる拠点を表す．各拠点には HGMDS が分散して配置され，クライアントからのリクエストを受け付ける．

3.1 システム構成

提案システムの構成を**図 1** に示す．広域分散計算機環境の各拠点に HGMDS を分散配置し，各 HGMDS はシステムが管理する全メタデータの複製を保持する．

クライアントは，既存の Key-Value ストアライクなメタデータ操作のための API 呼び出しを行い，inode 構造の Read/Write を行う．クライアントのリクエストは低遅延な LAN を介して処理される．各 HGMDS は全てのメタデータの複製を保持しているので，HGMDS はクライアントに即座に応答を返すことができる．

inode への更新が起こると，HGMDS は他拠点の HGMDS に対して，非同期に更新を伝播し，自動的に同期する．

メタデータサーバはマルチマスタ型の構成を採用し，複数クライアントからの Read/Write 操作の並列性を確保する．

3.2 Vector Clock を用いた inode のバージョン管理

結果整合性を採用している提案システムでは，同時に 2 つ以上のクライアントから inode の更新操作が行われた場合，更新が衝突する可能性がある．Multiple Master 型で厳密な一貫性のための排他制御を実現するには，Write/Read 操作時に，広域に分散するメタデータサーバ間で分散ロックを取得する必要があるが，これを行わない．提案システムでは，分散ロックの代わりに Vector Clock[11] を用いる．Vector Clock は，それぞれのサーバ識別子とサーバローカルなカウンタのペアのベクトルで，半順序が定義される．メタデータサーバの inode の各エントリは，Vector Clock によってバージョニングされる．バージョニングによって，inode の更新操作の順序関係を保存し，順序が定義されなければ，更新操作により衝突を検知する．もし更新操作が衝突したら，一時的にメタデータサーバ間で inode に違いが起こるが，予め定義するマージアルゴリズムに従って，システムが自動的に衝突を解決し，いずれすべてのメタデータサーバ通知される．これにより，結果整合性を守る．

表 1 定義型の説明

記号	説明
k	任意の型の key
v	任意の型の value
cv	他拠点に書き込まれ，更新が衝突している value
c	contex．バージョン情報等が格納されている構造体．クライアントは context の中身を意識する必要はない．
sid	サーバ識別番号
[]	配列

3.3 クライアント用低レベルインタフェース

HGMDS は，以下のクライアントインタフェースを提供する．インターフェースの説明で使用している記号については，**表 1** を参照されたい．

3.3.1 インタフェース

- [v,new_c] = get(k, c)
  k に対応する v を読み込み，c を新たに読み込んだ v のバージョンでアップデートし，new_c を返す．
- [[v1,v2,...],new_c] = get_multi([k1,k2,...],c)
  複数の k を指定して，複数の v を読み込む．c を新たに読み込んだ v のバージョンでアップデートし，new_c を返す．
- [[v1,cv1,cv2,...],new_c] = g_get(k, c)
  k に対応する v と，更新の衝突によって保留されていた値 cv を読み込む．c を新たに読み込んだ v のバージョンでアップデートし，new_c を返す．
- [[[v1,v2,...],[cv1,cv2,...],...],new_c] = g_get_multi([k1,k2,...],c)
  複数の k を指定して，対応する複数の v を読みこむ．同時に，更新の衝突によって保

留されていた複数の値 cv も読み込む．対応する値がない場合は NULL を返す．c を新たに読み込んだ v のバージョンでアップデートし，new_c を返す．

- put(k, v, c)
  指定した k に対応する v を c のバージョンに従って書き込む．c は，クライアントがこれまでに get() や g_get() で読み込んだ k のバージョンが格納されている．書き込みは，まず拠点内の HGMDS で楽観ロックのチェックを行い，問題なければ書き込みが実行され，応答が返る．その後，書き込みは直ちに他拠点の HGMDS に伝播される．
- put_multi([[k1,v1],[k2,v2],. . .],c)
  複数の k と v のペアを c のバージョンに従ってアトミックに書き込む．
- bool = sync(k,sid) 自拠点の HGMDS で直前に行われた k への更新が，サーバ sid に衝突することなく転送された場合，true を返す．衝突した場合，false を返す．
- sync_all(k) 自拠点の HGMDS で直前に行われた k への更新が，全ての他サーバに衝突することなく転送された場合，true を返す．1 箇所でも衝突した場合，false を返す．

