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

2004－ARC－157　(27)
2004－HPC－ 97　(27)
2004／3／3

# PVFS-NFS proxy の実装と性能評価

瀬河浩司† 児玉祐悦†
建部修見† 工藤知宏†

広く用いられているオープンソースのクラスタファイルシステム PVFS では、そのファイルシステム上のファイルにアクセスするためには、クライアント側で kernel のコンパイルや PVFS 対応アプリケーションの作成が必要である。また、PVFS は linux マシンでしか動作しないので他の OS からはアクセスすることができない。そこで我々は、PVFS ファイルシステムにアクセスできるように PVFS サーバとの通信と NFS クライアントとの通信を中継する簡易な PVFS-NFS proxy システムを構築した。

本 proxy システムを用いて NFS クライアントから PVFS ファイルへのアクセスを確認すると共に、ファイルシステム上に展開した Program の source package を Build するのに要する時間の計測、および 2GB ファイルの読み書き性能について、本システムと通常の PVFS や NFS との比較を行った。この結果は、NFS の読み書きブロックサイズが PVFS にとって小さすぎるため、あまり性能が得られないことがわかった。今後は、この問題を解決し性能を改善していく予定である。

# Performance Evaluation of PVFS-NFS proxy

Koji SEGAWA ,† Yuetsu KODAMA ,† Osamu TATEBE †
and Tomohiro KUDOH†

The PVFS is an open-sourced cluster file system. Users must re-compile the kernel and/or make a PVFS-ready application to access a file on the file system. And since PVFS system works only on the linux machine, the user of other OS cannot access it. To access the PVFS file system from any machine, we built the simple PVFS-NFS proxy system.

We made sure that NFS client can access PVFS files by use of this proxy sustem. And we evaluated the performance of this system in comparison with the normal PVFS and NFS by building time of a source package and reading/writing time of 2GB file on each file system.

The main reason of poor performance was that the read/write block size of NFS is too small for PVFS.

We will improve the performance by settling the problem.

## 1. はじめに

1 台の計算機では実現できない高スループットを実現する方法として PC クラスタがある。また、PC クラスタの各ノードのディスクに並列にアクセスするクラスタファイルシステムが、ファイルアクセスのバンド幅をスケーラブルに向上させる方法として、注目されている。しかし、現在実装されているクラスタファイルシステムの大部分は特定の OS に依存したシステムであり、またそのファイルシステムへのアクセスにはクライアント側で新たな変更を必要とするなどの問題点がある。我々はこの問題を解決するために、クラスタファイルシステムに NFS からアクセスする簡易なシステムを開発した。

利用したクラスタファイルシステムは、Clemson 大学で開発が進められておりオープンソースで提供されている PVFS[1] である。また PVFS で提供されるファイルシステムに NFS からアクセスするため、user space nfsd (unfsd)[4] を元に修正を行った。

これにより、クライアント側で新たに何もインストールしなくても、通常の NFS クライアントで PVFS のファイルシステムにアクセスすることができるようになる。また、PVFS は linux 上の実装しか存在しないため、通常は linux マシンからのアクセスしかできないが、このシステムを経由することによって他の OS からも PVFS がアクセスできるようになる。

ソースの Build time と 2GB のファイルの読み書きについて、このシステム (PVFS-NFS proxy) の性能を NFS や PVFS のみを使った場合と比較検討した。

† 産業技術総合研究所 グリッド研究センター
Grid Technology Research Center, AIST

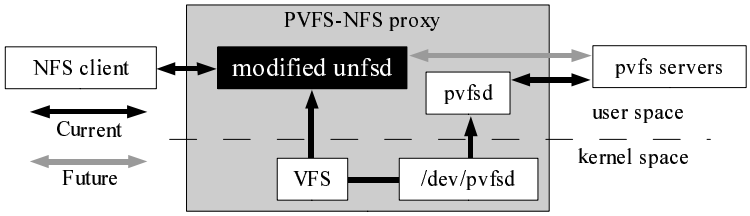


**図 1** PVFS-NFS proxy システム

## 2. PVFS-NFS proxy の構築

### 2.1 PVFS の native API と kernel module

PVFS では、meta data manager (mgr) および複数の I/O daemon (iod) により、ストライピングファイルシステムが構成される。それぞれのファイルは一定のストライプサイズごとに分割され、複数のディスクに格納される。

