アクセスパターンと回線遅延を考慮した遠隔ファイルアクセスの最適化

大辻弘貴[†]　建部修見[†]

広域分散ファイルシステムにおける遠隔ファイルアクセスの性能は，回線遅延やアクセスパターンに大きな影響を受ける．本論文では様々な条件下で遠隔ファイルアクセスの性能を評価し，システムの設定が性能に与える影響を調査した．そして，その結果に基づき自動的に設定を最適化する手法を考案した．この手法により，既存の遠隔ファイルアクセス方式では大幅に性能が低下する高遅延環境や非シーケンシャルのアクセスにおいて，大幅な性能向上を達成した．併せて，本提案手法による性能向上が有意なものであることを検証し，実測最大値に近い性能を示す事を確認した．

Optimization of Remote File Access Considering Access Pattern and Network Delay

Hiroki Ohtsuji[†] and Osamu Tatebe[†]

Performance of remote file access in distributed file system depends on network latency and access pattern. In this paper, we investigated effects to the performance of system configurations by evaluating performance of remote file access under various conditions. Then, we invented a method to optimize configuration automatically based on the result of the investigation. This method improves the performance of remote file access significantly under the high network delay condition where the existing method demonstrates lower performance. We also validate whether the performance improvement by the proposed method is significant and affirm that the improved performance is nearly maximum measured performance.

1. 序　　論

増加の一途を辿るデータを取り扱うため，広域環境でデータを共有する必要性が高まっている．特に e-サイエンスやデータインテンシブコンピューティングの発展により，複数のスパコン間における大規模データ共有や，地理的に離れた複数拠点共同での解析する機会も増えている．そのような背景で，広域分散ファイルシステムが広く用いられている．広域分散ファイルシステムではデータ共有の高速化のためにファイル複製[1)]が行われる事があるが，クライアントにストレージが十分にない場合や，ファイルの一部分のみが必要な場合には遠隔ファイルアクセスを行う必要がある．ファイル複製時のバルク転送と異なり，遠隔ファイルアクセスは回線遅延の影響を大きく受ける．例えば，高遅延環境におけるランダムアクセスは，LAN 内などの低遅延環境の場合と比較して大幅に性能が低下する．**図 1** は既存の方式における性能評価であり，低遅延環境では十分な性能を示していても，ランダムアクセスや高遅延環境での動作では性能が低下する事が分かる．このような性能低下に対処する場合，管理者が手動で設定を最適化することがあるが，考慮すべき要素が多い上，状況の変化に対応する事は多大な労力を要する．そこで，本論文では，分散ファイルシステムにおいて単一クライアントがネットワーク経由で単一のサーバに対してファイルアクセスを行う部分にフォーカスし，アクセスパターンや回線遅延，回線帯域を考慮して遠隔ファイルアクセスを自動的に最適化する手法を示す．

† 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

2. 関 連 研 究

遠隔ファイルアクセスには大きく分けて 2 つの方式がある．一つは遠隔手続き呼び出し（RPC）によってファイルのオフセットとサイズを指定して必要な部分を転送する方式と，Web のようにファイル全体を転送する方式である．前者には Gfarm[2)] ファイルシステムや NFS[3)]，PVFS[4)]，Lustre[5)] がある．これら

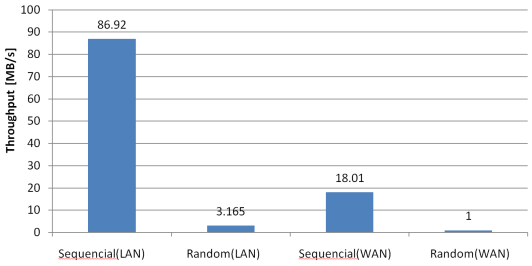

図 1 既存システム[2] における性能
LAN のネットワーク遅延は 50 μ s，WAN は 25ms
Fig. 1 Performance of existing method.
Network delay of LAN is 50us and WAN is 25ms.

の方式では，サーバとクライアントの間で，予め設定した一定量のデータを一つの単位としてやりとりを行う．このサイズは Gfarm バージョン 2.4.1 においては 1MB の固定値となっており，NFS ではバージョンによって異なるが数十 KB である．後者の方式には AFS[6] やその後継である Coda[7]，FTP の拡張である GridFTP[8] が挙げられる．本研究が対象とするのは前者の RPC による転送を行う形式である．

