広域分散ファイルシステムのための適応的な先読み手法

堀内美希[†]　田浦健次朗[†]

分散環境でデータ集約的計算を行う際に重要な役割を果たす分散ファイルシステムであるが，高遅延環境でのファイルアクセスでは遅延の影響を受け，大幅にスループットが下がってしまうことがある．それを回避するために，アプリケーションに変更を加えず適応的に実行可能なアクセスデータの先読み手法を提案する．提案手法により，評価に用いた高遅延広帯域環境下でのファイルアクセスでは，シーケンシャルアクセスで約 700 ～ 800%，ストライドアクセスで約 300 ～ 400%の読み込みスループット向上を確認することができた．

An Adaptive Prefetching Method for Distributed File Systems

Miki Horiuchi[†] and Kenjiro Taura[†]

Distributed file systems play an important role for data intensive computation but current systems often fail to achieve good throughputs in high latency environments. To achieve a good access throughput, we propose a data prefetching method that can be adatively applicable without any modification to applications. In the evaluation environment with high latency and wide bandwidth, the proposed method achieved the “read” performance improvement of around 700 ～ 800% in sequential access and of around 300 ～ 400% in stride access.

1. はじめに

近年，マルチコアアーキテクチャの普及，解析対象データサイズの増大，分散ノードを扱うための環境の整備等により，分散環境で並列計算を行う機会が増えている．Montage[1] 等の，分散環境で大規模なデータを扱って複雑なデータ解析を行うワークフローアプリケーションも登場しており，このような分散計算を行う際には共有ファイルを，ローカルファイルを扱うのと同じ通常のファイルシステム API を用い透過的にアクセスできるとユーザからみて扱いやすい．そのため分散環境でのワークフローアプリケーション実行の際には，よく分散ファイルシステムが基盤システムとして用いられる．しかし，複数拠点間にまたがったデータ解析の場合では，高遅延環境で分散ファイルシステムを基盤としたファイルアクセスを行うと，遅延の影響を受けて高スループットを達成できない場合がある．これを回避するために本稿では広域分散ファイルシステムのためのデータ先読み手法を提案する．データ先読みを行うときは，先読みが外れるリスクも考えてネットワーク性能を限界まで引き出せる最小の大きさで先読みを行うとよいが，その大きさは環境に応じて変化する．本稿では，この大きさの自動調整を既存の TCP 上でシンプルに行う手法を提案する．

† 東京大学大学院情報理工学系研究科
The University of Tokyo

2. 関連研究

本研究では分散ファイルシステムの高遅延広帯域環境におけるファイルアクセスを，適応的なデータ先読みによって高速化する手法を提案する．関連する研究ではこれまでに，遠隔ファイルアクセスを Non-blocking RPC を用いて高速化する手法が提案されている[2]．ただし，先読みを行う量を自動調整する手法は提案されておらず，ユーザが手動で調整を加えなければならない．

広域分散ファイルシステムには Gfarm[3]，HDFS[4] 等が広く用いられており，これらの分散ファイルシステムではレプリケーションを行い，一番遅延の小さい場所のレプリカにアクセスする等の高遅延環境でのアクセスに対応するための手法が提案されている．GPFS[5] では MPI-IO によるデータ先読みを行いファイルアクセススループットを向上させる研究が行われている[6]．また，PVFS[7] を用いて MPI プログラムのコードを解析し，ファイルアクセスに関する部分を別スレッドの処理へ切り出し，更に事前に実行しておくことで先読みを実現する研究も行われている[8]．い

ずれの場合も，先読みは並列アプリケーションを変更することでアプリケーションが行うものであり，先読みサイズはアプリケーション内で決定される．本研究では，既存のアプリケーションを書き換えることなく，TCP の標準的な用い方で得られる情報を元に，このサイズを自動決定する手法を提案する．

また，高遅延環境でのファイルデータ転送手法として，GridFTP[9)10)] の研究が行われている．しかしこれは FTP の機能を目的としており，ファイル全体の転送しか行わない．本研究ではファイルシステム API をサポートし，それに応じたデータ転送を行う．

3. アクセスパターン検出手法

広域分散ファイルシステムに限らず，ファイルシステムのデータ先読みのためにはデータアクセスパターンを検知し，今後必要とされるであろう部分を予測することが重要である．本節では提案手法におけるファイルアクセスパターンの検知・分類アルゴリズムに関して述べる．はじめにアクセスパターン検出のために，ファイルシステムのレイヤで保持しておく情報（パラメタ）に関して述べる．次に，アクセスパターンを，シーケンシャルアクセス，ストライドアクセス，ランダムアクセスの 3 つの種類に分類し，その 3 つの種類のファイルアクセスをファイルアクセス中に検出する手法を提案する．その後，そのアルゴリズムによるアクセスパターン検出を用いた先読み要求の流れに関して述べる．