クライアントライブラリは、PVFS I/O API (PVFS native API) を提供する。クライアントライブラリでは、基本的にファイルオープン時にファイル情報を mgr から得たあと、ファイルへの書き込み読み込み時には mgr を介さず直接必要な iod と通信することにより、効率の良いファイル入出力を可能としている。また、PVFS I/O API により、ストライプサイズ、I/O ノードの数および I/O ノードのオフセット (ファイルのストライピングされた最初のブロックが格納されるノード) を指定することができる。

PVFS は、複数の iod と複数 disk を利用することにより、単一のファイルサーバを用いる NFS のようなボトルネックはなく、高バンド幅のファイル入出力を実現している。native API の使用にはアプリケーション毎にその library を組み込み、動作させる必要があるが、PVFS 本来の高い性能を利用することができる。

PVFS では、kernel module の追加による Linux の VFS(仮想ファイルシステム) の実装もなされている。

この場合の動作は、user space で動いている pvfsd daemon が POSIX/Unix I/O API と PVFS API の変換を受け持つので、kernel space で動いている VFS との間でデータがやりとりされる際、user space と kernel space との間でデータコピーが発生する。このため、native API を用いる場合に比べて性能が低下するが、PVFS タイプのファイルシステムとしてマウントし、POSIX/Unix I/O API により PVFS を利用することができる。

以下、クライアントライブラリを使って PVFS にアクセスする場合を PVFS-native、kernel module を使ってアクセスする場合を PVFS-kernel と呼ぶ。

### 2.2 unfsd と knfsd

Linux には二種類の NFS サーバ実装が存在する。一つは User-Space NFS server (unfsd) で、もう一つは Kernel-Space NFS server (knfsd) である。unfsd は、最初に Linux 上に実装された NFS サーバで NFSv2 ベースのプロトコルを使っており、knfsd は NFSv3 ベースのプロトコルで実装されている。このため、NFSv2 と NFSv3 の違いがそのまま unfsd と knfsd の違いとなる。

NFSv2 では、データをファイルに書き込んでから応答する同期書き込みを行うことになっており、ファイルへの格納を保証しているが、書き込みスループットがサーバ上のディスク書き込み待ちの時間に影響を受ける欠点をもっている。NFSv3 では、データを実際にファイルに書き込む前に応答する非同期書き込みを行うことになっており、サーバの書き込み待ち時間が縮小されが、実際のファイルへの書き込みにはクライアントからサーバへコミットリクエストを送る必要がある。但し、Linux 上の unfsd では、実際にはこの同期書き込みを行わず、ディスクへの書き込みが終る前に終了通知を返す仕様になっている。よって、見掛け上のスループットは高くなるもののサーバクラッシュの際にはデータが失われる危険がある。

ブロックサイズについても NFSv2 では最大 8KB となっており、ネットワーク上で一度に転送できるデータ量がこの値で制限されるが、NFSv3 ではこの制限が除かれている点も両者のスループットに影響を与えている。

さらに、NFSv2 では、ファイルの名前と属性情報を別々に扱っているためキャッシュデータの有効性を調べるためのファイル属性リクエストやディレクトリリストを作成するためのルックアップリクエストのオーバーヘッドが NFSv3 に比べて大きくなってしまう。

また、名前の通りそれぞれの NFS サーバは、user space と kernel space で動いているので、unfsd は uid、gid のマッピングが楽にできるという利点やソースの変更でバグが発生しても最低限の障害で済むという利点がある。

両者にはマウント時の動作にも差が存在する。knfsd は既に import されているファイルシステムを再び export できないという制限があるが、unfsd はオプションによってこれを回避することができる。

よって、スループットにおいては unfsd が knfsd に劣っているものの、我々のシステムに即応的に利用す

る場合 unfsd のほうが適しているといえる。

### 2.3 proxy システムの構成

PVFS サーバに対して NFS クライアントからアクセスするには、次のような方法がある。

- PVFS-kernel を使って PVFS のファイルシステムをマウントし、そのマウントポイントを unfsd で export する方法
- 上項で unfsd の代わりに knfsd を使う方法
- PVFS-native を使って unfsd に PVFS サーバとのやりとりを行わせるようにする方法。
- 上項で unfsd の代わりに knfsd を使う方法

本稿ではもっとも簡便な一番目の方法をとった。これは、まず本方式の妥当性を検証することを目的としたからである。今後高い性能が実現できる方法へ移行することを想定している。具体的には kernel 2.2 用の unfsd[4] と PVFS の kernel module を併用し、PVFS のファイルシステムを NFS のプロトコルでアクセス可能とするものであり、図 1 のような構成である。unfsd を用いる方法を選択したのは、knfsd と違って修正が簡単であり、また何が起こっているかをモニタすることが容易であるからである。

