システムソフトウェアと
オペレーティング・システム 73－6
（1996．8．26）

並列ファイルシステム MFS

青木 久幸　大和 純一　大谷 寛之　相場 雄一
aoki@csl.cl.nec.co.jp
日本電気(株) C&C 研究所
〒216 川崎市宮前区宮崎 4 丁目 1－1

データベースの大規模化やマルチメディアサーバの発展に伴い、大容量かつ高スループットなファイルシステムの必要性が高まっている。一方、ネットワーク技術の発達に伴い、ワークステーションクラスタや MPP のように、高速な通信網で複数の計算機ノードを接続したシステムが普及してきている。そこで、我々は、多数の二次記憶装置を接続したワークステーションクラスタシステム上で動作する並列ファイルシステム MFS を開発した。このシステムは、ノード間分散格納や並列アクセス機能により、大容量ファイルや高スループットファイルアクセスを実現する。本稿では、MFS の構成や評価結果について報告する。

Parallel Filesystem: MFS

Hisayuki Aoki, Jun-ichi Yamato, Hiroyuki Ohtani, Yuichi Aiba
C&C Research Laboratory, NEC corporation
4-1-1, Miyamae-ku, Kawasaki, Kanagawa, 216 Japan

As the size of databases expands and multimedia servers evolve, large capacity files and high throughput filesystems become more necessary. On the other hand, progress in computer networks makes both workstation clusters and MPPs popular. So we have developed a parallel filesystem called MFS, which runs on a workstation cluster with a number of disks. This system supports large capacity files and high throughput file access by striping data across nodes and permitting parallel accesses. In this article, we explain the structure of MFS and provide measurements of its performance.

1 はじめに

ビジネス計算分野で使われるデータベースは、年々情報蓄積量が大きくなる傾向がある。さらに、近年は、データベースに蓄積された膨大なデータを様々な意志決定に利用する気運も高まっている。このような使い方では、検索時間を短縮するために、記憶容量の増大に見合ったアクセス性能の向上が必要である。

また、VOD{1}などのアプリケーションでは、情報量の大きな動画を多数蓄積するために、サーバには数百 GB～数 TB の二次記憶容量が必要になる。さらに、各ビデオストリームが比較的大きな転送レートを必要とし、複数ビデオを同時供給するサーバには非常に大きな二次記憶アクセススループットが要求される。

最近のディスク装置の性能向上は著しいが、単体では容量・アクセス性能共に限界がある。従って、上記のような要求を満たすためには、多数のディスク装置を併用するしか方法はない。特に、アクセス性能は、二次記憶装置だけの問題ではなく、バスや主記憶アクセス性能など、様々な部分が関係する。従って、システム全体を見渡した対策が必要である。

従来、複数の二次記憶装置を統括管理できるシステムとしては、NFS[2]に代表される分散ファイルシステムがある。しかし、これらのシステムは、名前空間の接続とリモートファイルアクセスに主眼があり、ファイル容量拡大やアクセススループット向上には限界がある。一方、複数の小型ディスク装置を束ねて大きな二次記憶装置とした RAID 装置がある。RAID[3]は二次記憶装置の信頼性向上を第一目的として提案されたものであるが、容量とアクセス性能を増大する効果もある。しかし、現状ではひとつの筐体に入る範囲での規模拡大が主流であり、さらに、装置内部に閉じた並列化を行うために、ソフトウェアの動作を含めた効果的な並列化は困難である。

このような状況の中で、MPP や WS クラスタを想定した大規模なファイルシステムの研究が行われている[4]-[7]。我々も、多数のディスク装置を接続した WS クラスタ上で動作し、二次記憶の容量拡大とアクセススループット向上を実現する並列ファイルシステム MFS を開発した。以下、このシステムの機能や実装方式、性能測定結果などを報告する。

2 並列ファイルシステムの構成

本システムは、図 1に示すように、二次記憶装置が接続された複数の計算機ノードが高速な通信網で接続されたプラットフォーム上で動作する。各々の二次記憶装置は一台の計算機ノードにのみ接続されるディスク非共有型を前提とする。