遠隔ファイルアクセスの性能向上の例としては，PVFS がファイルを分割して複数の I/O サーバに保管してストライピングを可能としている．また，Lustre も同様にストライピングを行ったり，同一ファイルに対する複数プロセスからのアクセスを集約したりと高速化が図られている．また，サーバとクライアント間では，回線遅延のある環境を考慮して複数の転送命令を同時に発行することも出来る．ただし，この場合にも転送データ量の単位など，固定されたパラメータが存在しチューニングが必要となる．その他にも，FTP をグリッド環境向きに拡張して作られた Grid FTP があり，複数の TCP コネクションを同時に利用する事で性能を向上させる機能が備わっている．この TCP セッションの数は基本的に固定値であるが，状況に合わせて最適化する研究も行われている．9) は実際に計測されたスループットを元にその時点で最適な TCP セッション数を探索する手法で，実環境に近い条件下においても高い性能を示している．

本論文では，利用状況と環境に基づいて，人手を介さずに RPC ベースの遠隔ファイルアクセスを最適化する研究について示す．

3. 遠隔ファイルアクセスの方式と性能評価

遠隔ファイルアクセスに用いられる RPC を更に細かく区分すると，同期型と非同期型の RPC がある．同期型 RPC とは，クライアントが 1 個ずつサーバにリクエストを送信し，完了を待ってから次に進む方式である．一方の非同期型は手順の進行状況に関係なくリクエストを発行することができる方式である．ここでは前者の同期型 RPC を対象としている．非同期型の RPC は回線遅延の影響を受けにくい利点があるが，アプリケーションが非同期型の I/O に対応していなければ，遠隔ファイルアクセスの段階で予測などが必要となり，その効力を最大限に発揮することは出来ない．従って，すべてのアプリケーションが非同期型の I/O に対応していない限りは，同期型 RPC による遠隔ファイルアクセスが必要でありその性能を高めなければならない．本論文において提案する手法は，あらゆる同期型 RPC を用いた遠隔ファイルアクセスに適用可能である．

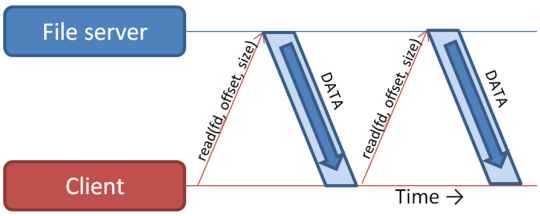

図 2 同期型 RPC による遠隔ファイルアクセス
Fig. 2 Remote file access using synchronous RPC

本章では，複数のアクセスパターンと回線遅延における性能評価の結果を示す．

3.1 同期型 RPC によるファイルアクセス

図 2 に同期型 RPC によるファイルアクセスの流れを示す．これは，クライアントとサーバの二台の間でどのような情報がやりとりされるのかを示しており，クライアントの要求に含まれる fd はファイルの識別子，offset は要求データのファイル中における位置，size は要求データ量である．これら 3 つの引数を含む要求を受け取ったファイルサーバは，対応するデータをクライアントに返送する．この一連の動作を 1 回ずつ行うのが同期型 RPC による遠隔ファイルアクセスである．この場合，回線遅延が発生すると，クライアントの要求がファイルサーバに届くまでの時間が増大し，結果的に RPC の実行時間が増大する．この結果，単位時間あたりに実行される RPC の回数が減少し，性能が低下する．

3.2 実験環境および評価方法

本論文に示す評価に用いた実験環境は，Linux をセッ

トアップした 2 台のコンピュータを GigabitEthernet で接続したものである．それぞれをクライアントとファイルサーバとして使用した．サーバ側には SATA 接続 7200 回転のハードディスクを RAID-0 構成で 2 台接続しており，170MB/s とネットワークに比べて十分高い性能が出ることを確認している．また，評価に当たっては，ソケットを直接用いたクライアント用およびサーバ用のプログラムを作成し，実際に同期型 RPC によるファイルアクセスを行った．このプログラムは 3.1 で述べた手順を忠実に実行する．回線遅延は Linux の tc コマンドによりクライアントコンピュータのネットワークインターフェースで発生させた．各評価結果については，3 回測定して平均値を取った値を示している．

以後，本論文における評価はすべて同じ方法で行っている．

3.3 RPC バッファサイズと性能

3.1 で述べた同期型 RPC によってクライアントはファイルサーバにデータを要求する．この RPC には size というパラメータがあり，これは 1 回の RPC で転送するデータ量を表している．以後この値を RPC バッファサイズと呼ぶ．NFS や Gfarm 等の既存のシステムでは RPC バッファサイズは固定値である．遠隔ファイルアクセスは，RPC バッファサイズ分のデータのやりとりを繰り返すことで行われる．