3.1 アクセスパターン検出のために保持しておくべき情報

分散ファイルシステムのレイヤでアプリケーションのファイルアクセスに関して共通して得られる情報は，各 read の offset, size のみである．これらから適応的に今後アクセスされそうな範囲を先読みするために，開かれているファイル毎にパラメタとして保持しておくべき情報を以下に述べる．

mode 現在アプリケーションがどのようなファイルアクセスモードと判断されているか．シーケンシャルアクセスモード，ストライドアクセスモード，ランダムアクセスモードの三つの種類がある．

stride_read_size ストライドアクセスのときに，シーケンシャルに読み込むと推定されるデータ量．

stride_seek_size ストライドアクセスのときに，シーケンシャルな読み込みのあと，シークを行うデータ量．

last_seq_read_start 最後にシーケンシャルアクセスがはじまったファイル内 offset．各 read で前回の read の続きを読んだ場合は更新されず，そうでない場合にその read の offset が入る．

last_read 一つ前に呼び出された read で読んだデータ末尾の offset．

Algorithm 1 シーケンシャルアクセスモードのときのアクセスパターン検出アルゴリズム

```
if offset == last_read then
  mode = シーケンシャルアクセスモード
else if offset > last_read then
  mode = ストライドアクセスモード
  stride_read_size = last_read - last_seq_read_start
  stride_seek_size = offset - last_read
else
  mode = ランダムアクセスモード
end if
```

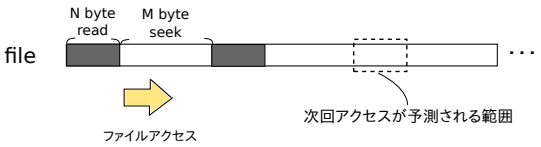

図 1 ストライドアクセス
Fig. 1 Stride Access

3.2 各モード中のアクセスパターン検出アルゴリズムと先読み

次に，前節で述べたパラメタと，アプリケーションからの各 read の offset と size を元にアクセスパターンのモード（パラメタの mode）を変更しつつ，そのモードに合わせた先読みを行うアルゴリズムに関して説明する．ファイルを open したときのモードはシーケンシャルモードからはじまり，last_seq_read_start, last_read の値は 0 に設定しておく．

3.2.1 シーケンシャルアクセスモード

現在の先読みモードがシーケンシャルアクセスモードである場合には，**Algorithm 1** に示したアルゴリズムに従い，今回の読み込み offset と last_read の比較によって次のアクセスパターンを推定する．このモードから他のモードに変わらない間は，現在アクセスしてる部分からシーケンシャルにそのままアクセスが続くことを予測し，シーケンシャルな先読み要求を行う．

3.2.2 ストライドアクセスモード

シーケンシャルアクセスモードに続いて，ストライドアクセスモードのときのアクセスパターン検知と先読みアルゴリズムを **Algorithm 2** に示す．本提案手法でストライドアクセスとして扱うのは，一定の大きさ N byte のデータを読み込み，一定の大きさ M byte のデータを seek して飛ばすアクセス（**図 1**）のみである．現在の提案手法では，このアクセスパター

Algorithm 2 ストライドアクセスモードのときのアクセスパターン検出アルゴリズム

```
if offset == last_read then
  if (offset + size) が次にアクセスが予測される部分を越える then
    mode = シーケンシャルアクセスモード
  else
    mode = （引き続き）ストライドアクセスモード
  end if
else if last_read, last_seq_read_start, offset の間隔がストライドアクセスパラメータにマッチ then
  mode = （引き続き）ストライドアクセスモード
else
  mode = ランダムアクセスモード
end if
```

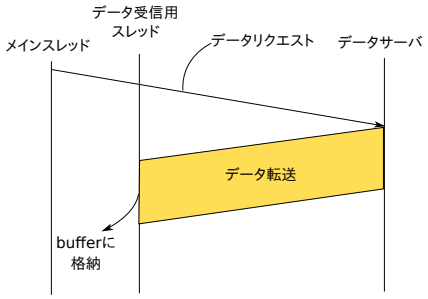

図 2 データ受信用スレッドとデータリクエスト
Fig. 2 A Thread Receiving Data and a Data Request

ンから少しでもずれてしまうとストライドアクセスとは判断しない．

ストライドアクセスモードのときは，このままのサイズの read と seek を繰り返した場合にアクセスすることになる部分を計算し，先読み要求を行う．このとき，パラメタ stride_read_size，stride_seek_size，last_seq_read_start を用いる．