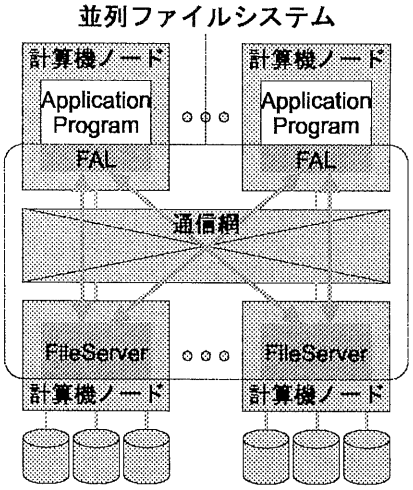

図 1: 並列ファイルシステムの構成

二次記憶装置が接続された計算機ノード上で FileServer が常駐動作し、二次記憶装置を管理する。一方、任意の計算機ノード上で File Access Library(以降 FAL と略す)をリンクしたアプリケーションプログラムを動かすことができる。FAL は、アプリケーションプログラムからのファイルアクセス呼び出しを FileServer への通信に変換する。この他にも、システムの運用や定型的なファイル操作を行うコマンドがいくつかある。

MFS では、FileServer やアプリケーションプログラムを同時に複数の計算機ノード上で動作させることができる。また、ひとつの計算機ノード上で、これらを混在して動かすこともできる。

3 並列ファイルシステムの機能

3.1 ノード間分散格納

本システムでは、図 2に示されるように、複数計算機ノードの複数二次記憶装置にデータ格納領域が分散された論理的なファイルを作成できる。従って、通常のファイルシステムよりも、容量の大きなファイルが利用可能である。

3.2 ダイレクトアクセス

本システムでは、ファイルオープン時に二次記憶装置の担当状況に関係する情報を FAL にキャッシュし、以降のファイルアクセス時に直接担当 FileServer に通信することで、ファイル読み書きアクセスパスを最短化している。これにより、ディスク非共有構成を前提としたファイルシステムとしては、ファイルデータ

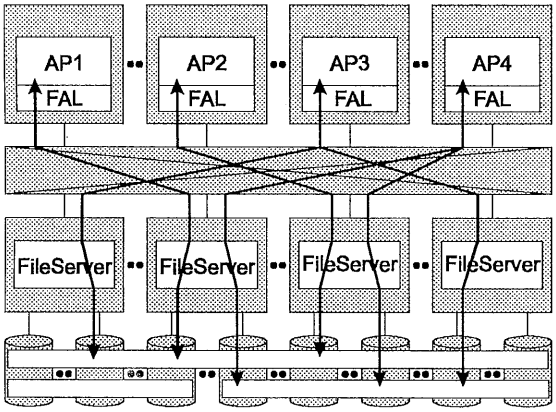

図 2: ファイルの分散格納

中継のオーバーヘッドが少なくなっている。

3.3 多重アクセス

本システムでは、必要最低限の FileServer 間同期で、同時進行する複数のファイルアクセスを処理している。さらに、各 FileServer は、二次記憶装置 I/O の完了待ち時間などを利用して、複数のファイルアクセス要求を効率的に並行処理する。これらの工夫により、使用するプラットフォームの規模に応じたスケーラブルなスループット拡大が可能になる。

特に、前述のノード間分散格納やダイレクトアクセス機能のために、同一ファイルへのアクセスが同時に複数発生しても、アクセス位置が違えば、複数の FileServer に負荷が分散する。従って、同一ファイルにアクセスが集中する場合でも、スケーラブルなスループット拡大が可能である。

3.4 並列アクセス

本システムでは、ひとつのファイルを複数 FileServer が分割管理することを利用して、時間の掛かる処理を並列実行し、処理時間を短縮する。

例えば、図 3に示されるように、一回で大きなサイズの読み書きを行う場合は、FAL がアクセス範囲を二次記憶装置毎に分割し、それぞれの要求をほぼ同時に担当 FileServer に送る。従って、これらの要求が並列に処理され、逐次的に実行するよりも早く終る。

この他にも、ファイルオープン・クローズやファイル縮小において、二次記憶装置毎に行う処理が並列化され、ファイルの分散されている二次記憶装置数が増加しても、処理時間があまり増加しない。