従って，同期型 RPC でバッファサイズが固定値の場合，クライアントとサーバの間の回線遅延が増大する事によって応答待ち時間の割合が増大すると，単位時間あたりに送受信できるデータ量が低下する．これはシーケンシャルアクセスを行う場合に問題となる．一方で，クライアントの要求が例え 1 バイトであったとしても，一度の RPC ではバッファサイズ分の転送を行うため，RPC バッファサイズが必要とする読み込み量に対して大きい場合には，無駄な転送が増えて性能が低下する．この状態はランダムアクセスやストライドアクセスによって引き起こされる．

RPC バッファサイズが遠隔ファイルアクセスに与える影響を評価するため，様々な回線遅延やアクセスパターンの元で RPC バッファサイズを変えながら性能評価を行った．次節以降にその結果を示す．

図 3，**図 4** と**図 5** はそれぞれ，シーケンシャルアクセス，512KB の読み込みと 3.5MB のシークを繰り返すストライドアクセス，同じく 3MB の読み込みと 6MB の読み込みのストライドアクセスの性能を示している．横軸は RPC バッファサイズで 64KB から 512MB までの幅であり，縦軸はスループット [MB/s] である．グラフの各線は設定したネットワークの遅延時間で，0ms から 100ms の幅である．グラフ中のエラーバーは測定する際に得られた最大値と最小値を表している．

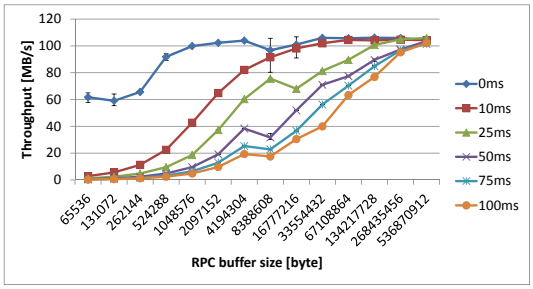

図 3 シーケンシャルアクセスの性能
Fig. 3 Performance of sequential access

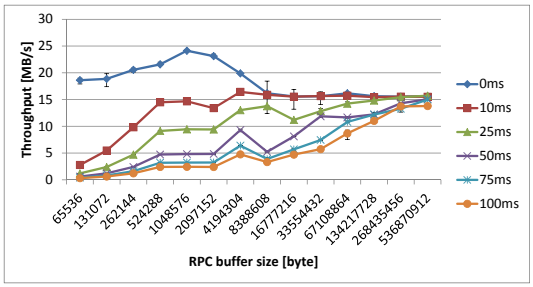

図 4 ストライドアクセス：512KB 読み込み 3.5MB シークの性能
Fig. 4 Performance of stride access: read 512KB and seek 3.5MB

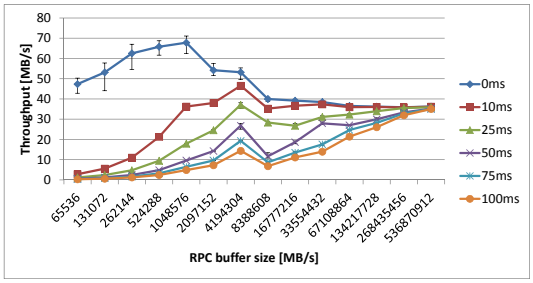

図 5 ストライドアクセス：3MB 読み込み 6MB シークの性能
Fig. 5 Performance of stride access: read 3MB and seek 6MB

図 3 から分かるように，シーケンシャルアクセスにおいては回線遅延が小さい場合には RPC バッファサイズが約 1MB 以上であれば十分な性能を示す．これ以下のサイズにおける性能低下は，プロトコルオーバーヘッド等の増大が原因である．一方，回線遅延が大きい場合，RPC バッファサイズも大きく無ければ高い性能が出ない．例えば 100ms の回線遅延の場合，RPC バッファサイズが 4MB でようやく 20MB/s を

超え，512MB でようやく 100MB/s に達する．この特性には 2 つの原因が挙げられる．1 つは同期型 RPC が完了するまでの時間が，回線遅延が長くなるとその分だけ増大することである．時間がかかる程次のリクエストを発行するまでに間隔が空き，何もしない時間が増える．2 つめは TCP の特性で，小さなサイズの転送を断続的に繰り返しても，輻輳ウィンドウのサイズが広がらず転送速度が上がらないことである．従って，シーケンシャルアクセスでは RPC バッファサイズを大きくする程高い性能を出せる．