HGMDS は，既存の Key-Value ストアライクなクライアント用 API を備えている．ここで上げているすべての API は，通信路のトラブルや，サーバビジーによるシステム停止の対処のため，タイムアウトを設定できる．

4. HGMDS 実装

HGMDS の実装について述べる．分散メタデータサーバの詳細な設計については，文献 5) を参照されたい．

4.1 内部データ構造

HGMDS は，Key-Value ストアの実装である Tokyo Cabinet[12] を用いて実装を行った．HGMDS は内部データ構造として，以下のデータベースを保持している．

- メインデータ DB
  メタデータを格納する主データベース．Hash 型の Key-Value ストアで，key はクライアント指定の key，value はクライアント指定の value とバージョン番号が付与される．
- 衝突データ DB
  衝突データを保留するデータベース．実体は 2 つの Key-Value ストアで構成される．メインデータに既に格納されているデータのバージョンとの比較により，更新衝突が検知され，メインデータ DB への書き込みを保留されたデータが格納される．衝突データは，アトミックな書き込み単位で格納されており，読み込みの際はその単位で読み込まれる．
- 更新ログ
  サーバに到着した更新リクエストを全て保持するファイル．実体は HGMDS の各サーバごとに分割されたファイルに，更新リクエストそのものが格納される．
- 複製完了 ACK 待ちバッファ
  他拠点の MDS に複製を送信し，未だ ACK が帰ってきていない key が格納される．各 MDS ごとにバッファは作成される．各 ACK 待ち key には，WAIT と COLLISION 属性が存在する．WAIT はリモート MDS からのレスポンス待ち状態，COLLISION は，レスポンスは帰ってきたが，複製は衝突したことを表す．
- 因果順序メッセージバッファ
  HGMDS 間で，非同期にやりとりされる更新リクエストの複製メッセージには全て，Vector Clock によるバージョン番号が付与されている．これらのメッセージは，MDS へ実際の書き込み順序によらず，任意の順序で到着する可能性がある，MDS に到着した更新リクエストの複製メッセージが，因果順序[13] を満たしていない場合，メッセージは一旦，因果順序メッセージバッファに保留され，因果順序を満たすまで，配送は保留される．

4.2 更新処理 put_multi()

put_multi() の HGMDS 内での処理の流れを**図 2** に示す．

(1) クライアントが put_multi リクエストを送信する．
(2) これから書き込む key に対応する，衝突データがあれば，読み込みを行う．
(3) これから書き込む key より，順序が古い衝突データがあれば，衝突データを読み込み，これから書き込む key のリストとマージを行う．これから書き込む key のバージョンの前のバージョンが衝突データ DB に保留されている場合，前のバージョンを先に解決しなければメインデータ DB への更新の適用が，因果順序を満たさないからである．
(4) key に対応するメインデータ DB のデータを読み込み，既に格納されているデータを取得する．
(5) メインデータに格納されていたバージョンと，書き込みを行うバージョンを比較し，既に他のクライアントに更新されていないかどうか，楽観的ロックのチェックを行う．既に書き込みが行われていた場合，クライアントに書き込み失敗を返す．
(6) 安全に書き込みが行える状態だった場合，書き込みのためのバージョン番号を，書き