3.2.3 ランダムアクセスモード

ランダムアクセスモードのときは各読み込みの offset が last_read と一致したとき，つまり前回の read の続きにアクセスした場合のみシーケンシャルアクセスモードに切り替える．またこのモードでは将来読み込まれる部分を予測することが困難であるため，先読み要求は行わない．

3.3 アクセスパターン検出と先読み要求の流れ

分散ファイルシステムでは多くの場合，ファイルはある大きさのブロック単位（これをブロックサイズと呼ぶ）で管理され，データ転送もこのブロックの単位で行われる．このブロックはアプリケーションから意識されることはない．ここでは提案手法におけるファイル読み込みの流れに関して述べる．まず，ファイルを開いたときに目的のファイルを持っているサーバ

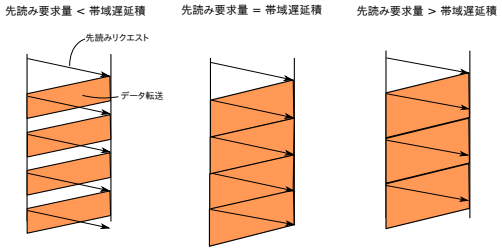

図 3 先読み要求量と帯域帯域遅延積の関係
Fig. 3 Relationship of the Prefetch Request Size and the Bandwidth-Delay Product

（データサーバとする）とコネクションを持つが，それと同時にデータの受信用スレッドを作成する．以降ファイルを読み込むときにはこのデータ受信用スレッドがデータを受け取り，バッファに格納する (**図 2**)．メインスレッドではアプリケーションからの要求を受け取り，アプリケーションに要求されたデータを返すまでの手順を分散ファイルシステムの read システムコールとして実装する．実装した read システムコールの流れを以下に示す．

（Step1） read 要求の offset, size と保持しているパラメータを用いて，アクセスパターンを推定する（3.2 節）
（Step2） 要求されたデータがバッファに存在しない場合のみデータサーバにリクエストを出し，受信用スレッドがデータを受信してバッファに格納するまで待つ
（Step3） Step1 で判断したアクセスパターンとパラメータにより次にアクセスされそうなブロックの集合を決定する．このときのブロック数の決定方法は 4 節で述べる．
（Step4） Step3 で計算したブロックのうち，データサーバに要求を未だ出しておらず，バッファにも格納されていないブロックの転送要求をデータサーバに出す．
（Step5） Step2 で得た要求データを用いて，read システムコールを返す

4. 適応的なデータ先読み量自動調整手法

本節では，3 節で延べたアクセスパターンの分類・検出手法を用い，どの程度の量の先読み要求をデータサーバに出すかを，標準的な TCP の用い方で得られる情報を元に，シンプルに自動決定する手法に関して述べる．

4.1 理想的なデータ先読み要求量

まず本提案手法で理想とする先読みデータ要求量に

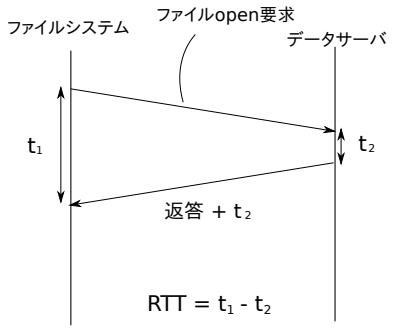

図 4 RTT の算出手順
Fig. 4 The Procedure of RTT Calculation

関して説明する．ここではデータサーバ上でファイルデータがキャッシュに存在していて，ディスクの読み込み帯域に律速されず，データ速度限界はネットワーク帯域に律速されていることを前提とする．分散ファイルシステムでネットワークの性能を極限まで引き出したデータ転送を行うためには，データサーバがあるデータ先読み要求を受け取ってから，次のデータ先読み要求を受け取るまでの間にデータをネットワークに転送し続ける必要がある．これをちょうど実現するためには (RTT × ネットワーク帯域幅)，帯域遅延積分のデータ先読み要求を送信するようにすればよい．帯域遅延積より小さい量を先読み要求量にした場合には**図 3** の左側に示した図のようなデータ転送となり，ネットワークの性能を活かしきれない．逆に帯域遅延積以上のデータ先読み要求を出してしまうと，図 3 の右側に示した図のようなデータ転送となり，先読みが外れたときに前の先読み要求によるデータの転送後，実際にアプリケーションが要求したデータの転送が行われるため，先読みを行うことによるリスクが大きくなってしまう．従って，データ先読み要求量を帯域遅延積とすることによって，ネットワーク帯域を十分に活かしつつ，先読みが外れたときのリスクは最小限とすることができる．この場合，先読みが外れたときの，次データ送信にかかる遅延時間は最悪で RTT となるはずである．(先読み要求を出した直後にアプリケーションが先読みから外れた場所をアクセスした場合)