### 2.4 unfsd を PVFS-kernel に用いる場合の問題点

unfsd も NFS daemon であるので、このシステムへの適用にあたって問題となる２つの制限をもっている。一つは二重マウントをしようとしているかどうかのチェックであり、もう一つは既に export されたボリュームを再び export するところでなされるチェックである。これらは、通常の NFS の運用にあたってはファイルの整合性を保つために必要である。本システムの場合は二重マウントの１つ目は proxy 上で行い、かつここでは不特定ユーザによるファイルアクセスが存在しないので不要となる。二重マウントのチェックは、mountd daemon の中の２箇所で行われている。そのうち 1 箇所は pathconf のチェックが目的なのでこちらは skip を行わない。また、後者の export チェックについては、unfsd ではオプション設定によって回避できるようになっている。

### 2.5 PVFS-NFS proxy

本システムの具体的な構築は次のように行う。まず、proxy 用のノード上に PVFS kernel support パッケージをインストールする。この compile には kernel header と PVFS native API library が必要である。出来上がった、kernel module を insmod し pvfsd daemon を走らせる。PVFS ファイルシステムマウント用の mount.pvfs コマンドで PVFS をこのノード上のマウントポイントにマウントする。このマウントポイントを export した後に二重マウントチェックルーチンを解除した mountd daemon と”–re-export” オプション付きで nfs daemon を起動する。unfsd は、起動時に export 情報を取得するのでこの情報を変更した際は、daemon の再起動が必要となる。なお、本システムは PVFS サーバからみて PVFS クライアントでもあるので、同時に複数のノードで動作させることが可能である。実際に、4 ノード上で本 proxy を動作させ、4 ノードの NFS クライアントからそれぞれの proxy にアクセスさせる実験を行い (実際の確認環境に合わせる必要あり)、動作の確認を行った。

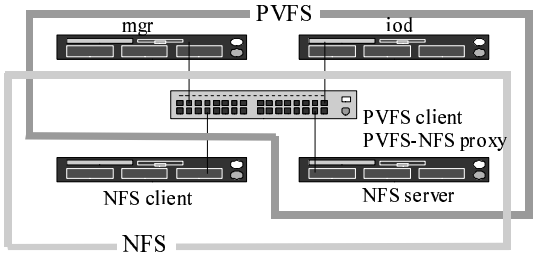


**図 2** 評価環境

## 3. 性能評価

まず、以前の PVFS の評価 [2] で PVFS の性能が低かった make の性能について評価を行った。ローカルディスク、NFS(unfsd)、PVFS-kernel、PVFS-native、PVFS-NFS proxy のそれぞれのファイルシステム上に PVFS(PVFS-1.5.5) のソースファイルを展開し、make するのにかかる時間を測定した。

次に大きなファイルの読み書きバンド幅を測定することによってそれぞれのファイルシステムの性能を比較した。

### 3.1 性能評価環境

評価環境は、次の通りである (図 2)。

- ノード計算機
  - Fujitsu PRIMERGY L200 × 17 台
    chipset: ServerWorks HE SL
    CPU: Pentium III 1.13GHz (2CPU SMP だが 1CPU のみ実装)
    メモリ: 1GB
    HDD: Fujitsu Model: MAN3184MC x 2, SEAGATE Model: ST373307LC x 1 SCSI 接続
    OS: Linux 2.4.19
    * システムと転送データ領域は別ディスク
- ネットワーク
  - NIC: 3Com 3C996B Gigabit Server NIC
  - Switch: 3Com SuperStack 3 (Switch 4924 3C17701)
- Gigabit Ethernet ドライバ
  - Broadcom Gigabit Ethernet Driver bcm5700 with Broadcom NIC Extension (NICE) ver. 2.0.18 (08/24/01)

PVFS 系の構成ではメタデータサーバが 1 台、I/O ノードおよびクライアントノードについては 1-4 台まで変化させることができる。iod daemon (iod) は、それぞれの I/O ノード上で 1 つだけ動作させているので iod の数も 1-4 まで変化することになる。NFS 系の構成には NFS サーバおよび NFS クライアントをそれぞれ 1-4 台まで変化させることができる。なお、PVFS のクライアントノード、NFS サーバ、PVFS-NFS proxy 用のノードは同一であるので proxy ノードも同時に 4 つまで動作させることができる。また、これらは同一の GbE スイッチに接続されている。

