Vol.2010-ARC-192 No.6
Vol.2010-HPC-128 No.6
2010/12/16

# 遠隔ファイルアクセスにおける アクセスパターンと遅延の影響

大辻弘貴[†]　　建部修見[††]

広域分散環境における遠隔ファイルアクセスでは，ネットワーク遅延やアクセスパターンが性能に大きく影響を与える．本論文では，自動的なアクセスパターンの認識によって設定を最適化する手法を提案する．この手法により，既存ファイルシステムの性能が大幅に低下する高遅延・ランダムアクセスの条件下にて，最大 350%の性能向上を達成した．また，新しいストレージデバイスである SSD を用いた遠隔ファイルアクセスの性能を評価した．SSD の使用によって，低遅延・ランダムアクセスにおいて HDD 比 450%の性能向上を示した．

# Effects of Access Patterns and Delay on Remote File Access

Hiroki Ohtsuji[†]　and　Osamu Tatebe[††]

Network latency and access pattern have a major impact on performance of wide-area distributed remote file access. This paper proposes a method to optimize parameters of remote file access by automatic recognition of access pattern. This method can improve the performance of random access in the high latency condition by 350 percent, compared to conventional file system which indicates low performance in the above condition. In addition, we evaluate the performance of remote file access using a SSD which is one of new storage devices. Using a SSD improve the performance of random access in the low latency condition by 450 percent, compared to using a HDD.

## 1. はじめに

e-サイエンスの発展に伴い，複数のスパコン間で大規模データの共有，大規模データを複数拠点共同で解析するなど，広域環境において効率的にデータを共有することが求められている．遠隔の拠点間におけるデータ共有を効率的に行うためには，ファイル複製が利用されている[1]．ファイル複製作成はデータをバルク転送できるため，ネットワーク遅延が大きくても効率的に転送することができる．しかしながら，ファイルの一部しか利用しない場合や，ファイル複製を作成するための十分なストレージ容量が確保できない場合など，遠隔ファイルアクセスを利用することとなる．

遠隔ファイルアクセスの性能はアクセスパターンやネットワーク遅延に大きく左右される．既存のシステムにおいては，図 1 に示すように高遅延環境での動作やランダムアクセスの性能が LAN 内におけるシーケンシャルアクセスに比べて著しく低い問題があり，性能を向上させるためには性能低下の原因を調査し，設定の最適化を行う必要がある．特に回線遅延は地理的に分散した環境では不可避なものであるため，慎重な検討を要する．本論文では，遠隔アクセスに関してこれらの問題を分析するとともに，最適な設定を自動的に行う手法を提案する．

分散ファイルシステムの遠隔アクセスには，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 のように要求に対してファイル全体を転送する方式がある．前者には Gfarm ファイルシステム[2]や NFS[3]があり，後者には AFS[4]やその後継の Coda[5]がある．本論文では前者の同期型の RPC を対象としており，この方式における遠隔ファイルアクセスがアクセスパターンや回線遅延によって受ける影響を分析し，設定値の動的な最適化による性能向上を評価する．

## 2. 関連研究

遠隔アクセスの設定を状況に応じて自動的に最適化する例は既に存在している．例えば，[6]は遠隔ファイルアクセスの自動的な最適化の研究である．この論文と本研究は，遠隔ファイルアクセスの状況をモニタしてそれに応じて設定値を最適値に変更する点で，考え方および方法は類似している．異なる点は最適化する対象である．前述の論文においては読み書きされるファイルの重複部分の度合いを推定して deduplication 呼ばれる操作を行うか否かを決定する．deduplication とは同じデータを一つに集約してデータ転送の高速化やストレージ領域を節約する操作である．それに対

[†] 筑波大学情報学群情報科学類
College of Information Science, University of Tsukuba
[††] 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

Vol.2010-ARC-192 No.6
Vol.2010-HPC-128 No.6
2010/12/16

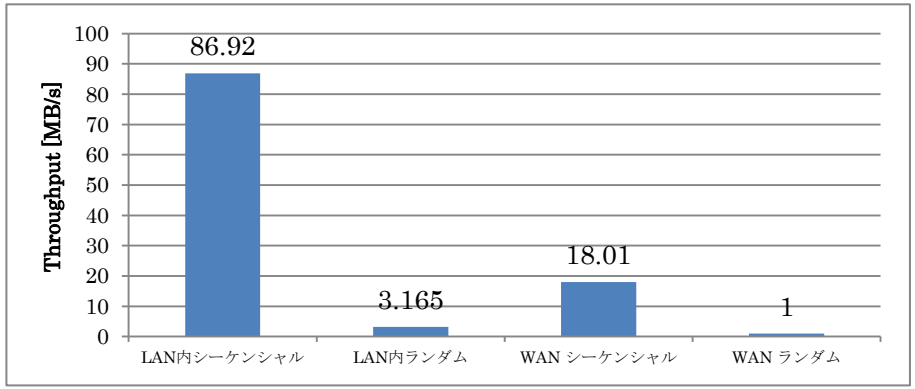


図 1　Gfarm にて性能に問題がない場合と，特に性能が低下する 3 例の比較
ネットワーク遅延は LAN が 50 $\mu$ s，WAN が 25ms である
ランダムアクセスは読み込みサイズ 61KB である