4.2 先読み量自動調整手法

理想的なデータ先読み要求量を帯域遅延積とした上で，分散ファイルシステムでの帯域遅延積の推定方法に関して述べる．

帯域遅延積を推定するため，まず遅延（RTT）の測定をファイルを開いた際に行う．アプリケーションからあるファイルの open 要求を受け取ったのちに，データサーバに open 要求を送ってからその返答を受け取るまでの時間 t_1 を測定しておき，データサーバ側では open 要求を受け取ってから処理を行い返答を返すまでの時間 t_2 を測定する．そして t_2 を返答と共に送信し，ファイルシステム側で $t_1 - t_2$ を計算することで RTT を得る（**図 4**）．

Algorithm 3 データ受信用スレッドのアルゴリズム

```
while ファイルが開かれている do
  if socket からデータ読み出し可能 then
    帯域推定を行わないモードに変更
  else
    帯域推定を行うモードに変更
  end if
  データのヘッダを受信する
  start_time = now()
  ブロックデータを受信する
  end_time = now()
  if 帯域推定を行うモード then
    推定帯域 = 受信したデータサイズ / (end_time - start_time)
    先読み要求量を推定帯域 × 推定 RTT に設定
  end if
end while
```

次に，帯域推定の手法に関して述べる．データを受信し，帯域推定から，データ先読み要求量を帯域遅延積に設定するまでのアルゴリズムを **Algorithm 3** に示す．帯域推定は，データの受信用に作成したスレッドでデータを受信する際の時間計測により行う．ただし，データ受信時に既にマシンの受信バッファにデータが到着しており，バッファからデータを読み込むだけになった場合は，データ受信にかかった時間から正確な帯域を計算することはできない．しかし，そのような場合は図 3 の左側に示したようなデータ転送の待ち時間は生じてないと判断できるため，ネットワーク帯域を十分に活かした通信が行われており，先読み要求量を増やす必要はない．この場合を検知するために，データ受信の前にそのソケットが読み込み可能であるかどうかをチェックし，既に読み込み可能であった場合には，帯域推定やデータ先読み要求量の変更を行わないこととする．

5. 評　　価

本節では，ここまで述べた提案手法を二つの観点から評価する．まず，3 節で述べたアクセスパターン検知・分類手法を用いたファイルの先読みの効果を評価する．その後，4 節で述べた適応的に先読みデータ量を自動調節する手法の効果を評価する．本評価では分散ファイルシステムを FUSE[11] ベースで独自実装し，評価対象として用いる．この分散ファイルシステムのブロックサイズは 1MB として実装を行っている．ま

表 1 実験マシン基本性能
Table 1 Basic Spec of Machines for Experiments

ノード	CPU	メモリ	OS	NIC
huscs	Intel Xeon E5530 8cores (w/ HT 16cores)	24GB	Linux 2.6.26 (64bit)	Broadcom NetXtreme II BCM57711
tsukuba	Intel Xeon E5620 8cores (w/ HT 16cores)			Intel(R) 10 Gigabit
kyoto	Intel Core2 6400 2cores	4GB		Intel(R) 1Gigabit

表 2 実験ネットワーク環境
Table 2 The Network Environment for Experiments

	物理帯域 [Gbps]	RTT [msec]
huscs - tsukuba 間	10	約 27
huscs - kyoto 間	1	約 38

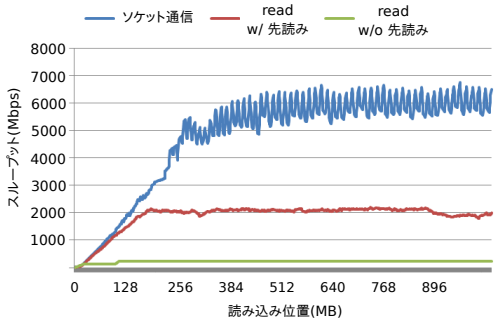

図 5 シーケンシャルアクセスの読み込みスループット (huscs → tsukuba)
Fig. 5 Read Throughput in Sequential Access (huscs → tsukuba)

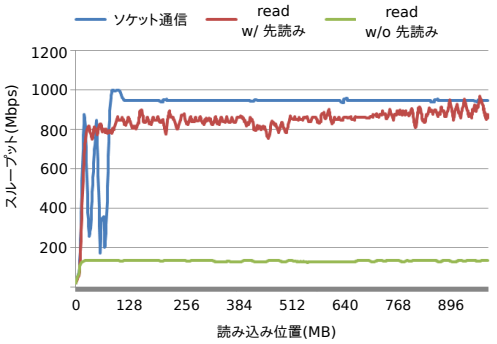

図 6 シーケンシャルアクセスの読み込みスループット (huscs → kyoto)
Fig. 6 Read Throughput in Sequential Access (huscs → kyoto)