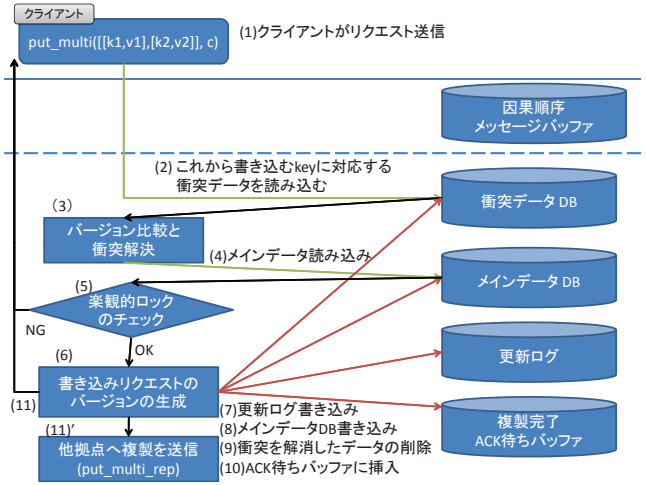

図 2 HGMDS 内の，put_multi リクエストの処理の流れ

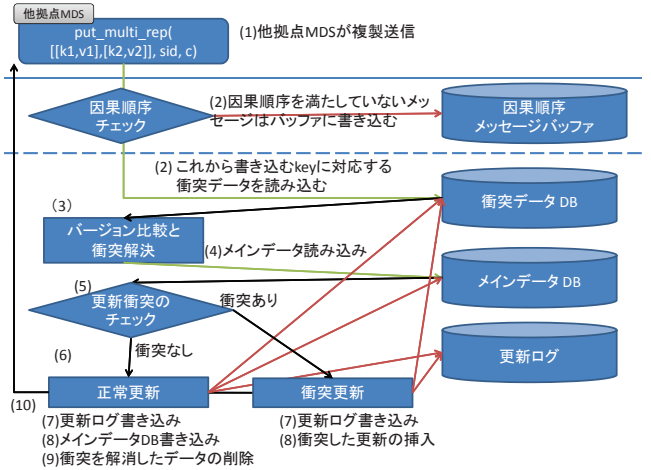

図 3 他拠点から更新が伝播した際に呼ばれる，HGMDS の put_multi_rep リクエストでの処理の流れ

込みをハンドリングするサーバのカウンタを用いて生成する．

(7) 更新ログへ書き込みを行う

(8) メインデータ DB へ書き込みを行う

(9) 衝突を解消したデータがあれば，削除を行う

(10) 複製完了 ACK 待ちバッファに，key を挿入する

(11) クライアントに成功の応答を返し，他拠点へ複製を非同期に送信する．

put リクエストは，put_multi と同様の処理を行う．

4.3 更新の伝播 put_multi_rep()

put_multi_rep() は，他拠点の MDS から呼び出される HGMDS のリモートプロシージャコールで，更新リクエストの複製の受け取り毎に呼ばれる．put_multi_rep() の処理の流れを**図 3** に示す．ほぼ put_multi() と同様の処理を行うが，他拠点 MDS から送信されてきたメッセージは因果順序を満たしている場合にのみ，配送される．また，楽観的ロックのチェックの代わりに，更新衝突のチェックを行い，更新に衝突があった場合は，衝突データ DB に書き込み，通常の書き込みの場合は，メインデータ DB に書き込む．

4.4 読み込み get_multi()/g_get_multi()

get_multi() と g_get_multi() は，どちらも指定の key の value を読み込むインターフェースで，g_get_multi() は，さらにメインデータ DB の内容に加え，衝突データ DB の値も存在すれば読み込む．value と同時にバージョン番号も読み込み，クライアントに送信する．

5. 性能評価

5.1 評価環境

実験には，InTrigger[14] の，北海道大学拠点，東北大学拠点，法政大学拠点，九州工業大学拠点の 4 拠点を利用した．InTrigger は，日本国内の様々な教育・研究機関に設置されたクラスタを，高バンド幅なネットワークで接続した，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．実験に使用した拠点のハードウェアスペックを**表 2** に示す．拠点間の RTT を**表 3** に示す．