して本研究ではアクセスパターンをモニタして，3 章で後述する RPC バッファサイズという設定値を最適値に変更する．

本研究は，利用状況や環境を考慮して，人手を介さずに遠隔ファイルアクセスを最適化する試みの一つである．

## 3. アクセスパターンと遅延の影響

この章では，同期型 RPC が用いられているファイルアクセスにおいてアクセスパターンと遅延が性能に与える影響を示す．まずは，同期型 RPC によるファイルアクセスの流れを図 2 に示す．fd はファイルの識別子，offset は要求データのファイル中での位置，size は要求するデータ量である．クライアントはこれら 3 つの引数を含む要求をファイルシステムサーバに送信し，データを受け取る．この一連の動作を 1 つずつ行うのが同期型 RPC である．従って，発行したリクエストの完了を待たなければ次のリクエストを発行することが出来ないことから，回線遅延によって遅れが発生すると何もしていない応答待ち時間が増大し，性能が低下する．

リクエスト 1 回につき転送するデータサイズを RPC バッファサイズと呼ぶ．Gfarm においては現在 1MB の固定値である．この RPC バッファサイズ分のデータの転送を繰り返すことによってクライアントアプリケーションが要求するデータを転送する．RPC バッファサイズが固定値の場合，クライアントとファイルシステムサーバ間のネットワーク遅延が大きくなると，応答待ちや処理などのオーバーヘッドが増大してしまう．一方で，必要とする領域が RPC バッファサイズよりも小さいランダムアクセス

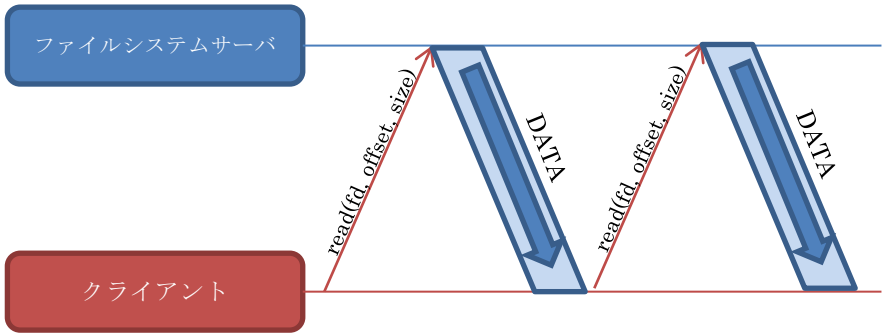


図 2　同期型 RPC による遠隔ファイルアクセス

を行うと，不必要なデータを送ることになり無駄が発生する．

この特性を検証するため，Gfarm v2.4.0 においてシーケンシャルアクセスとストライドアクセスの性能測定を行った．測定に当たり Gfarm に変更を加え，RPC バッファサイズを起動時のパラメータで指定できるようにした. 図 3 と図 4 はそれぞれ, Gfarm におけるシーケンシャルアクセスおよびストライドアクセス（1MB 読み込みと 16MB シークの繰り返し）のスループットであり，横軸はクライアント・サーバ間のネットワーク遅延時間，グラフのそれぞれの線は RPC バッファサイズである．遅延は tc コマンドを用いて，クライアントのネットワークインターフェースで発生させている．なお，Iperf と示されている線はクライアント・サーバ間における TCP による転送速度の 90 秒平均であり, 遠隔ファイルアクセスの性能向上を目指す上での一つの基準である．これらの図から，シーケンシャルアクセスについては 0ms の低遅延環境では RPC バッファサイズの大小にかかわらず 65MB/s を超える高い速度を出しており，その後遅延が増大するに従って RPC バッファサイズが小さい場合には性能が低下することが読み取れる．一方のストライドアクセスでは，0ms の低遅延環境では RPC バッファサイズが小さいほどスループットが高くなっており，遅延が増大すると総じて性能が低下し，100ms の遅延では 8MB の RPC バッファサイズでは 3.25MB/s，1MB の RPC バッファサイズであっても 4MB/s 程度の速度となる．

しかしながら，Gfarm を含め NFS 等も RPC バッファサイズは固定値で動作中に変わる事はないことから，利用が多いと想定される条件を元に設定値を決めることになり，その条件を外れると性能が著しく低下してしまう．例えば，主に低遅延環境で使用する場合には RPC バッファサイズを小さくするが, 遅延の大きな回線を経由してクライアントがシーケンシャルアクセスを行った場合，性能が低下することとなる．