### 3.2 Build time の評価方法

NFS に関してはサーバ、クライアントをそれぞれ 1 台づつ、PVFS-kernel に関してはメタデータサーバ、I/O ノード、クライアントノードをそれぞれ 1 台づつ用いて build time の評価を行った。PVFS-NFS proxy に関しては、PVFS メタデータサーバ、PVFS I/O ノード、proxy 用ノード、NFS クライアントをそれぞれ 1 台づつという条件のもと、各ファイルシステムをマウントし、そこに pvfs-1.5.5 のソースツリーを展開し、build にかかる時間を time コマンドで測定した。また、PVFS の仕様で iod が 1 つの時はブロックサイズ=ストライプサイズになるので、ストライプサイズはブロックサイズと同じく 8KB である。なお、PVFS-native は、マウントができないので本測定では除外してある。

### 3.3 バンド幅の評価方法

前項に記述した各ノード数の条件のもと、PVFS-native、PVFS-kernel、PVFS-NFS proxy、local、NFS (unfsd) の各ファイルシステムについてそれぞれ block read / block write 性能の評価を行った。

ここで用いた評価プログラムの動作は以下の通りである。各クライアントとしては次項で述べたベンチマークプログラムを用いており、script による組み合わせで動作させている。多数のクライアントがそれぞれ個別の 2GB ファイルをアクセスする。

(1) 全クライアントノードは一斉に同じ大きさの異なるファイルの書き込みを 3 回行う。このうち 2 回目の書き込みに要した時間を測定し、書き込みバンド幅を測定する。これは、書き込みに要する時間がクライアント毎に異なるため、終了が遅いプロセスがネットワークバンド幅を占有して使うことによる影響を避けるためである。
(2) 全プロセスはバリアにより同期する。
(3) 全クライアントノードは一斉に (1) で書き込んだファイルを 3 回読み出し、2 回目の読み出しに要した時間を測定して読み出しバンド幅を求める。3 回読み出しを行っているのは (1) と同じ理由による。

また、PVFS のストライプサイズは 64KB。但し、前項の評価と同様 PVFS の仕様で iod が 1 つの時はストライプサイズはブロックサイズと等しくなる。

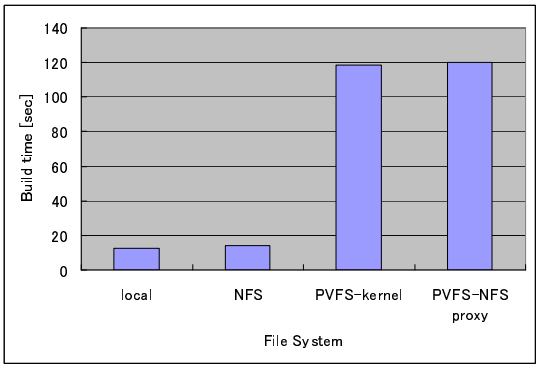


**図 3** Build time の比較

### 3.4 バンド幅評価に用いたプログラム

PVFS-native での測定用に、PVFS クライアントライブラリを利用したディスク I/O ベンチマークプログラムを作成した。本ベンチマークプログラムでは、PVFS native API を用いて、クライアントマシンから PVFS ファイルシステムに対しファイルの読み書きを行う。プログラムのどこが性能ボトルネックとなっているか検証するために、プロセッサタイマを用いた正確な時間測定を用いた。このプログラムには、ブロックサイズ、全体のファイルサイズ、および PVFS のストライプサイズ、I/O ノードの数をパラメータとして渡すことができるようになっている。PVFS によるファイルの読み書きの処理の流れは次の通りである。まず、クライアントがブロックサイズ単位で pvfs_read() / pvfs_write() 関数を呼び出す。すると、PVFS ライブラリ関数がブロックサイズのアクセスをストライプサイズに分割して、設定ファイルに記述された順番に iod に割り当ててアクセスする。これをファイルサイズ／ブロックサイズだけ繰り返す。

また、PVFS-native 以外での測定では、pvfs_read() / pvfs_write() 関数の代わりに通常の read() / write() 等 VFS のシステムコールを使う同様のプログラムを使用した。