5.2 ベンチマーク

提案メタデータサーバの性能評価のため，実験用ベンチマークを作成した．ベンチマークは，提案システムの API である,put_multi()/get_multi()/g_get_multi() を同期的に呼び出し，リクエスト応答時間を測定する．“00000001” のような，8 バイトの文字列を key, “vvvvvv...” という 50 バイトの文字列を value としている．複数のマシンからなるクライアントを使って，サーバの#ops/sec を測定する．

表 2 実験に使用した InTrigger の拠点とスペック

拠点名	ノード名	CPU	Sockets/node	Cores/socket	Memory
北海道大学	huscs	Xeon(R) E5530 2.40GHz	2	4 + HT	24GB
法政大学	hosei	Xeon(R) E5530 2.40GHz	2	4 + HT	24GB
東北大学	tohoku	Xeon(R) E5410 2.33GHz	2	4	32GB
九州工業大学	kyutech	Xeon(R) E5410 2.33GHz	2	4	32GB

表 3 InTrigger 拠点間の Round Trip Time

	北海道大学	東北大学	法政大学	九州工業大学
北海道大学	-			
東北大学	14 ms	-		
法政大学	44.7 ms	21 ms	-	
九州工業大学	61 ms	44 ms	38.3 ms	-

5.3 クライアント応答時間の測定

北海道大学，九州工業大学拠点に提案メタデータサーバを配置し，北海道大学拠点のクライアントから拠点内のメタデータサーバに対して，前述のベンチマークを用いて，それぞれの API を 1000 回ずつ呼び出し，リクエストの送信から応答を受け取るまでの経過時間の平均を測定する．

また比較として，単一メタデータサーバによる集中型メタデータ管理を行った場合の，リクエスト応答時間を測定する．集中管理方式の場合，メタデータアクセスはリモートの拠点に配置されたメタデータサーバにアクセスすることを想定している．事前に ping コマンドで測定した拠点間の RTT は約 61msec であった．

5.3.1 評 価 結 果

評価結果を**図 4** に示す．集中管理方式は，クライアントがリクエストの応答を得るまでに，read と write が共に 61.2msec 必要であった．提案システムでは，応答時間は write 145μsec read 107μsec であった．

メタデータを集中管理する場合，クライアントのリクエスト応答時間は，拠点間の高遅延ネットワークの影響を避けられない為，リクエスト応答時間が増加している．提案システムの場合，クライアントは高遅延ネットワークの影響を受けないリクエストの発行が可能であることを確認できた．

5.4 単一拠点でのメタデータサーバのリクエスト処理性能

北海道大学拠点に提案メタデータサーバを配置し，同一拠点内のクライアントから，並列にリクエストを発行した時の，サーバのリクエスト処理性能を測定する．サーバにリクエストを発行するクライアントを 1 台ずつ追加していき，システムのスループットを測定する．