一方，図 4 と図 5 等のストライドアクセスから分かることは，回線遅延が小さい場合には RPC バッファサイズも小さい方が性能が高く，回線遅延が大きい場合には逆に RPC バッファサイズが大きい方が高い性能を示していることである．すなわち，アクセスパターンが同じであっても，回線遅延によって最適値が異なる．この性質は 3.1 章で述べた事が原因である．まず，RPC バッファサイズがクライアントの要求するデータ量に比べて大きすぎる場合には，ネットワーク上を転送したデータのうち使用されない部分が生じる．従って RPC バッファサイズを小さくすることで必要なデータのみを転送することが可能となり，高い性能を得ることができる．これは図 5 中 0ms や 10ms の線からも分かるように，ピーク性能が RPC バッファサイズの小さな領域に存在する．反対に，回線遅延が大きくなると小さな RPC バッファサイズでは十分な性能が出ない．これは，RPC の完了時間は最低でも回線遅延時間分かかる事が原因である．従って，回線遅延が大きい場合には，無駄な伝送を減らすために少量のデータを何回も転送するよりも，RPC バッファサイズを大きくして RPC 発行回数を抑えた方が高い性能を得られる．

つまり，シーケンシャルアクセスならば RPC バッファサイズが大きい方が性能が高いが，ストライドやランダムアクセスでは場合に依るため，一概に決めることは出来ない点が問題である．

4. アクセスパターンの認識と RPC バッファサイズの動的変更

4.1 アクセスパターンの認識 – RPC バッファ利用率

これまでにアクセスパターン・回線遅延・RPC バッファサイズの 3 要素が，遠隔ファイルアクセスの性能に与える影響を示してきた．最適な RPC バッファサイズを求めるためには，アクセスパターンの認識をする必要があり，本章ではそのための手法について述べる．

3 章に示した通り，遠隔ファイルアクセスでは，クライアントが発行した RPC によってバッファサイズ分のデータを転送する．そこで，アクセスパターンを判断するにあたっては，この転送されたデータのうち実際にクライアントが必要としていた割合を測定することでアクセスパターンを判断する事とした．この結果を RPC バッファ利用率と呼ぶこととする．

例えば，シーケンシャルアクセスを行った場合には，RPC バッファの領域を連続して読み込むことを繰り返すため，RPC バッファ利用率は 100%になる．一方で一度の読み込みサイズ（ブロックサイズ）が RPC バッファサイズよりも小さなランダムアクセスを行った場合には，転送されたバッファのうち一部分のみしか利用されないことから，RPC バッファ利用率は 100%未満の値となる．従って，RPC 利用率が高ければシーケンシャルアクセスであり，低ければランダムアクセスであると判断できる．

この RPC バッファ利用率を測定する方法はいくつか考えられる．最も直接的な方法は，クライアントアプリケーションのアクセスログを定期的に集計し，RPC バッファのうちどの程度使用されたかを集計する方法である．しかし，この方法では非常に小さなサイズの読み込みが何度も行われた場合に，ログサイズが増大して処理に要する時間が長くなる問題がある．また，アクセス順序や回数は不要な情報であり，メモリを無駄にしてしまう．

そこで，本研究では，RPC バッファ中でどこが使われたかを示す情報を保持する手法を提案する．まず考えられるのは，RPC バッファ中のバイト毎に利用の有無を記録する方法で，利用されたバイト数を総バイト数で除算することで RPC バッファ利用率を求めることが出来る．ところがバイト毎に利用状況を管理した場合，1 バイトにつき 1 ビットの管理情報が必要となり，RPC バッファサイズの増加につれてメモリ使用量も増大してしまう．この問題を解決するために，本研究では RPC バッファ領域を n 個のブロックに分割して管理することとした．あるブロックに含まれる領域を一度でもクライアントアプリケーションがアクセスした場合に，利用済みとしてマークし，マークされたブロック数によって RPC バッファ利用率を定義する．

図 6 は RPC バッファをブロックに分割して RPC バッファ利用率を測定する手順を示している．図中一段目はアプリケーションが発行した I/O リクエストである．また，二段目は遠隔アクセスのクライアント

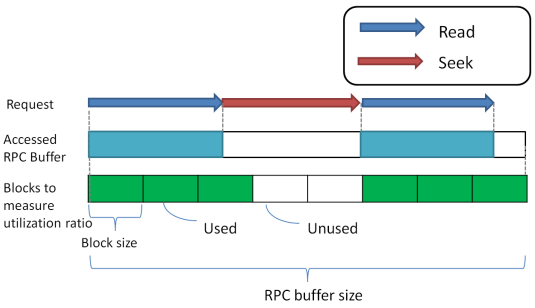

図 6 RPC バッファ利用率の測定手順
Fig. 6 Procedure to measure RPC buffer utilization ratio

プログラムがサーバから受け取ったデータを保存している RPC バッファ領域を示しており，水色の部分はクライアントアプリケーションが読み込みを行った領域である．三段目は，この領域をブロックに分割し，その領域に含まれる RPC バッファが一部でも読み込まれていれば使用済みとして緑色にマークしたものである．三段目に示すように，利用の有無をブロック数単位で RPC バッファ利用率を計算し，その式は以下の通りとなる．