従って，遠隔ファイルアクセスにおいて高い性能を出すためには，アクセスパター

Vol.2010-ARC-192 No.6
Vol.2010-HPC-128 No.6
2010/12/16

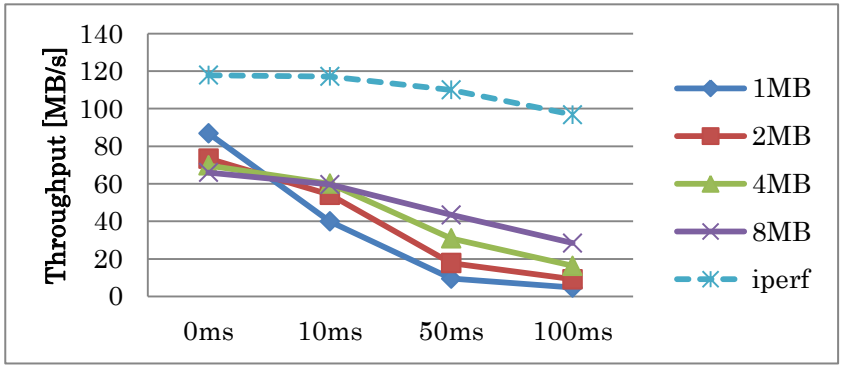


図 3　シーケンシャルアクセスのスループット(Gfarm)

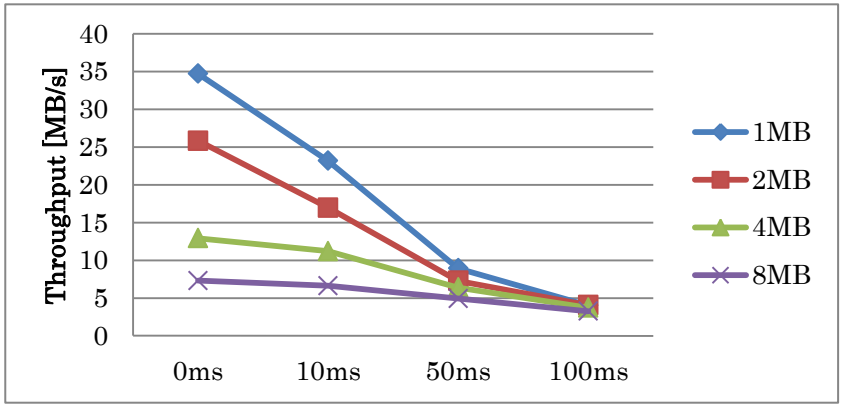


図 4　ストライドアクセスのスループット(Gfarm)

ンと回線遅延の双方を勘案して RPC バッファサイズを動的に最適値に設定しなければならない．以降，これを実現するための新しい手法について述べ，性能評価の結果を示す．

## 4. 提案手法とその評価

### 4.1 アクセスパターンの認識 –RPC バッファ利用率

RPC バッファサイズを最適化してスループットを向上させるためには，まずアクセスパターンを認識しなければならない．本論文ではアクセスパターンを認識するための一つの手法を提案する．アクセスパターンの認識にあたり，クライアントからの RPC に応じてファイルシステムサーバが転送した RPC バッファサイズ分のデータ中，クライアントが実際にそのデータを使用した割合を考える．ここではこの数値を RPC バッファ利用率と呼ぶ．シーケンシャルアクセスを行った場合，RPC バッファの領域を連続して読み込むことから，RPC バッファ利用率は 100%になる．一方で読み込みサイズが RPC バッファサイズよりも小さいランダムアクセスを行った場合には，RPC バッファのうち一部しか読み込まれないことにより，100%未満の値になる．この事により，RPC バッファ利用率が高ければシーケンシャルアクセスと判断し，低ければランダムアクセスであると判断する．

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．ひとつは，クライアントアプリケーションのアクセスログを取り，定期的に集計して RPC バッファ領域のうちどれだけが読み込まれたかを計測する方法である．この方法では非常に小さなサイズの読み込みが行われた場合，ログサイズが増加し処理に要する時間も増大する問題がある．また，アクセスされた回数や順番は必要なく，冗長な情報を保持することでメモリを消費してしまう問題がある．アクセスパターンを完全に理解するためには，RPC バッファのバイト毎につき，読み込みの有無を記録する方法が考えられる．しかし，1 バイトにつき 1 ビットを必要とし，RPC バッファの 8 分の 1 のメモリを消費する．また 1 バイトアクセスする毎に読み込みの有無を記録する必要があり，オーバーヘッドが大きいと考えられる．そこで，RPC バッファ領域を n 個のブロックに分割し，その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読み込んだのであれば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 5 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流れを示している．1 段目は，アプリケーションが発行した I/O リクエストをクライアントプログラムが受け取り，RPC バッファサイズ分のデータをサーバに要求する様子を表している．そして，2 段目はクライアント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，水色に着色された部分はアプリケーションが I/O によって実際に読み込んだ部分である．3 段目では，この領域をブロックに分割し，その範囲に含まれる 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ている．RPC バッファ利用率は最終的に以下のようになる．

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

続いて，ブロック分割数である n について検討する．n を増加させると，より細かくバッファ領域を分割できることから，正確に RPC バッファ利用率を測定できるようになる．しかしながら，細かければ細かいほど良いとは限らない．例えば，非常に小