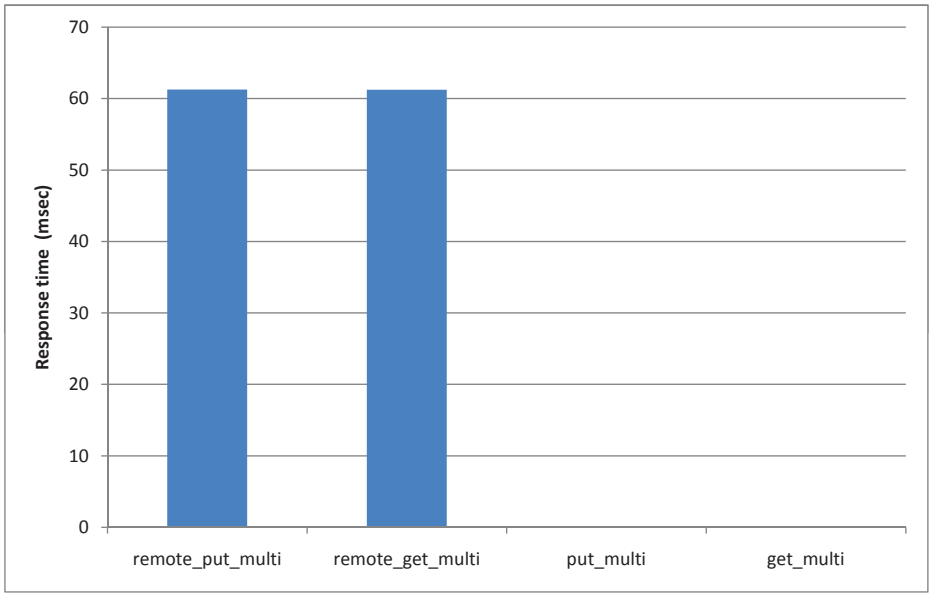

図 4 クライアント応答時間

5.4.1 評 価 結 果

評価結果を**図 5** に示す．読み込みは，15 台のクライアントからの並列リクエスト発行により，最大 101687ops/sec の処理性能であった．書き込みは，10 台のクライアントから並列にリクエストを発行した時が最も性能が高く，34387ops/sec であった．

読み込みは並列性が高く，クライアントの増加でスループットが向上することが確認できた．書き込みは，テーブル全体をロックしている為読み込みリクエストほど並列性は高くないが，複数クライアントからの同時更新に 32000~ 34000ops/sec まで対応できることが分かった．

5.5 複数拠点でのメタデータサーバのリクエスト処理性能

4 拠点に提案メタデータサーバを配置し，データを同期させた状態で，各拠点のクライアントから，拠点内のメタデータサーバにリクエストを発行し，リクエスト処理性能を測定す