$$\text{RPC バッファ利用率} = \frac{\text{使用済みブロック数}}{n} \quad (1)$$

次に，ブロック分割数の n について考える．n を増加させると，バッファ領域の利用の有無を細かく管理できるようになるので，測定された RPC バッファ利用率がより正確になる．しかしながら，n を増加させて正確にすればするほど良いとは限らない場合がある．その理由は 3.3 章に示したストライドアクセス時の性能特性である．非常に小さな領域の読み込みとシークを交互に繰り返すようなアクセスを行った場合，遅延の大小によって RPC バッファサイズを大きくすべきか小さくすべきかが変わる．

そこで n が変わると RPC バッファ利用率の測定結果がどのように変化するかを**図 7** に示す．最上段はクライアントからのリクエストを表しており，同じ量の読み込みとシークを繰り返している．n が 16 の場合にはクライアントからのリクエスト通り，RPC バッファ利用率は 50%と測定され，ランダムアクセスであると判断される．一方で n が 4 の場合には RPC バッファ利用率は 100%と測定される．すなわち，シーケンシャルアクセスであると判断されるのである．RPC バッファ利用率の定義として，あるブロックが一度でもアクセスされれば使用済みとしていることから，n

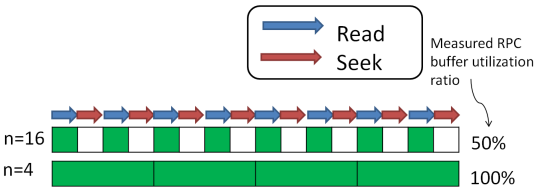

図 7 n の違いによる RPC バッファ利用率の変化
Fig. 7 Deference of RPC utilization ratio by n

を減らすと，測定される利用率は変化しないか増加するかのどちらかである．極端な例として n が 1 ならば，どのようなアクセスも 100%と測定される．この性質をうまく利用することで，RPC バッファサイズをどう変化させるべきかが回線遅延の大小によって変わる場面において，適切に対応する手法を考案した．

前章で述べたように，低遅延環境下のランダムアクセスやストライドアクセスにおいて，RPC バッファサイズを小さくする意義は，無駄な転送を減らすことである．しかし，RPC バッファサイズを小さくすると必然的に RPC 発行回数が増える．RPC の完了時間は最低でも回線遅延分を必要であるから，回線遅延が増大すればそれだけ時間がかかる．従って，RPC バッファサイズを小さくするメリットは，削減できた無駄な転送量が，回線遅延分の時間に転送出来るデータ量を上回っている間にのみ存在する．つまり，このメリットが失われた状況では，シーケンシャルアクセスと同じ扱いをして，まとめて転送してしまった方が良い．このアクセス方法は，data sieving[10] と呼ばれている．

この点を n の決め方に応用することを考える．回線遅延分の間に転送出来るデータ量は，回線遅延時間と帯域の積で表すことが出来，以後これを帯域幅遅延積と呼ぶ．そして，この帯域幅遅延積がシーク量，要するに削減し得る無駄な転送よりも大きければ，そのシークされた領域を利用済みと見なすことでシーケンシャルアクセスと同じ扱いを出来る．これを実現するためには，RPC バッファ利用率測定に用いるブロックサイズが帯域幅遅延積になるよう，n を設定すれば良い．従って，n は次の式を満たす値を取れば良い．

$$\text{帯域幅遅延積} = \text{ブロックサイズ} \quad (2)$$

$$= \frac{\text{RPC バッファサイズ}}{n} \quad (3)$$

この n と，測定された回線帯域・回線遅延を元に RPC バッファ利用率を求める．

4.2 RPC バッファサイズの動的変更

4.1 章に示した方法で RPC バッファ利用率を求め，アクセスパターンを判別することができる．RPC バッファ利用率は，連続的なアクセスでは高く，不連続なアクセスでは低くなる．従って，RPC バッファ利用率が高ければシーケンシャルアクセスであり，低ければランダムアクセスであると判断することが出来る．この判断に基づき，一度の RPC でやりとりするデータ量である RPC バッファサイズを動的に変更したい．この動的な変更を行うにあたっては，一時的なアクセスパターンの変化にすぐ対応するか，あるいはある程度継続したアクセスパターンを重視するかを設定できるようにした．そのために，過去 m 回分の RPC バッファ利用率の平均値に基づいて RPC バッファ利用率を変更することとした．また，バッファサイズが非常に大きい場合，一度の RPC で必要とする転送時間が長く，最適ではない状態が長く続くとペナルティが大きい．そのため m の値について，RPC バッファサイズが $LARGE\_BUFSIZE$ を上回ったら M_{small}，$LARGE\_BUFSIZE$ 以下であれば M_{normal} と設定する．アクセスパターンの変化の頻度やアプリケーションのアクセスの性質によりこれらのパラメータの最適値は変わる．