Vol.2010-ARC-192 No.6
Vol.2010-HPC-128 No.6
2010/12/16

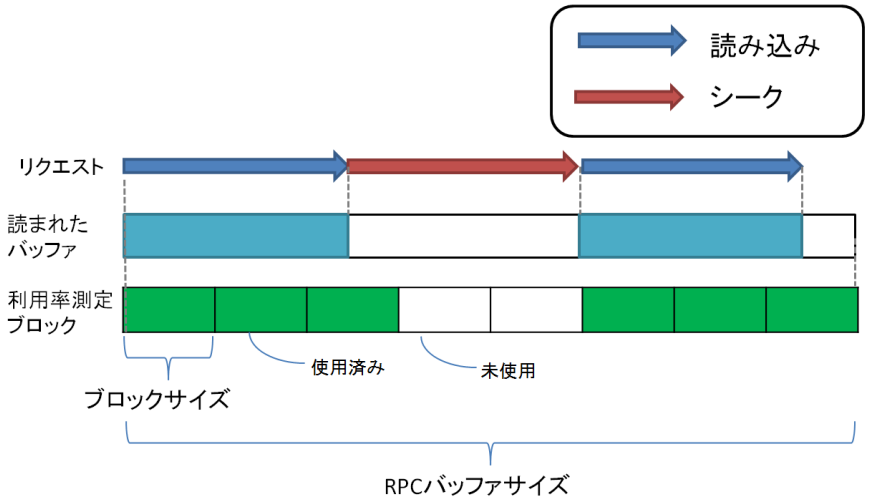


図 5　RPC バッファ利用率の測定手順

さな量の読み込みとシークを交互に繰り返したとする．n が十分に大きければ図 6 の 1 段目に示すように RPC バッファ利用率が 50%と測定され，ランダムアクセスであると判断される．つまり，無駄なデータを転送している状態であると示している．そこで，この無駄な転送を省くために小さな読み込み量に合わせて RPC バッファサイズを小さくしたとする．この場合，RPC1 回につき 1 個の少量の領域が転送される．従って，読み込む領域の数だけ RPC が発行されることになり，その都度少なくとも回線遅延分の時間がかかる．一方で回線帯域が十分にあるならば回線遅延と RPC 発行回数の積よりも短い時間で，ランダムアクセスされる領域を 1 回の RPC で転送できる可能性がある．この判断をするためには，図 6 の 2 段目のように読み込まれなかった小さな領域を無視して，シーケンシャルアクセスであるとみなすようにすれば良い．無視すべき領域のサイズは，次に示すように回線帯域と遅延に応じて変わる．

ここで，回線帯域と遅延を乗算したデータ量を帯域遅延積と呼ぶ事にする．読み込まれなかった RPC バッファ領域のうち，帯域遅延積よりも小さなものを無視すれば，1 個 1 個転送すると回線遅延による RPC 応答待ち時間がデータ転送時間を大幅に上回ってしまう状況において，読み込みはシーケンシャルアクセスであるとみなされる．この結果，その都度 RPC を発行することなくまとめて転送出来るようになる．これは data sieving[7]と呼ばれるアクセス方法で，これを実現するためには n を以下のように決める．

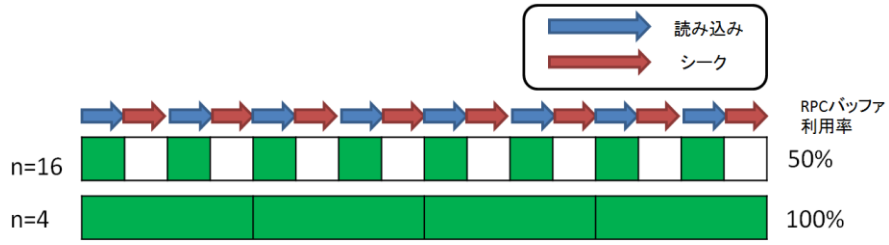


図 6　ブロックサイズによる RPC バッファ利用率の違い

$$\text{帯域遅延積} = \text{ブロックサイズ} = \frac{\text{RPC バッファサイズ}}{n}$$

この n で 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 に示す方法で RPC バッファ利用率を求め，アクセスパターンを次のように判別する．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．この判別結果に基づき，1 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する．しかしながら，直前の結果だけで判断すると，一時的なアクセスパターンの影響を大きく受けることになる．そこで，過去 m 回分の RPC バッファ利用率の平均値を考える．一方で，RPC バッファサイズが非常に大きい状態でのアクセスパターンの変化は，性能に与えるペナルティが非常に大きい．したがって，このとき出来る限り素早く RPC バッファサイズに反映するほうがいいと考えられる．そこで，RPC バッファサイズにより m を変化させることを考え，LARGE_BUFSIZE を上回ったら $M_{small}$，LARGE_BUFSIZE 以下であれば $M_{normal}$ とする．