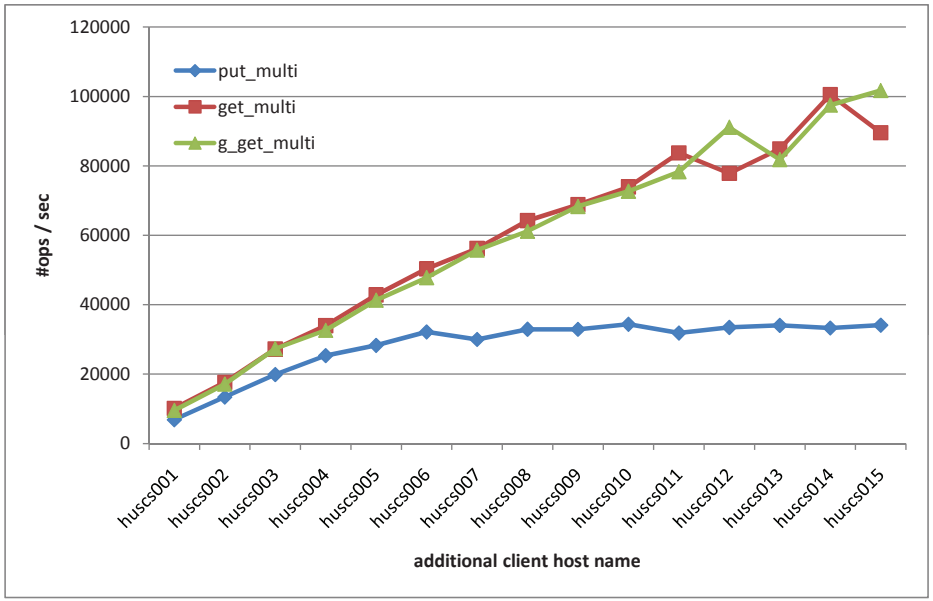

図 5 単一拠点でのメタデータサーバのリクエスト処理性能

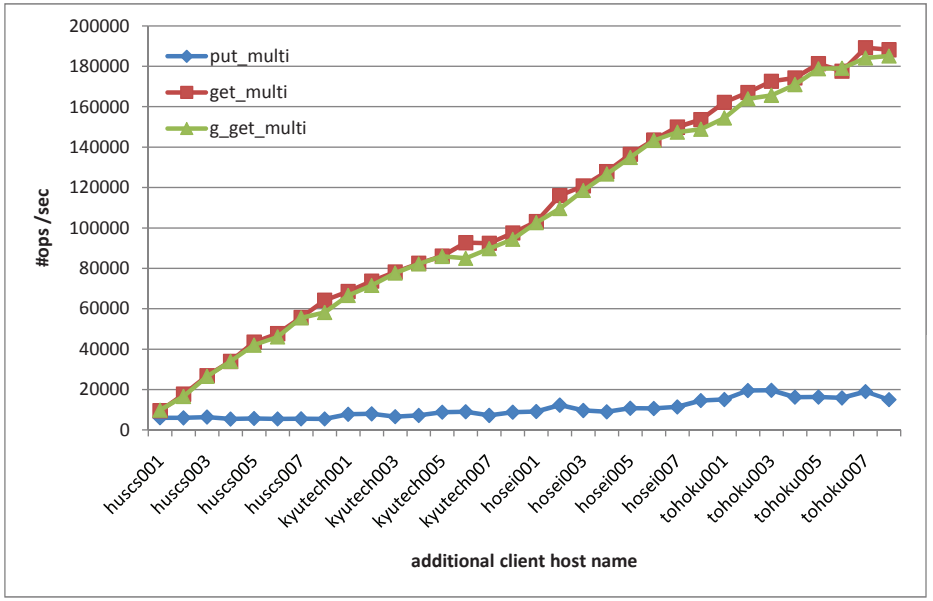

図 6 複数拠点でのメタデータサーバのリクエスト処理性能

る．クライアントはまず拠点内で 1 台ずつ追加していき，閾台数を超えたら次にできるだけ距離的に分散した拠点を選択し，その拠点内でクライアントを 1 台ずつ追加する．クライアントの数は各拠点 8 台ずつで，北海道大学，九州工業大学，法政大学，東北大学の順にクライアントを増加させる．

5.5.1 評 価 結 果

評価結果を**図 6** に示す．読み込みは 4 拠点 32 台のクライアントからの並列リクエスト発行により，最大 188170ops/sec の処理性能であった．書き込みも 4 拠点を使った場合が最も性能が高く，最大 19667ops/sec であった．

読み込みは，複数拠点で同時にリクエストを発行した場合でもスケールアウトすることが確認できた．書き込みは 4 拠点を使った場合，20000ops/sec 程度まではスケールしたが，サーバを 1 拠点にのみ配置した場合よりは性能が低下している．これは，他拠点のサーバに書き込みリクエストをブロードキャストする部分がオーバヘッドになっているとためである．

5.6 メタデータサーバの更新リクエスト処理性能における，利用拠点数の増加の影響

複数拠点に提案メタデータサーバを配置し，複数のクライアントから書き込みリクエストを送信した場合において，書き込みの複製作成がシステム全体の性能へ与える影響を調査する．

メタデータを共有する拠点数が 1 拠点~ 4 拠点の場合のそれぞれで，北海道拠点内のクライアントから，書き込みリクエストである put_multi() を並列に発行し，リクエスト処理性能を測定する．サーバを設置する拠点は，北海道大学，九州工業大学，法政大学，東北大学の順で増やす．

クライアントから直接リクエストを受け付ける北海道拠点内のメタデータサーバは，クライアントのリクエスト処理を行ない，クライアントに通知を返した後，他拠点のサーバに非同期に複製リクエストをブロードキャストする．