4 実現方式

本システムでは、ランダムアクセス可能で、かつ、同時にはひとつのアクセスのみが可能な二次記憶装置をスピンドルと呼ぶ。

4.1 マルチスピンドルファイルシステム

本システムでは、ファイルデータのノード間分散格納を実現するために、複数のスピンドルから構成されるマルチスピンドルファイルシステムを定義する。このファイルシステムは、最大でスピンドル数の同時アクセスが可能である。マルチスピンドルファイルシステム中の各スピンドルは、次のような領域に分割して管理される。

SuperBlock…ファイルシステムを管理するためのデータを記録する領域であり、スピンドル内部の各領域の位置関係の情報、ファイルシステム内スピンドル番号、空きブロック管理情報などが記録される。

Fnode-List…ひとつのファイルを代表する管理データ fnode の配列である。各スピンドルには、同数の fnode が記録される。

Pnode-List…ひとつの部分ファイルを代表する管理データ pnode の配列である。各スピンドルには、ファイルシステム内の総 fnode 数分の pnode を用意する。

DataBlocks…ファイルデータや可変サイズのファイル管理データを記録するブロックの集合である。この領域中のブロックは、ブロック単位で動的に割り当て解放を行い、様々な用途に使用する。なお、空きブロックはスピンドル毎に管理されるため、違うスピンドル上のブロック割り当てを同時に処理できる。

4.2 ファイルと部分ファイル

ファイルは、マルチスピンドルファイルシステム中の複数ブロックで構成される論理的な二次記憶領域の集合である。ひとつのファイルは、ブロック単位で複数スピンドルに分散可能である。図 4に示されるよう

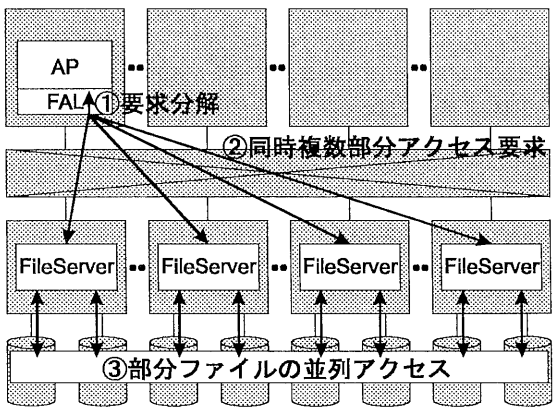

図 3: 分散格納ファイルの並列読み書き

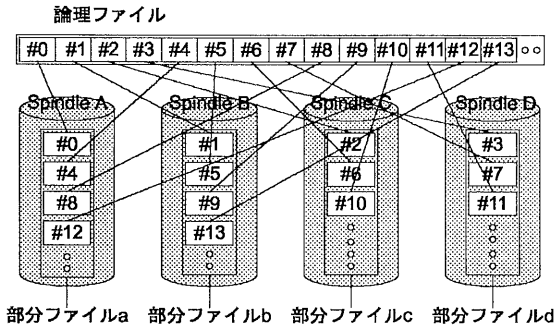

図 4: 部分ファイル

に、ひとつのファイルの同一スピンドルに属するブロックの集合を、そのファイルの部分ファイルと呼ぶ。

従って、ひとつのファイルは複数の部分ファイルから構成され、ひとつの fnode と複数の pnode で管理される。fnode にはファイルのバイトサイズなどが記録され、pnode には部分ファイルのデータブロックの並びを示す BlockMap が付属する。BlockMap は、pnode が属するスピンドルから割り当てられたブロックに格納され、そのブロック番号が pnode に格納される。

4.3 モジュール構成

本システムでは、FileServer がファイル管理の大半の機能を実現する。図 6に示されるように、FileServer は複数のモジュールに分割されている。

FnodeManager は、ファイルシステム中のファイルを、重複と抜けがないように分担管理する。本モジュールは、オープンされているファイルの fnode をロードし、ファイル参照数の管理、ファイル全体の拡張と縮小などを制御する。

PnodeManager は、ファイルシステム中の部分ファイルを、重複と抜けのないように分担管理する。オープン中のファイルの部分ファイルの pnode をロードし、BlockMap 管理、部分ファイル内ブロック番号からスピンドル内ブロック番号への変換、部分ファイルの拡大と縮小などを行う。