この RPC バッファ利用率は RPC バッファのうちどれだけのデータを転送すべきであったかを示す数値であるから，利用率が低ければ RPC バッファサイズを縮小し，逆に高ければ RPC バッファサイズをある増加率で乗算して拡張する．利用率の高低の判断は 2 つの閾値によって行い，高い方が U_{high}，低い方を U_{low} とする．RPC バッファサイズの変更手順を，以下に示す．

if (RPC バッファ利用率 $\rangle$ U_{high})
　RPC バッファサイズ　*=
　　RPC バッファサイズ増加率
else if (RPC バッファ利用率 $\langle = U_{low}$)
　RPC バッファサイズ *=
　　RPC バッファ利用率

この手順により，シーケンシャルアクセスによって RPC バッファ利用率が高い状態が続けば RPC バッファサイズが増加する．一方でランダムアクセスによって RPC バッファ利用率が低くなると，RPC バッファサイズが縮小されていき，適切な値になると RPC バッファ利用率が再び上昇する．RPC バッファサイズが最適値になると利用率が高い値になるが，これはシーケンシャルアクセスではないので RPC バッファサイズを拡大すべきではない．しかしこの手順だけでは，利用率の上昇によりバッファサイズが増加し，最適値を外れて利用率が低下してバッファサイズが元の小さなサイズに戻るというサイクルを繰り返してしまう．ただし，シーケンシャルアクセスに移行していた場合にはそのままバッファサイズが増大し，それは正しい動作である．

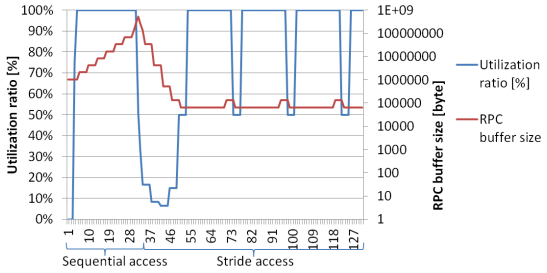

図 8 提案手法の挙動
Fig. 8 Behavior of proposed method

しかしながら，RPC バッファサイズが最適値を頻繁に外れることは好ましくなく性能に悪影響を与える．そこで，RPC バッファサイズ増加直後に利用率が低下した場合，以降 p 回の RPC 発行の間はバッファサイズを拡大しない仕組みを導入した．p が大きければランダムアクセスの継続性を重視し，小さければアクセスパターンの変化に素早く反応できるようになる．

4.3 実装の検証

この章でこれまで述べてきた，RPC バッファ利用率に基づく RPC バッファサイズの動的変更手法を実装して動作させ，その挙動を検証した．

この検証にあたっては，各種パラメータを次の通り設定した．$LARGE\_BUFSIZE$ を 128MB，M_{small} を 1，M_{normal} を 4，U_{high} を 95%，U_{low} を 50%，RPC バッファ増加率を 2，p を 16 とした．また，RPC バッファサイズは 2 のべき乗を取るので，RPC バッファサイズを縮小する際には，RPC バッファ利用率が 50%以下であれば半分，25%以下では 4 分の 1，12.5%以下では 8 分の 1 とした．

図 8 に示したグラフは，1GB のシーケンシャルアクセスから 61KB の読み込みと 1MB のシークを繰り返すストライドアクセスに移行した際の RPC バッファ利用率と RPC バッファサイズの推移を示している．横軸は RPC 発行回数で，縦軸は左が RPC バッファ利用率 [%]，右が RPC バッファサイズ [KB] の対