5.6.1 評 価 結 果

評価結果を**図 7** に示す．put_multi() の性能は，他サーバへ複製送信を行う必要が無い，北

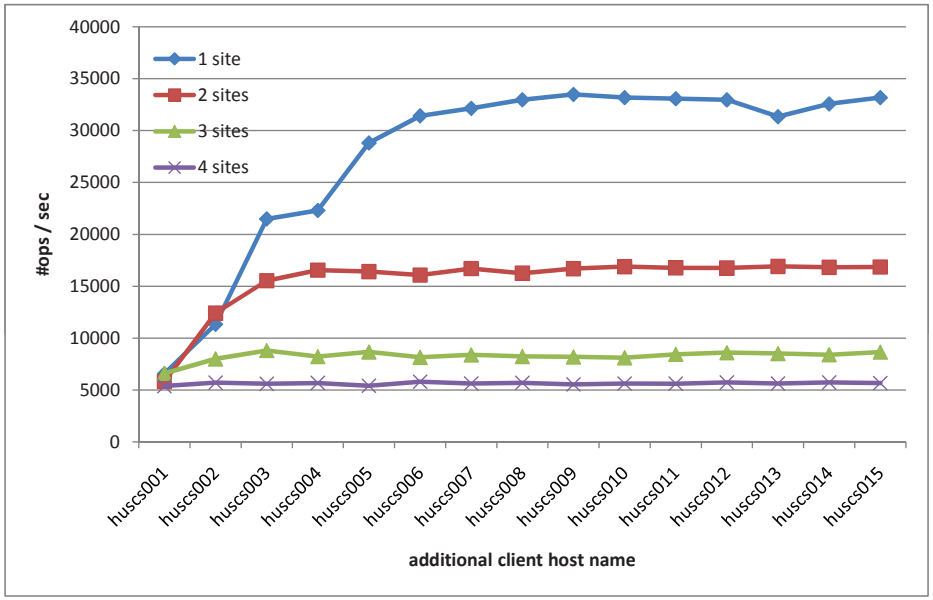

図 7 メタデータを同期する拠点数を変化させた時の，メタデータサーバの更新リクエスト処理性能．

海道大学拠点の 1 拠点の場合が最も性能が高く，その場合の最大スループットは 33489ops/sec であった．リクエスト処理のスループットは，2 拠点，3 拠点，4 拠点と，メタデータを共有する拠点数を増やす度に低下している．2 拠点の場合の最大スループットは 16910ops/sec，3 拠点では 8807ops/sec，4 拠点では 5800ops/sec であった．サーバのリクエスト処理限界に達した後は，クライアント数を増やしてリクエスト送信の並列度を増加させても，サーバのリクエスト処理性能は限界性能のままほぼ変化していない．

評価結果から，サーバの書き込みリクエストのボトルネックは，内部 Key-Value Store への書き込みではなく，書き込みリクエストの複製送信部分にあることが確認できた．現在の実装では，書き込みリクエストの複製送信は，クライアントから書き込みリクエストを受け取ったサーバのブロードキャストでなされているが，これをメタデータサーバでリングを構成し，ブロードキャストではなくリレーで送信すれば，クライアントから書き込みリクエストを受け取るサーバの負荷を低減させられる．また，書き込みリクエストを即座に複製送信せずに，一定時間貯めておくことで，同じ key への連続した書き込み等を圧縮してから，他サーバへ複製転送を行える．さらに根本的に書き込みスループットを向上させるためには，各拠点ごとに複数メタデータサーバを構成し，書き込みリクエストの担当範囲を分割し，複数のサーバに担当を割り当て，他拠点への転送も，他拠点での担当メタデータサーバへ転送を行うことで，書き込みリクエストの担当サーバ分割を行う必要があると考えられる．

6. 関 連 研 究

Yahoo! PNUTS[15] は，レコードベースの共有ストレージシステムで，広域のデータセンター間でのデータ複製を考慮している．同一のシステムコンポーネントが各データセンターに分散配置され，クライアントからの更新は非同期で別のデータセンターに転送される．timeline consistency と呼ばれる一貫性モデルにより，レコード単位にマスターサーバが割り当てられ，レコードの更新順序はマスターサーバへのリクエスト到着順で決定される．HGMDS とは更新順序は因果順序をのみを保証し，関連のない 2 つ以上の更新は検知し上書きは保留されるため，必要ならば 2 つの更新のマージを行える．