SpindleManager は、システム内の全スピンドルを、重複と抜けがないように分担管理する。担当するスピンドル内の空きブロック管理、ブロックキャッシュの LRU 管理などを行う。

4.4 ファイルのオープンとクローズ

ファイルオープン時に、該当ファイルの fnode を担当 FnodeManager に、各部分ファイルの pnode を担当 PnodeManager 中にロードし、部分ファイルの分担情報を FAL にロードする。一方、クローズ時に、これらの更新されたファイル管理データを書き出す。こ

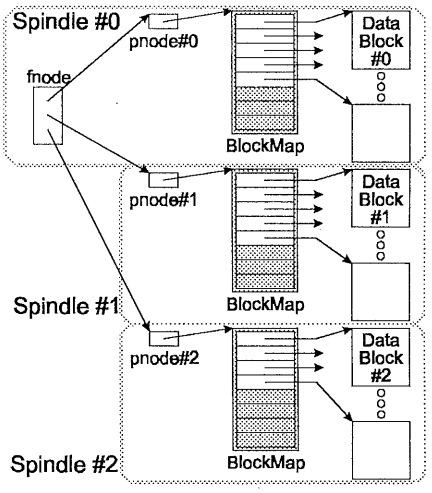

図 5: ファイルの管理構造

の制御により、オープン中はこれらの管理データを I/O せずに参照可能になる。なお、同じファイルが同時に複数のアプリケーションプログラムからオープンされても正しく制御できるように、オープンされた回数を示す参照数を fnode 毎に管理する。

また、各部分ファイルのオープンやクローズを複数の PnodeManager で同時に処理し、分散するスピンドル数が増えても、これらの処理時間があまり増加しないようにする。

4.5 ファイルの読み書き

ファイル読み書きでは、FAL がアクセス範囲をブロック単位に分割し、これら複数のブロックアクセスを同時進行で処理する。これにより、複数ブロックに跨った読み書きを高速化する。

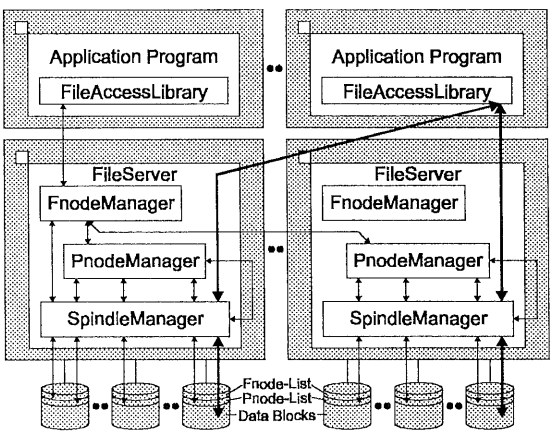

図 6: モジュール構成

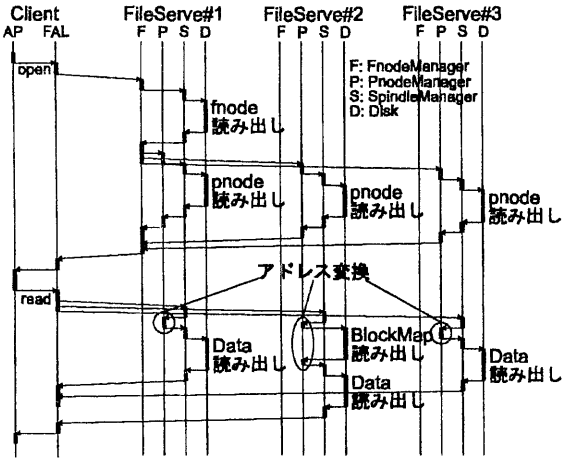

図 7: 通信プロトコルの例

FALは、オープン時にロードした部分ファイル分担情報から、直接担当FileServerにアクセス要求を送る。アクセスブロックのファイル内ブロック番号は、FALが部分ファイルと部分ファイル内ブロック番号に変換し、PnodeManagerがBlockMapを使って部分ファイル内ブロック番号をスピンドル内ブロック番号に変換する。なお、BlockMapはPnodeManager中にLRUでキャッシュし、効率的なアドレス変換を実現する。