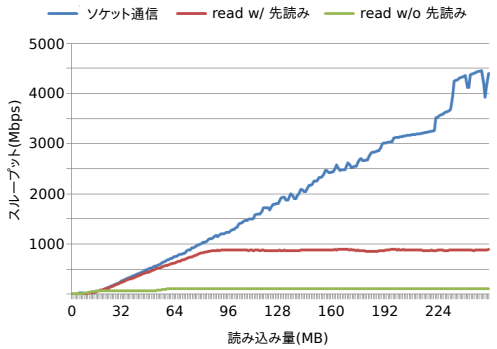

図 7 ストライドアクセスの読み込みスループット (huscs → tsukuba)
Fig. 7 Read Throughput in Stride Access (huscs → tsukuba)

た，評価では 1MB 毎のデータ読み込みにかかる時間を計測するが，分散ファイルシステムを用いたファイルアクセススループットは大きくばらつくため，直近の 5MB のデータを（1MB ずつ）読むのにかかった時間の平均（移動平均）を算出して評価に用いている．

5.1 評価環境

評価では，InTrigger プラットフォーム[12] 中の huscs（北海道大学），tsukuba（筑波大学），kyoto（京都大学）クラスタのマシンを用いる．それぞれのマシンの基本性能を**表 1** に，ネットワーク環境を**表 2** に示す．実装した分散ファイルシステム上にファイルを作成し，tsukuba, kyoto からそれを読み込む．ファイルはデータサーバ上のキャッシュに存在する状態で行う．高遅延環境での通信をする際，Linux 標準の TCP 送受信バッファサイズでは性能が十分に出ないため，各マシンの TCP 送信受信バッファを十分に大きくとって行う．使用する TCP 輻輳制御アルゴリズムは Linux 標準の TCP-CUBIC である．

5.2 アクセスパターン検知による先読み効果の評価

アクセスパターンの分類・検出手法による効果を測定するために，シーケンシャルアクセス，ストライドアクセスそれぞれに関して，ファイルの読み込みスループットを測定した．読み込むファイルは 1GB の大きさのファイルである，シーケンシャルアクセスではファイルを 1MB ずつ順番に読み込み，ストライドアクセスでは 1MB 読み込み，3MB 読み飛ばすという処理をファイルの末尾まで繰り返し行い，1MB 読み込むごとのスループットを測定した．評価は，実装した分散ファイルシステムの先読みを無効化したもの，先読みを有効にして手動設定により先読み要求量を決定したもの，単にソケット通信でデータを送り続ける

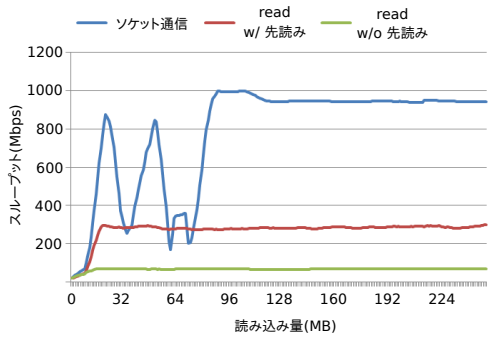

図 8 ストライドアクセスの読み込みスループット (huscs → kyoto)
Fig. 8 Read Throughput in Stride Access (huscs → kyoto)

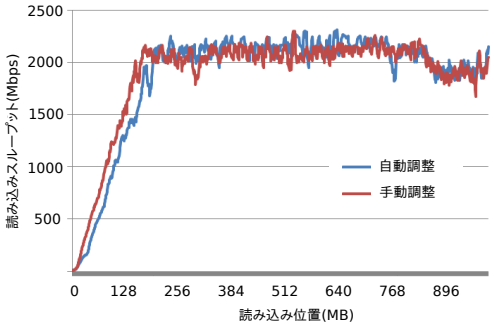

図 9 先読み要求量自動調整手法の評価（huscs→tsukuba）
Fig. 9 Evaluation of the Prefetch Request Size Auto-tuning Method (huscs → tsukuba)

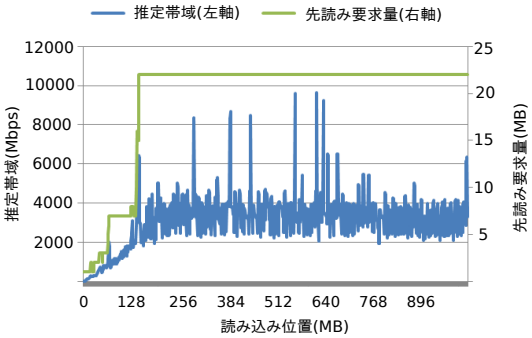

図 10 推定帯域と先読み要求量（huscs→tsukuba）
Fig. 10 Estimated Bandwidth and Prefetch Request Size (huscs → tsukuba)