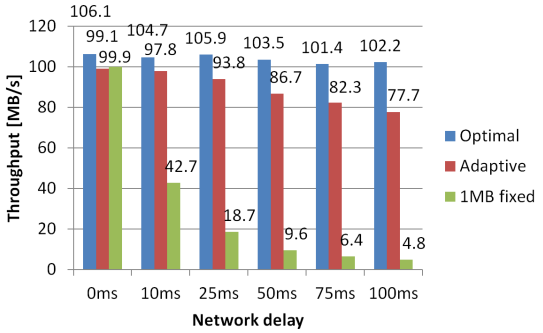

図 9 シーケンシャルアクセスの性能比較
Fig. 9 Comparison of sequential access performance

数をとったものである．

RPC30 回目まではシーケンシャルアクセスが行われたことで RPC バッファ利用率が上昇し，それに伴ってバッファサイズも拡大している．それ以降はストライドアクセスが行われたことで RPC バッファ利用率が低下し，バッファサイズが縮小する．バッファサイズが縮小し，適正値に近づくと徐々に利用率が向上し，適正値となる．バッファサイズが適正値になると利用率も 100%を示す．しかし，これはシーケンシャルアクセスではないため，バッファサイズを拡大すると利用率が低下する．そのため，前章で述べたように p 回分はバッファサイズを維持することで最適値をより長く保っている．この定常状態がグラフの後半に示されている．

5. 性 能 評 価

遠隔ファイルシステムに本提案手法を適用し，性能評価を行った．その結果を各節に示す．評価環境は 3.2 に示した通りである．

5.1 シーケンシャルアクセス

図 9 はシーケンシャルアクセスの性能比較のグラフで，横軸は回線遅延，縦軸はスループットを示している．グラフ中の各項目で，Optimal は 3.3 に示した各グラフ中で，最も高い性能を示した部分を抽出した物で，性能の限界値を示している．Adaptive は本提案手法を適用した場合の性能である．1MB fixed は Gfarm バージョン 2.4.1 相当の性能を示しており，既存システムの一例で比較対象でもある．性能測定はいずれも 6GB のファイル転送によって行った．

グラフからも分かるように，1MB fixed では遅延が増大するに従って性能が大幅に低下するが，Adaptive ではそれ程低下せず，高い性能を保っている．Adaptive と Optimal との乖離は回線遅延の増大につれて大きくなるが，これは RPC バッファサイズが立ち上がるまでに時間がかかっている事が原因で，より大きなファイルを転送すると乖離は小さくなる．**表 1** に示したように，提案手法が選択した RPC バッファサイズは，低遅延環境においてバッファサイズが飽和する場合を除いては，実測パフォーマンスから求めた最適値（Optimal）と同等となっている．

表 1 シーケンシャルアクセスで選択された RPC バッファサイズ
Table 1 Selected RPC buffer size in sequential access

Network delay	Adaptive	Optimal
0ms	512MB	128MB
10ms	512MB	256MB
25ms	512MB	512MB
50ms	512MB	512MB
75ms	512MB	512MB
100ms	512MB	512MB

5.2 ストライドアクセス

図 10 と**図 11** はそれぞれ 512KB 読み込み 3.5MB シークを繰り返すストライドアクセスと，同 3MB 読み込み 6MB シークの性能比較を行ったグラフである．グラフの軸や各項目はシーケンシャルアクセスと同じである．ほとんどのケースで，提案手法は 1MB fixed よりも高い性能を示しており，Optimal に近づいていることが分かる．

表 2 と**表 3** に，各回線遅延の下で本提案手法が選択した RPC バッファサイズと Optimal の値を示す．こちらも，本提案手法がほぼすべての状況で最適な RPC バッファサイズを選択していることが読み取れる．

5.3 提案手法のオーバーヘッド

本提案手法を実装するにあたっては，RPC バッファの利用状況を管理するためのメモリと，履歴用のバッファが必要である．それぞれ，ブロック数を 4096 とすると 512 バイト，倍精度浮動小数点の変数 8 個で 64 バイトとなり，合計 576 バイトに過ぎず，通信用のバッファなどに比べれば非常に小さい．また，処理に必要な時間も非常に短く，実測で RPC1 回あたり 160 μ秒であった．オーバーヘッドが非常に小さいことから，本提案手法による性能向上をそのまま活かすことが可能である．

6. ま と め

本論文では，同期型 RPC を用いた遠隔ファイルアクセスについて，回線遅延と回線帯域，アクセスパターンの 3 つの要素を同時に考慮しながら最適化する方法を示した．最適化する対象は一度の RPC によって転

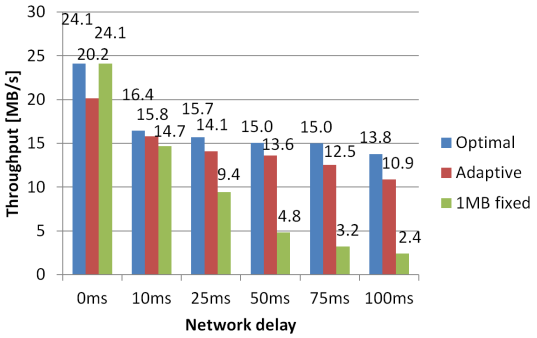

図 10 ストライドアクセス（512KB 読み込み 3.5MB シーク）の性能比較
Fig. 10 Comparison of stride access (read 512KB and 3.5MB seek) performance