書き込み範囲がファイルサイズを超えているときは、ファイルの自動拡張が行われる。最初に、担当FnodeManagerがファイルサイズを拡大し、続いてPnodeManagerが新データブロックを割り当て、これをBlockMapに登録して、部分ファイルを拡張する。なお、読み出し時にアクセス範囲がファイルサイズを越えている場合はエラーリターンである。

ファイルオープンからファイル読み出しの通信シーケンス概略を図 7に示す。

5 性能評価

5.1 測定方法

本システムを評価するために、図 8に示すようなシステムを構築した。7 個の EWS4800/320 を ATM-SW(ATOMIS5)で接続し、このうち4個のEWSに1GBのディスク装置を 4 個づつ接続した。残りのEWSはアプリケーションプログラムを動作させるノードとした。OSにはUNIX SVR4.2MP系のUX/4800を使用し、ノード間通信にはUDPを用いた。

この構成の上で本システムを動作させ、8KB単位に16 個のディスク装置に分散格納された 32MB のファイルを作成し、性能測定プログラムからファイル読み書きを発行する。この上で、FAL呼び出しを発行して から応答が戻るまでの処理時間を測定した。

5.2 測定結果

一回のFAL呼び出しのIOサイズを変化させたときの入出力転送レートを図 9 に示す。

Append Write はファイルを拡張しながらデータを書き込む処理であり、Update Write は既にアロケートされている領域への上書きである。また、Write Through は二次記憶装置までの書き込みを確認してから完了する書き込み処理であり、Write Back は FileServer 中のキャッシュに書き込んだ時点で応答を返す書き込み処理である。実際の書き込み処理には、この二つの特性の組み合わせで合計 4 種類のパターンがある。

一方、Cache Mishit Read は FileServer 中に該当データがキャッシュされていない状態での読み出し処理であり、Cache Hit Read は該当データが FileServer 中にキャッシュされているときの読み出し処理である。この場合は、スピンドルアクセスは発生せず、ノード間通信のみで処理できる。

5.3 並列読み書きの効果

図 9 を見ると、FAL 呼び出しの IO サイズを大きくしていくと、入出力レートは向上し、やがて飽和する。

本システムでは、FAL 呼び出しの IO サイズがブロックよりも大きくなると、並列に読み書きが行われる。図 9 の測定結果は、様々なタイプの読み書き処理において、この並列アクセスの効果が出ていることを示している。アクセスの種類で高速化率は変化するが、およそ 4 並列で 2.8～3.4 倍程度高速化されている。

並列アクセスの性能向上が飽和するのは、FAL 中の通信処理時間の増加が原因と考えられる。並列度が増加すると、FAL の通信回数が増え、このための CPU 時間が増加する。この通信処理だけで CPU 稼働率が