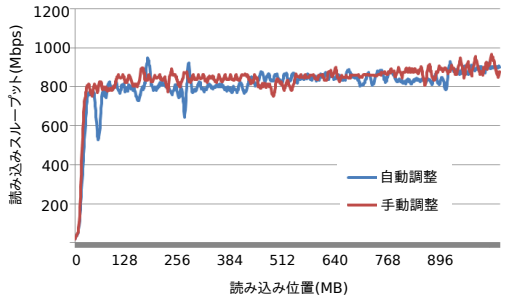

図 11 先読み要求量自動調整手法の評価（huscs→kyoto）
Fig. 11 Evaluation of the Prefetch Request Size Auto-tuning Method (huscs → kyoto)

簡易プログラムの比較で行う．ソケット通信のみのものは理論上可能な最大スループットを与えるためのものである．この評価のシーケンシャルアクセスの結果を**図 5**，**図 6** に，ストライドアクセスの結果を**図 7**，**図 8** に示す．

まずシーケンシャルアクセスに関して，先読みを行うことによってどちらの環境でも読み込み性能が大幅に改善されていることが分かる．しかし図 6 ではソケット通信の結果で示された，理想スループットに近いアクセススループットが得られているが，図 5 ではこのスループットの 1/3 程度のスループットしか得られていない．また，ストライドアクセスに関しては，先読みを行うと大幅なアクセス性能向上は確認できたが，シーケンシャルアクセスのときの読み込みスループットには達していない．うまくアクセス予測ができていればシーケンシャルアクセスとほぼ同じスループットが出るはずであるが，調査を行った結果これは FUSE の先読み機能が原因であると分かった．デフォルト設定の FUSE では，例えばアプリケーションからファイルの先頭から 1MB のサイズの read 要求を出したとき，FUSE は該当 1MB の次の 128kB のデータも要求を出す．評価用分散ファイルシステムは 1MB の大きさのブロックでデータを転送・管理するので，結果的にこの場合はファイル先頭の 1MB とその次の 1MB の転送が行われることとなり，スループットが約半分になっていた．

5.3 適応的な先読みデータ要求量調整手法の評価

次に，本稿で提案する適応的な先読みデータ要求量を自動調整する手法に関して評価を行う．提案手法を用いて先読みデータ量を自動調整した際に，手動で適切とする値にした場合と変わらない読み込み性能を達成できるか，また先読みが外れたときのリスクは想定した範囲内かという二つの観点から評価を行う．

5.3.1 先読み要求量自動調整による効果

提案手法による先読み要求量の自動調整手法を評価する．評価は 5.2 と同じく 1GB のファイルを，シーケンシャルに読み込んだときの読み込みスループットを測定する．評価対象として，先読み要求量を手動で

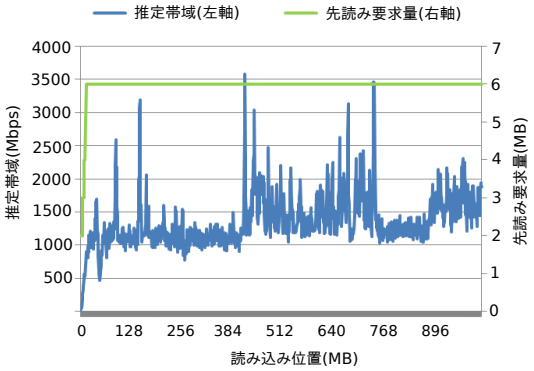

図 12 推定帯域と先読み要求量（huscs→kyoto）
Fig. 12 Estimated Bandwidth and Prefetch Request Size (huscs → kyoto)

調整した場合と，自動調整を行った場合を比較する．また自動調整を行ったときのデータ受信用スレッドによる推定帯域と，そこから計算した先読み要求量も測定を行う．huscs → tsukuba で評価を行った結果を**図 9**，**図 10** に，huscs → kyoto で行った結果を**図 11**，**図 12** に示す．

どちらの環境でも提案手法による自動先読み量調整手法は，管理者が手動でパラメタを設定した場合と比べて，大きくアクセススループットを落とすようなことはなかった．ただし，huscs → tsukuba では先読み要求量が増加するのが比較的遅く，ファイルの先頭付近のアクセススループットが少し低い．このような広帯域環境で小さなファイルを読み込む場合などは広帯域なネットワーク性能を活かした通信を行うのは難しく，ファイル先頭を読み込むときの帯域推定アルゴリズムに関してはもう少し改良の余地があるように思える．