この RPC バッファ利用率は RPC バッファのうちどれ程のデータが本当に転送すべきであったかを表した数値であるから，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 RPC バッファサイズ増加率で乗算して拡張する．利用率の高低の判断は閾値によって行い，その値は高い方が $U_{high}$ で低い方が $U_{low}$ である．RPC バッファサイズの変更手順を，以下に疑似コードで示す．

Vol.2010-ARC-192 No.6
Vol.2010-HPC-128 No.6
2010/12/16

```
if (RPC バッファ利用率 > 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 < = U_low )
        RPC バッファサイズ *= RPC バッファ利用率
end if
```

以上のようにすると，ランダムアクセスによって RPC バッファサイズが小さくなり，アプリケーションが 1 回に読み込むデータ量と RPC 回数バッファサイズが一致した場合に利用率は 100%を示す．この場合の RPC バッファサイズは最適値であるが，同時に RPC バッファ利用率も高いのでバッファサイズ増加の条件に合致する．この条件に従って RPC バッファサイズを増加させると，最適値を外れた事により RPC バッファ利用率が低下し，再び RPC バッファサイズを縮小する．そして，ランダムアクセスが続く限りは以後この拡大と縮小の繰り返しとなる．一方で，拡大後にも利用率が低下しなければシーケンシャルアクセスであり，そのまま RPC バッファサイズを増加させれば良い．

しかしながら，アクセスパターンの継続性を考慮すると，頻繁に RPC バッファサイズが最適値を外れる事は好ましく無く，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，以後 p 回の RPC ではバッファサイズを拡大しないようにする．p が大きければランダムアクセスの継続性が重視され，小さければシーケンシャルアクセスへの変化に対する対応が迅速になる．

### 4.3 実装した手法の挙動

これまでに述べた，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の手法を実装して動作させた結果を示す．

実行に当たっては，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 のべき乗としているので，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 7 は本手法の挙動を示すためのグラフで，1GB のシーケンシャルアクセスの直後に，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を行った場合の利用率と RPC バッファサイズの推移を示している．横軸は RPC の回数である．

前半のシーケンシャルアクセスにおいては，RPC バッファ利用率が 100%を示し，これによって RPC バッファサイズが増加することが分かる．また，途中でストライドアクセスに切り替わると同時に，RPC バッファ利用率が低下して RPC バッファサイズが縮小していくと，RPC バッファ利用率が再び 100%を示すようになる．そして，再び RPC バッファサイズを拡大するが，ストライドアクセスであるために RPC バッファ利用率が低下してしまう．この状態を検知すると，4.2 で述べた通りに RPC バッファサイズを元に戻し，しばらくそのままにする．その結果，同じ変化を繰り返しているのがグラフ後半の挙動である．

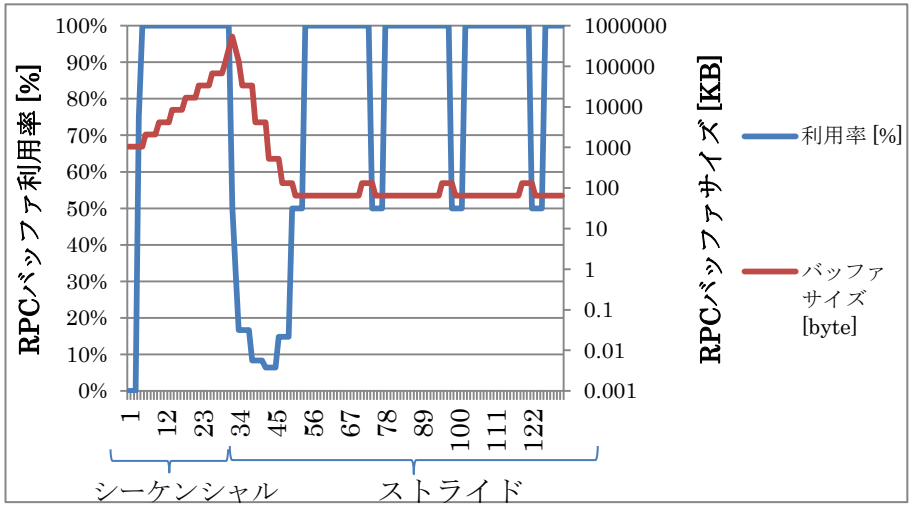


図 7　シーケンシャルからストライドアクセスに切り替わる場合の挙動

### 4.4 各ケースにおける性能の比較評価

遠隔ファイルアクセスにおけるいくつかのケースについて，Gfarm と提案手法の比較評価を行った．提案手法の性能測定にあたっては，Gfarm と同じ方法でファイルアクセスを行うクライアント・サーバ型のプログラムを作成した．このプログラムと Gfarm は同一条件下において同じ性能を示すことを確認している．提案手法の性能測定にあたっては，RPC バッファサイズを 64KB から 512MB の間で動的に変更するよう設定した．