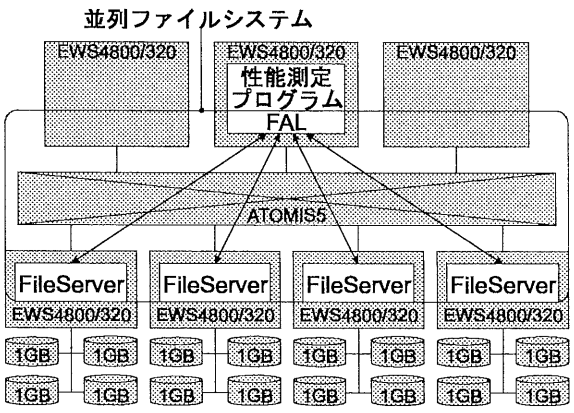

図 8: 性能測定のシステム構成

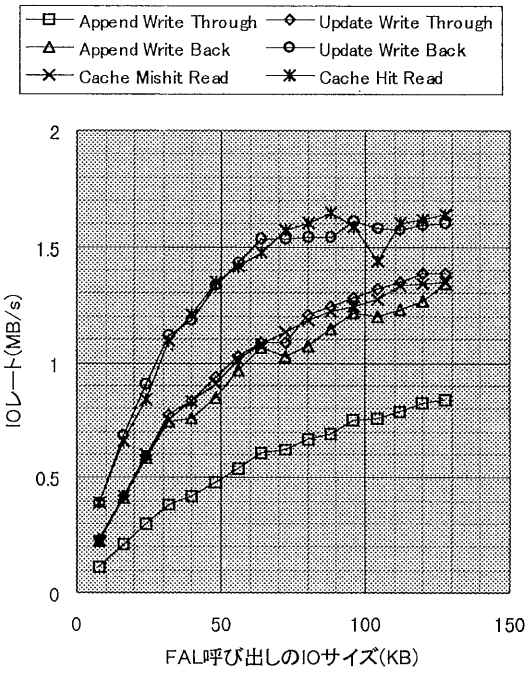

図 9: FAL 呼び出しの I/O レート

100%に達すると、FAL の処理ネックとなり、入出力レートは一定の値に収束する。実際、飽和している領域では、クライアントプロセスの CPU 負荷率は 100%に近い値を示す。

逆に考えると、ノード間通信に必要な CPU 時間が減少すれば、FAL は短い時間により多くの FileServer と通信可能になり、より高い IO レートを達成可能になる。すなわち、通信処理が軽くなるほど、並列アクセスの効果は大きくなる。

5.4 読み書きの種類による変化

図 9 を見ると、下記の順に性能が良くなっている。

1. Cache Hit Read, Update Write Back
2. Cache Mishit Read, Update Write Through, Append Write Back
3. Append Write Through

一般に、Update Write に比べて、Append Write はファイルの拡張処理が余分に必要なため、性能が悪化する。本システムでも、ファイル拡張に伴い、ファイル管理の一貫性を維持するために、FileServer との通信が 1 回余分に必要になっている。このため、かなり大きな性能差が出ている。

また、Write Through は毎回二次記憶装置まで書き込む必要があるので、Write Back に比べると性能が悪化する。Cache Mishit Read と Cache Hit Read との間にも同じ関係が存在する。本システムでも、この傾向が明確に表れている。

さらに、Cache Mishit Read と Update Write Through の性能はほとんど同じである。本システムの制御上は、両者とも、毎回ディスク装置まで一回 I/O が出、ノード間の通信回数も同じである。このように比較的重い処理が同じ回数実行されるため、両者の性能が近づくものと考えられる。Cache Hit Read と Update Write Back についても同じことが言える。

6 おわりに

本稿では、ノード間分散格納が可能な並列ファイルシステムの機能と実装方式を紹介した。このシステムは、多数の二次記憶装置が接続された WS クラスタ上で動作し、ファイル読み書きアクセスにおいて並列化の効果が観測された。しかし、まだ一般の通信プロトコルを使っているため、本来の絶対性能は得られていない。今後、この点の改善と、スループットのスケーラビリティ実証などを行っていく予定である。

謝辞

本研究の機会を与えて頂いた C&C 研究所山本前所長、後藤所長、日頃ご指導くださった中崎部長、さらには、MFS の開発に御協力いただいたビップシステムズ株式会社の内山氏、高笠氏に感謝いたします。

参考文献

[1] "日経ニュースメディア別冊 最前線レポートインタラクティブ・テレビ 通信・放送融合へのチャレンジ", pp.22-31

[2] Hal Stern, 倉骨彰訳, 砂原秀樹監訳, "Managing NFS and NIS", アスキー出版, 1992

[3] "日経ウォッチャーIBM 版別冊 出揃った並列汎用機とディク・アレイ", pp.190-204, 1995

[4] Peter F. Corbett, Dror G. Feitelson, Jean-Pierre Porst, Sandra Johnson Baylor, "Parallel Access to File in the Vesta File System", Proceedings of Supercomputing'93, pp472-481, 1993

[5] 住元真司, "クラスタシステム向け共有ファイルシステムの実現と評価", SWoPP'95, pp.9-16, Aug. 1996

[6] Roger L. Haskin, Frank B. Schmuck, "The Tiger Shark File System", Proceedings of COMPCON '96 , pp.226-231, 1996

[7] Wayne M. Cardoza, Frederik S. Glover, and William E. Snaman Jr., "Overview of Digital UNIX Cluster System Architecture", Proceedings of COMPCON'96, pp.254-259, 1996