5.3.2 先読みモード切り替え時の性能

次に先読みを行う際に生じるリスクに関して，提案手法を用いて先読み要求量を自動調整する場合と，先読みを無効化した場合を比較して評価する．先読みが外れたのちのデータ転送の遅延を測定するため，1GB の大きさのファイルを用意し，はじめに 512MB をシーケンシャルにアクセスし，その後ファイル末尾 1MB にアクセスを行うときに読み込みにかかる時間を計測する（**図 13**）．実験を huscs → tsukuba で行った結果を**図 14**, huscs → kyoto で行った結果を**図 15** に示している．

まずどちらの環境においても提案手法を用いて先読みを行った場合のほうが，連続領域をアクセスしている途中で他の場所をアクセスしたときのアクセスにかかる時間が短くなっている．提案手法では最悪 RTT 分の遅延を想定していたが，実際は先読みを行っているときのほうがネットワーク帯域が出ている状態であるため，先読みを外し無駄なデータ転送が生じてもこの実験環境では短い時間でデータ転送が行えている．

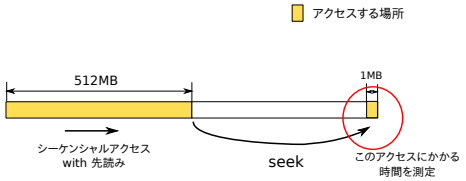

図 13 先読みモード切り替えの性能評価
Fig. 13 Performance Evaluation of Switching the Prefetching Mode

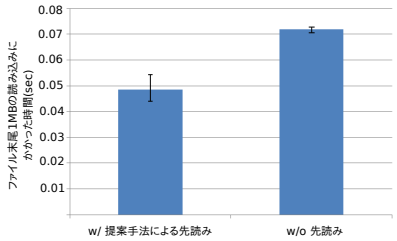

図 14 アクセスモード切り替え時のリスク評価（huscs→tsukuba）
Fig. 14 Evaluation of a risk in Switching Access Mode (huscs → tsukuba)

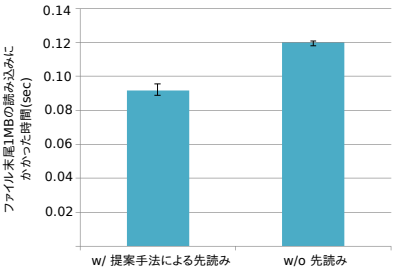

図 15 先読みモード切り替え時のリスク評価（huscs→kyoto）
Fig. 15 Evaluation of a risk in Switching Access Mode (huscs → kyoto)

5.4 複数プロセスからの同時アクセス性能

最後に提案手法による適応的な先読みを，複数プロセスで実行した場合のファイル読み込みスループットを測定する．1GB のファイルを用意し，あるプロセス（プロセス A とする）はシーケンシャルに 1MB ずつファイルを読み込む．先頭 512MB の読み込みが終わった直後にもう一つプロセスを立てて（プロセス B とする），プロセス A と同時に同じファイルを読み込む．プロセス B はファイルのオフセット 512MB の位

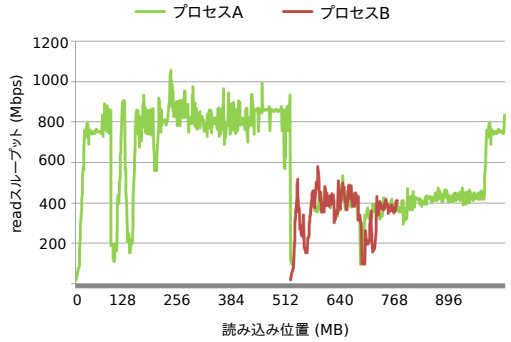

図 16 複数プロセスでの読み込みスループット
Fig. 16 Read Throughput with Multiple Processes

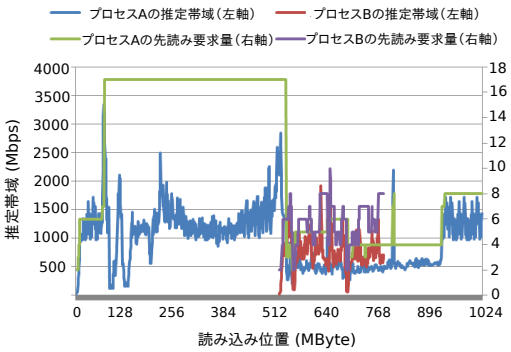

図 17 複数プロセスからの推定帯域と先読み要求量
Fig. 17 Estimated Bandwidth and Prefetch Request Size by Multiple Processes

置から 256MB を読み込んだのちに終了する．これらの読み込みの際に 1MB 毎に読み込み時間を測定し，そこからアクセススループットを算出したものを**図 16** に示す．また，この実験中の提案手法による帯域推定と先読み要求量の推移を**図 17** に示す．