評価環境は，2 台のコンピュータを Gigabit Ethernet で接続したものであり，サーバ側のストレージは SATA 接続 7200 回転ハードディスク 2 台を RAID0 構成で接続している．これは，ローカルストレージがボトルネックにならないようにするためで，実測 170MB/秒のシーケンシャル読み込みを確認している．また，一部のアクセスパターンについてはサーバストレージに SSD を採用した測定結果も掲載している．

Vol.2010-ARC-192 No.6
Vol.2010-HPC-128 No.6
2010/12/16

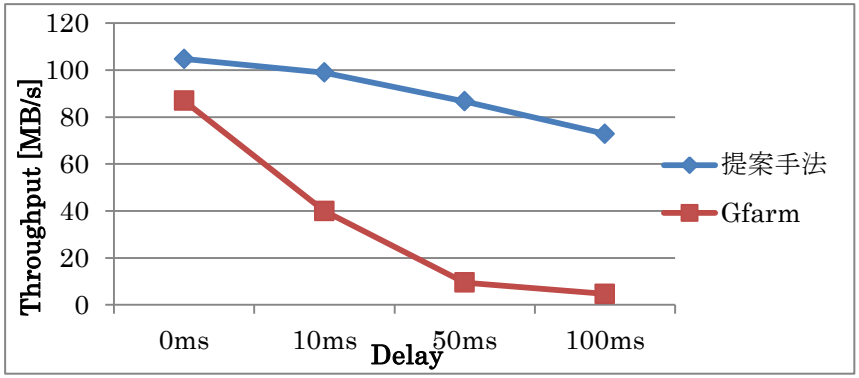


図 8　シーケンシャルアクセスにおける Gfarm と提案手法の比較

#### 4.4.1 シーケンシャルアクセス

シーケンシャルアクセスはファイルを連続的に扱うため，RPC バッファサイズを大きくすることでオーバーヘッドを低減し，高遅延環境においては応答待ちによる速度低下を押さえることが出来る．

図 8 はその評価で，サーバ上の 6GB のファイルにシーケンシャルアクセスした結果である．比較対象の Gfarm は現在の標準設定（RPC バッファサイズ 1MB）である．図から，遅延が増大しても提案手法は Gfarm よりも高い性能を維持していることが分かる．ところが，本来バッファサイズが 512MB まで拡大したのであれば，100ms 遅延環境においてはバンド幅の 97.6%の速度が出るはずであるが，実際にはそれ程の性能は出ていない．この原因を検証したところ，16MB 以上の RPC のバッファサイズにおいて連続してデータを転送していると，クライアントからのリクエストがサーバに届くまでに 200ms～400ms を要するケースが散見され，これによって性能が低下していることが分かった．また，サーバがデータを送信し終えてからクライアントがすべてのデータを受け取るまでの時間にも変動があり，RPC によるサーバからクライアントへの返送データのうち，最後のパケットが欠落することにより再送制御が遅れる事で発生する RTO による遅延も存在した．上記測定結果は RTO 遅延対策として転送の最後に 1 パケット以上のパディングを行った場合であるが，対策を行わない場合には RPC の完了までに 2.5 秒を要するケースもあり，測定結果も非常に不安定で性能についても 15%程度低下が見られた．RTO 遅延についてはパディングによって解消できたが，クライアントからのリクエストがサーバに届く間に発生する不規則な遅延の原因はまだ詳しくは分かっていない．これについては TCP の輻輳制御の影響を受けていると推測している．

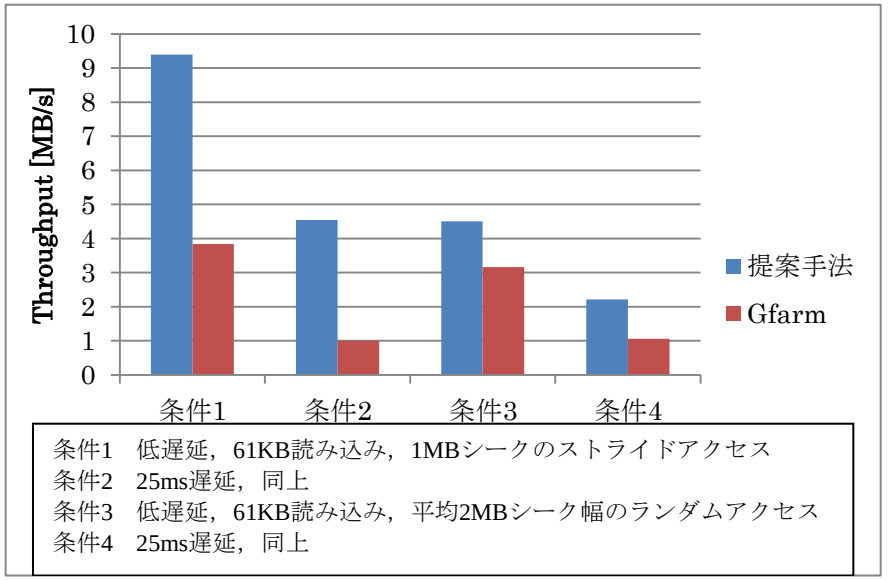


図 9　ランダム・ストライドアクセスの性能比較

#### 4.4.2 ストライドアクセス・ランダムアクセス

図 9 は LAN 内を想定した低遅延環境（50μ秒)および国内拠点間を想定した 25ms 遅延環境におけるストライドアクセス・ランダムアクセスの性能測定結果である．条件はそれぞれ図中に記したとおりである．