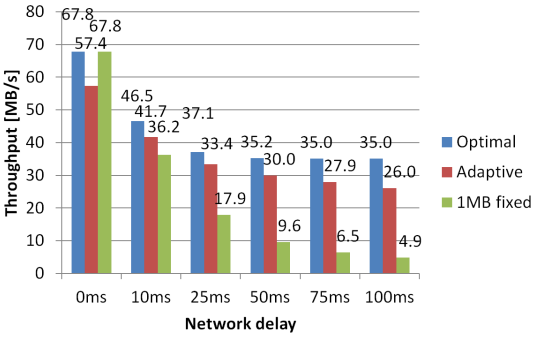

図 11 ストライドアクセス（3MB 読み込み 6MB シーク）の性能比較
Fig. 11 Comparison of stride access (read 512KB and 3.5MB seek) performance

送されるデータ量を表す RPC バッファサイズである．まずはこの値を変えながら様々な条件で性能評価を行い，最適値について調べた．その後に RPC バッファサイズを最適化するための手法について述べ，この手法が性能向上に有効であることを性能評価によって示した．提案手法は非常に小さなオーバーヘッドで大きな性能向上を達成できており，多くの場合で最良のパラメータを選択していた．シンプルなアイディアで守株の条件を勘案して最適化できる点は本提案手法の最大の特長であり，あらゆるシステムに導入が可能であり，非常に応用の範囲が広い．

表 2 ストライドアクセス (512KB 読み込み 3.5MB シーク) で選択された RPC バッファサイズ
Table 2 Selected RPC buffer size in stride access (read 512KB and seek 3.5MB)

Network delay	Adaptive	Optimal
0ms	512KB	1MB
10ms	4MB	4MB
25ms	512MB	512MB
50ms	512MB	512MB
75ms	512MB	512MB
100ms	512MB	512MB

表 3 ストライドアクセス (3MB 読み込み 6MB シーク) で選択された RPC バッファサイズ
Table 3 Selected RPC buffer size in stride access (read 3MB and seek 6MB)

Network delay	Adaptive	Optimal
0ms	1MB	1MB
10ms	4MB	4MB
25ms	4MB	4MB
50ms	512MB	512MB
75ms	512MB	512MB
100ms	512MB	512MB

7. 今後の課題

今回は同期型の I/O と RPC を用いることを前提としていたが，今後は同期型 RPC だけではなく，非同期に先読みを行うなど，更に性能を向上出来る可能性のある方式についても検討を進める．また，実システムへの導入を進め，より実践的な性能評価を行う事も課題である．

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 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) Chervenak, A. L., Foster, I. T., Kesselman, C., Salisbury, C. and Tuecke, S.: The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets, *JOURNAL OF NETWORK AND COMPUTER APPLICATIONS*, Vol. 23, pp. 187–200 (1999).
2) Tatebe, O., Hiraga, K. and Sod, N.: New Generation Computing, Ohmsha, Ltd. and

Springer, *Gfarm Grid File System*, Vol. 28, No. 3, pp. 257–275 (2010).
3) Callaghan, B., Pawlowski, B. and Staubach, P.: NFS Version 3 Protocol Specification, *RFC 1813* (1995).
4) Philip H. Carns, Iii, R. B. R. and Thakur, R.: Pvfs: a parallel file system for linux clusters, *In ALS 00: Proceedings of the 4th annual Linux Showcase and Conference* (2000).
5) Braam, P. J.: Lustre, *http://www.lustre.org/*.
6) Howard, J. H.: Scale and performance in a distributed file system, *ACM Trans. Computer Systems*, Vol. 6, No. 1, pp. 51–81 (1988).
7) Satyanarayanan, M.: Coda: A Highly Available File System for a Distributed Workstation Environment, *IEEE Trans. Computers*, Vol.39, No. 4, pp. 447–459 (1990).
8) Allcock, W.: GridFTP: Protocol Extensions to FTP for the Grid, *Global Grid Forum Draft* (2003).
9) 伊藤建志, 大崎博之, 今瀬眞: データ転送プロトコル GridFTP のための並列 TCP コネクション数調整機構の性能評価, 電子情報通信学会技術研究報告 情報ネットワーク 105(628), pp. 201–208 (2006).
10) Thakur, R., Gropp, W. and Lusk, E.: Data Sieving and Collective I/O in ROMIO, *In Proceedings of the Seventh Symposium on the Frontiers of Massively Parallel Computation*, pp. 182–189 (1988).