Amazon Dynamo[16] は，High Available な Key-Value のハッシュストレージシステムである．Vector Clock によるオブジェクトバージョニングを行っており，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．データの配置先は，ノードの物理的な位置を考慮しないため，広域の場合高遅延通信路の影響を受けてしまう可能性がある．また，複数 Key のアトミックな更新はサポートされていないため，複数のエントリをアトミックに更新する必要がある，inode を管理するのには適していない．

7. お わ り に

本稿では，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，性能評価について述べた．広域ファイルシステムに適したファイルシステムセマンティクスとして，2-level Consistency と，Soft Client-centric Consistency を保証した，HGMDS の実装を行った．広域分散計算機環境である InTrigger を利用し性能評価を行った結果，クライアントリクエスト応答時間は書き込みが 145μsec，読み込みが 107μsec であった．また，複数拠点を利用した，HGMDS のリクエスト処理性能の評価を行い，HGMDS が広域環境においても高いスループットと低遅延であることを確認した．

今後は，性能評価で明らかになった，メタデータ共有拠点数を増やした場合にシステムのスループットが低下する問題の改善を行い，未実装のインタフェースを実装した後，提案分散メタデータサーバを用いた低遅延な広域ファイルシステム HGFS の提案と実装を行う．

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21013005) 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) : Avian Flu Grid. http://avianflugrid.pragma-grid.net/.
2) 田中昌宏，建部修見：グラフ分割による広域分散並列ワークフローの効率的な実行，先進的計算基盤システムシンポジウム SACSIS2010 論文集 (2010).
3) Tatebe, O., Hiraga, K. and Soda, N.: Gfarm Grid File System, *New Generation Computing*, Vol.28 (2010). (to appear).
4) 平賀弘平，建部修見：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，情報処理学会研究報告，2009-ARC-182(24) 2009-HPC-119(24)，pp.139–144 (2009).
5) 平賀弘平，建部修見：広域ファイルシステムのための分散メタデータサーバの設計，情報処理学会研究報告，2009-HPC-121(32)，pp.1–7 (2009).
6) Vogels, W.: Eventually consistent. http://www.allthingsdistributed.com/2007/12/eventually_consistent.html.
7) Dean, J. and Ghemawat, S.: MapReduce: simplified data processing on large clusters, *Commun. ACM*, Vol.51, No.1, pp.107–113 (2008).
8) : GXPMake. http://www.logos.ic.i.u-tokyo.ac.jp/gxp/.
9) 柴田知秀，姜ナウン，黒橋禎夫，田浦健次朗，Jun, C.S.，Nan, D.，松崎拓也，辻井潤一，河原大輔，宇野毅明：GXP システムとそれを用いた大規模テキスト処理の実行，スーパーコンピューティングニュース， Vol.11, No.Special Issue 2, pp.114–135 (2009).
10) Terry, D.B., Demers, A.J., Petersen, K., Spreitzer, M., Theimer, M. and Welch, B.W.: Session Guarantees for Weakly Consistent Replicated Data, *PDIS '94: Proceedings of the Third International Conference on Parallel and Distributed Information Systems*, Washington, DC, USA, IEEE Computer Society, pp.140–149 (1994).
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) Hirabayashi, M.: Tokyo Cabinet. http://1978th.net/tokyocabinet/.
13) Charron-Bost, B., Mattern, F. and Tel, G.: Synchronous, asynchronous, and causally ordered communication, *Distrib. Comput.*, Vol.9, No.4, pp.173–191 (1996).
14) 斎藤秀雄，鴨志田良和，澤井省吾，弘中 健，高橋 慧，関谷岳史，頓 楠，柴田剛志，横山大作，田浦健次朗：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，*2007-HPC-111* (2007).
15) Cooper, B.F., Ramakrishnan, R., Srivastava, U., Silberstein, A., Bohannon, P., Jacobsen, H.-A., Puz, N., Weaver, D. and Yerneni, R.: PNUTS: Yahoo!'s hosted data serving platform, *Proc. VLDB Endow.*, Vol.1, No.2, pp.1277–1288 (2008).
16) 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).