すべての条件において，提案手法が高い性能を示していることが確認され，一部では 350%の性能向上が見られた．

条件 1 と条件 2 は，遅延が異なるだけであるが，内部での挙動はそれぞれで全く異なる．これは，4.1 で述べたように遅延と回線速度に応じて利用率を測定するためのブロック数が変化するからである．低遅延環境においては各ブロックのサイズは非常に小さくなり，RPC バッファサイズは最小設定値である 64KB にまで縮小される．一方で 25ms 遅延環境においては，各ブロックのサイズは Gigabit Ethernet が 25ms の間に転送出来るデータ量である約 3MB になる．従って，61KB の読み込みと 1MB のシークを繰り返した場合，すべてのブロックが使用済みとしてマークされ，シーケンシャルアクセスと同様の挙動となり，RPC バッファサイズは最大値まで拡大する．図 10 はその動作を簡単に表しており，遅延が 1 の場合はアプリケーションが要求したデータ量に基づく正確な RPC バッファ利用率が測定されているが，遅延が 4 の場合にはブロックサイズが 4 倍になり，シーケンシャルアクセスと見なすようになる．

Vol.2010-ARC-192 No.6
Vol.2010-HPC-128 No.6
2010/12/16

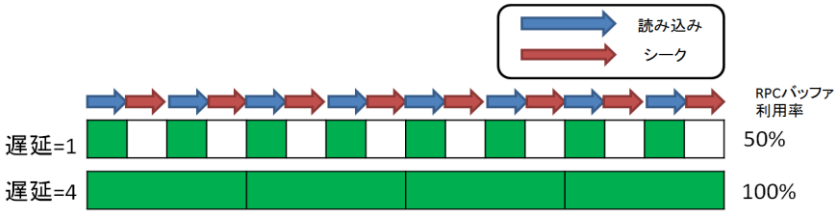


図 10　遅延による利用率測定結果の変化

従って，低遅延環境ではストライドアクセスの読み込みにつき毎回 RPC を発行してデータを転送し，高遅延環境では 1 度の RPC で多くの領域がまとめて転送されるようになる．後者が 4.1 で述べた data sieving を行っている状態で，回線遅延が大きく RPC を発行した際の待ち時間が長い場合には有用である．このように，単にランダムアクセスなら RPC バッファサイズの縮小，シーケンシャルなら拡大するだけでは不十分であるが，この手法はそのような点もカバーすることが出来ている．

本提案手法の特長は，回線遅延や帯域，アクセスパターンに応じて最適な RPC バッファサイズを選択することが出来，その処理をシンプルな一つの仕組みで実現している点である．

#### 4.4.3 SSD をサーバストレージに用いた場合の性能評価

ファイルサーバ側のストレージに SSD を用いた場合の遠隔ファイルアクセスの性能を評価した．SSD は記憶媒体にフラッシュメモリを使用しており，ヘッドを物理的に移動させてデータを読み書きする HDD よりも，指定したデータを読み込むまでの応答時間が非常に短い特徴がある．従って，ランダムアクセスにおける性能が HDD に比べ高い．これにより，RPC を用いた遠隔アクセスの完了時間も短くなり HDD とは違う挙動を示すと考えられる．そこで，その性能評価を行った．

図 11 はサーバストレージ Intel の MLC 型 SSD である SSDSA2MH080G1GC を用いた場合と HDD を利用した場合について，61KB 読み込み平均 2MB シークのランダムアクセス速度を測定したグラフである．特に低遅延環境におけるランダムアクセスで効果が大きく，5.5 倍の性能向上が見られる．この理由は，低遅延環境においては RPC バッファサイズが読み込みサイズに近い 64KB にまで縮小され，回線遅延がほとんど無い事から RPC の応答時間も非常に短く，サーバ側のストレージの応答速度がボトルネックとなるためである．SSD の短い応答時間がこのボトルネックを解消し，性能向上につながった．一方で高遅延環境においては大きな差は見られない．これは，サーバ側のストレージの応答時間よりも， RPC の応答時間や data sieving によって拡大したバッファサイズの転送のために必要な時間の方が十分に大きいからである．

以上のように，サーバストレージに SSD を用いた場合，同期型 RPC の遠隔ファイルアクセス（ランダム）の性能は低遅延環境において大幅に向上することが分かった．