## 4. 結果と考察

### 4.1 Build time の評価結果

Build time の測定結果は図 3 の通りである。当初、PVFS-kernel の性能はこのグラフより悪く、local や NFS の場合の 20 倍の時間がかかっていた。この原因を調べるために、1 つのノード上ですべてのファイルシステムを構築し同一プロセッサタイマの値をもとにどこで時間がかかっているか調べてみた。

PVFS での書き込み時には、クライアントから iod に request が送られ、引き続きデータが送られ、iod がそのデータを受け取ってディスクに書き込んだところ

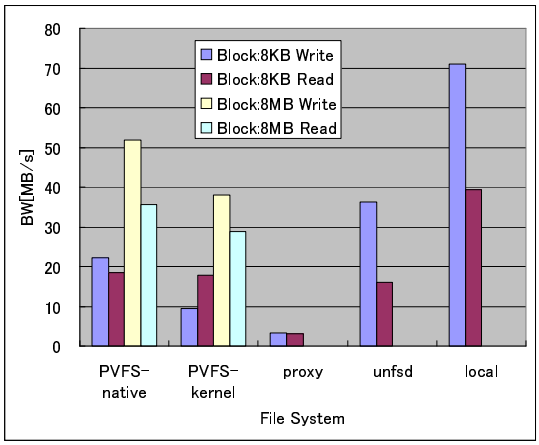


**図 4** 各 File System の性能

で ACK を返し、それをクライアントが受け取ったところで 1 ブロックの書き込みの処理が終わる。ところが、実際の動作を見てみるとクライアントから request が送られたあと、データを送りだすまでに 39msec かかっていることがわかった。また、この現象は、ブロックサイズに依存しており、16KB 以上では起こらないこともわかった。よってデータのバッファリングに関係があるものと予想し TCP_NODELAY ソケットオプションを指定するとこの現象が解消された。

このオプションを指定した状態で再び PVFS-kernel、PVFS-NFS proxy について測定してみると、オプションを指定しなかった場合の Build time にくらべ、どちらのファイルシステムにおいても 2.25 倍時間が短縮された。しかし、それでも NFS の 8 倍も時間がかかっていることになり、両ファイルシステムの共通の要素である PVFS-kernel にボトルネックの要素がまだ残っていることがわかる。

考えられるものに user space と kernel space 間のデータコピーによるオーバーヘッドあるが、複数ノードに渡るプログラム群を同一プロセッサタイマのもとで測定することができないのでまだ未確認である。

なお、以降の測定は、すべて TCP_NODELAY オプションを指定して構築したもので行っている。

### 4.2 バンド幅の評価結果

各ファイルシステムの読み書きバンド幅は、図 4〜図 6 に示した。ファイルサイズは 2GB であり、ブロックサイズは、NFS(unfsd) で用いられる 8KB と以前の測定 [2] で PVFS の性能のよかった 8MB を使用した。

図 4 が示しているように、2GB の大きさのファイルの読み書きの場合でも、8KB というブロックサイズでは PVFS-native、PVFS-kernel のいずれも NFS に比べて性能が悪く、それが PVFS-NFS proxy の性能に影響している。NFS の場合に比べて書き込み時で 1/12、読み込み時で 1/5 という結果がでた。一方、ブロックサイズが 8MB の場合は、PVFS-native、PVFS-kernel

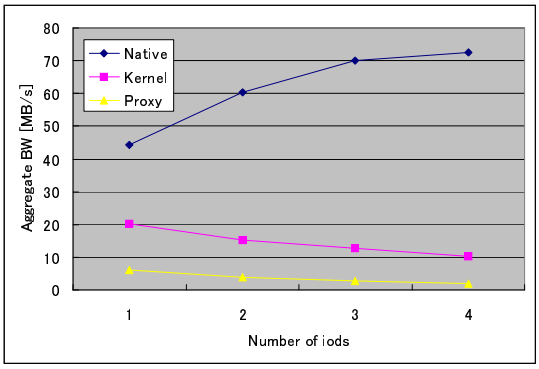


**図 5** iod 数の変化に対する書き込み性能の変化

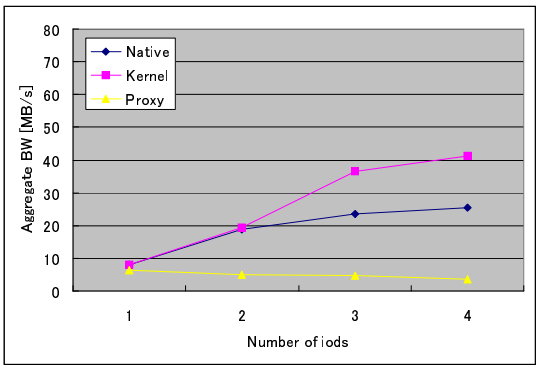


**図 6** iod 数の変化に対する読み込み性能の変化

ともに読み書きの性能は、NFS を上回っている。これは、もし、8MB のブロックサイズで PVFS-NFS proxy が構築できれば、性能が向上することを意味している。