プロセス A と B の間では同時アクセスが行われている間ほとんどスループットが同じ状態を遷移しており，限られた帯域を共有してデータ転送が行えていることが分かる．しかし，プロセス B が読み込みを終了してからプロセス A が元のスループットまで回復するまでにかなりの時間を要している．このことからも一旦スループットが落ちたあとに先読み量を増やしていく部分のアルゴリズムは改良の余地が残っていると思われる．

6. おわりに

本論文では，分散環境での並列計算の基盤システムとして用いられる広域分散ファイルシステムを用いた高遅延環境でのアクセスでも，ネットワークの帯域を最大限に活かした読み込みスループットを得るための適応的な先読み手法を提案した．アプリケーションやユーザから特別な情報を得ることなくアクセスパターンを分類・検出し，先読みを適応的に低リスクで行う手法に関して述べ，評価を行った．その結果，提案手法を用いると，評価に用いた高遅延広帯域環境でのシーケンシャルアクセスの読み込みスループットを約 700 ～ 800%，ストライドアクセスの読み込みスループットを約 300 ～ 400%向上させることができた．

今回提案したアクセスパターン分類手法は実装やアルゴリズムの簡単化のため，ストライドアクセスに対して柔軟性に乏しく，少し予測と外れたところをアクセスするだけで全く先読みを行わないという手法で，一定の決まった read, seek サイズでアクセスを行うアプリケーションには効果的だが，少し違ったアクセスをすると高遅延環境で読み込み性能が出ない．そのため今後アクセスパターンの検知アルゴリズムとして，もう少し柔軟性に富んだアルゴリズムを考案したい．また，今回の実験は国内のインターネット環境で実験を行っているためそのようなことはなかったが，RTT や利用可能帯域の変動する環境において本提案手法を適用する場合には少なくとも RTT を一回だけの計測にはせず，適度に測り直して調整する必要があると思われる．その他，本論文の提案手法は基本的にデータサーバにキャッシュが存在する場合に有効な手法であるため，データサーバにキャッシュが存在せず，ディスクアクセスが発生する場合にも効果的な先読み手法を探ることも，今後の課題とする．

参 考 文 献

1) : Montage : An astronomical image mosaic engine. http://montage.ipac.caltech.edu/.
2) 大辻弘貴, 建部修見: Non-blocking RPC を用いた遠隔ファイルアクセスの実装と性能評価, 研究報告ハイパフォーマンスコンピューティング（HPC）, Vol. 2011-HPC-132, No. 16, pp. 1–5 (2011).
3) Osamu Tatebe, K. H. and Soda, N.: Gfarm Grid File System, *New Generation Computing*, Vol. 28 (2010).
4) Shvachko, K., Kuang, H., Radia, S. and Chansler, R.: The Hadoop Distributed File System, *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pp. 1 –10 (2010).
5) Schmuck, F. and Haskin, R.: GPFS: A shared-disk file system for large computing clusters, *Proceeding of the Conference on File and Storage Technologies*, pp. 231–244 (2002).

6) Prost, J.-P., Treumann, R., Hedges, R., Jia, B. and Koniges, A.: MPI-IO/GPFS, an Optimized Implementation of MPI-IO on Top of GPFS, *SC Conference*, Vol. 0, p. 58 (2001).
7) Carns, P. H., Ligon, III, W. B., Ross, R. B. and Thakur, R.: PVFS: a parallel file system for linux clusters, *Proceedings of the 4th annual Linux Showcase & Conference - Volume 4*, Berkeley, CA, USA, USENIX Association, pp. 28–28 (2000).
8) Chen, Y., Byna, S., Sun, X.-H., Thakur, R. and Gropp, W.: Hiding I/O latency with pre-execution prefetching for parallel applications, *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, Piscataway, NJ, USA, IEEE Press, pp. 40:1–40:10 (2008).
9) Allcock, W., Bester, J., Bresnahan, J., Chervenak, A., Liming, L., Tuecke, S., Memo, S. O. T., Liming, L. and Tuecke, S.: GridFTP: Protocol extensions to FTP for the Grid, *GWD-R (Recommendation*, p. 3 (2001).
10) 伊藤建志, 大崎博之, 今瀬眞: GridFTP-APT : データ転送プロトコル GridFTP の並列 TCP コネクション数調整機構 (インターネットの測定・性能評価技術及び一般), 電子情報通信学会技術研究報告. IN, 情報ネットワーク, Vol. 105, No. 472, pp. 19–24 (2005-12-08).
11) : FUSE :. http://fuse.sourceforge.net/.
12) : InTrigger Platform :. http://www.intrigger.jp/.