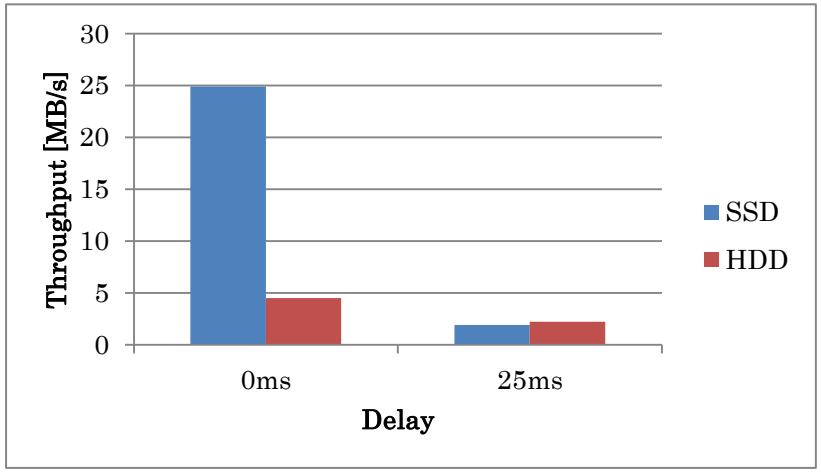


図 11　SSD と HDD における遠隔ランダムアクセスの性能比較

### 4.5 本提案手法の実装によるオーバーヘッドの考慮

この手法を実現するにあたって必要なリソースは，RPC 回数バッファ利用率を測定するためのテーブルと平均値を取るために必要ないくつかの履歴用バッファである．テーブルはブロック数を最大 4096 とすると 512 バイト必要で，履歴用のバッファは倍精度浮動小数点の数値を 8 個として 64 バイトであり，メモリ使用量は合計しても 576 バイトに過ぎない．これは，その他の通信用バッファに比べれば非常に小さい．また処理についても RPC バッファ利用率の測定はテーブル中で 1 であるビット数をカウントするだけであり，平均の計算も数回の加算と除算で済む．実際にこれらの処理に要する時間を測定したところ，RPC1 回あたり 160 $\mu$ 秒に過ぎず，データの転送時間に比べれば非常に小さなものであった．

これらの実装を組み込むことによる効果は 4.4 で述べた通りである．これが非常に小さなオーバーヘッドで実現できることから，本論文の提案する手法は性能向上に有効であると言える．

## 5. まとめ

### 5.1 評価

本論文で示した提案手法は同期型 RPC を用いる遠隔ファイルアクセスにおいて，RPC バッファサイズの動的な最適化という課題を，ごくわずかなオーバーヘッドで達

Vol.2010-ARC-192 No.6
Vol.2010-HPC-128 No.6
2010/12/16

成し，様々な条件下で性能向上を果たせることが確認できた．その性能向上は条件によっては 4 倍以上にも達し，この手法による性能低下は今のところ確認されていない．従って，この手法を遠隔ファイルアクセスに取り入れることは，非常にメリットが大きい．また，SSD 等の新しいデバイスの登場により，ローカルファイルアクセスのレスポンスは非常に高速化している．このような新しいデバイスの性能を広域分散環境においても発揮させるためには，遠隔ファイルアクセスの高速化が必須であり，本提案手法はそれにあたっての要求を満たすことが出来る．

### 5.2 今後の課題

性能評価の章でも述べたが，TCP/IP プロトコルの性質によって十分な性能が発揮できていないケースが見られたため，OS の TCP/IP スタックの実装を含め，下位のレイヤーにおける現象を正確に見極める必要が生じている．これは今後の課題である．

また，本論文では同期型の RPC を対象としていたが，今後は非同期の方式も検討し，更なる性能向上を目指す．特に，アプリケーションが非同期 I/O を利用している場合，遠隔ファイルアクセスのプロトコルレベルでも非同期の方式をサポートすることで，回線遅延の影響をさらに抑えることが可能である．

## 6. 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) Ann L. Chervenak, Ian T. Foster, Carl Kesselman, Charles Salisbury and Steven Tuecke, "The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets," J. Network and Computer Applications, Vol.23, No.3, pp.187-200, 2000

2) Osamu Tatebe, Kohei Hiraga, and Noriyuki Soda, "Gfarm Grid File System," New Generation Computing, Ohmsha, Ltd. and Springer, Vol.28, No.3, pp.257-275, DOI: 10.1007/s00354-009-0089-5, 2010

3) B. Callaghan, B. Pawlowski, and P. Staubach, "NFS Version 3 Protocol Specification," RFC 1813, 1995

4) John H. Howard et al., "Scale and performance in a distributed file system," ACM Trans. Computer Systems, Vol.6, No.1, pp.51-81, 1988

5) M. Satyanarayanan et al., "Coda: A Highly Available File System for a Distributed Workstation Environment," IEEE Trans. Computers, Vol.39, No.4, pp.447-459, 1990

6) Lauro Costa and Matei Ripeanu, "Towards Automating the Configuration of a Distributed Storage System," 11th ACM/IEEE International Conference on Grid Computing (Grid 2010), pp.201-208, 2010

7) Rajeev Thakur, William Gropp and Ewing Lusk, “Data Sieving and Collective I/O in ROMIO,” In Proceedings of the Seventh Symposium on the Frontiers of Massively Parallel Computation, IEEE Computer Society Press, pp.182-189, 1998