図 5、図 6 は、Block サイズが 8KB の時、PVFS-native、PVFS-kernel、PVFS-NFS proxy の各ファイルシステムについて、クライアント数および proxy 数が 4 の時、iod の数を 1 から 4 まで増やして読み書き性能がどう変化するかみたものである。

図 5 を見ると書き込み性能については、PVFS-native の場合は iod 数が増加すれば性能も増加しているが、PVFS-kernel と PVFS-NFS proxy では逆に性能が落ちていることがわかる。

図 6 を見ると読み込み性能については、PVFS-NFS proxy の場合は書き込みの場合と同様の傾向を示している。PVFS-kernel、PVFS-native の場合は iod 数の増加とともに性能が増加していることがわかる。

## 5. 関 連 研 究

Jin Xiong らは、Dawning Cluster File System (DCFS) というファイルシステムを開発している [5]。PVFS との一番の違いは、メタデータサーバが複数存

在し、メタデータ操作の性能を上げていることである。彼らは同一のクラスタ上で PVFS と DCFS の性能比較を行っており、2048MB のファイルを複数のクライアントで読み書きさせた場合、書き込みで PVFS の 2.0 倍、読み込みで 0.8 倍のピーク性能を得ている。また、49152MB のファイルの場合では、書き込みで 1.5 倍、読み込みで 1.3 倍のピーク性能である。また、このファイルシステムは PVFS の kernel module と同様の module ベースの VFS サポートシステムを持っており、通常のシステムコールでアクセスができる。よって、我々のシステムと同様な仕組みで PVFS 同様 NFS からのアクセスが可能になる。

## 6. おわりに

我々は、簡易 PVFS-NFS proxy システムを開発し、評価を行った。このシステムの利点は、クライアント側のいかなる改変もなしに、NFS クライアントを持つ全てのシステムのユーザにクラスタファイルシステム PVFS へのアクセス方法を提供できる点であり、システム自身の構築も誰もが比較的容易に行え、システムの変更やデバッグも行いやすいという点である。また、PVFS はクラスタファイルシステムであり、多数のクライアントからのアクセスを考慮して設計されているので、PVFS サーバからみて PVFS クライアントである PVFS-NFS proxy も同時に多数走らせることが可能であることも利点であり、実際に正しく動作することも確認した。

現行システムでは、NFS で使用されるブロックサイズが PVFS にとって小さすぎ、通常の NFS と比較してソースツリーの Build time で 1/8、2GB ファイルの書き込み時で 1/12、読み込み時で 1/5 と性能が落ちてしまっている。この原因の一つとして TCP_NODELAY オプションの影響が見つかり、それを排除したが他にも原因が存在しておりまだ解決に至っていない。

将来的には、図 1 の Future の矢印のように unfsd から直接 PVFS サーバにアクセスするような proxy daemon を開発、unfsd に起因する問題点を解決しつつ、高速化を図る予定である。

## 謝　辞

本研究の機会を与えてくださった産業技術総合研究所グリッド研究センター長関口智嗣氏に感謝する。

## 参考文献

1) P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur. PVFS: A Parallel File System For Linux Clusters. In *Proceedings of the 4th Annual Linux Showcase and Conference*, pages 317–327, Atlanta, GA, October 2000.
2) Koji Segawa, Osamu Tatebe, Yuetsu Kodama, Tomohiro Kudoh, Toshiyuki Shimizu. Design and implementation of PVFS-PM: a cluster file system on SCore. In *Proceedings of CCgrid 2003*, pages 705–711, 2003.
3) PC Cluster Consortium. *http://www.pccluster.org/*.
4) The LINUX User-Space NFS Server. *http://packages.qa.debian.org/n/nfs-user-server.html*.
5) Jin. Xiong, Sining Wu, Dan Meng, Ninghui Sun, Guojie Li. Design and Performance of the Dawning Cluster File System. In *Proceedings of Cluster 2003*, pages 232–239, 2003.