オペーレーティング・システム 54－4
（１９９２．３．１３）

ワークステーション用 Unix カーネルの高性能化の試み

手塚宏史
ソニー株式会社 ワークステーション事業部

年々高性能化するマイクロプロセッサに対して、メモリ、ディスクなどの性能はそれほど向上していない。我々はマイクロプロセッサの高い性能を活かすために、従来の Unix カーネルに対して、ファイル I/O、プロセス・スケジューリングなどの点でいくつかの改良を試みた。これにより、より高いファイル I/O スループット、CPU 性能の向上、レスポンスタイムの改善などの結果が得られた。

Improved Performance of Workstation Unix Kernel

Hiroshi Tezuka

Workstation Division, Sony Corporation
Gotanda AN Bldg. 1-22-1, Higashigotanda, Shinagawa-ku, Tokyo 141, Japan

Although the performance of microprocessors has recently been radically improved, the performance of memories and disks has not. We report our work on file I/O, memory management and process scheduling of a workstation Unix kernel to allow the efficient use of high-performance microprocessor. As the result of this work, we can get higher file I/O throughput, higher CPU performance and shorter response times.

1. 背景

最近のマイクロプロセッサの性能向上には目を見張るものがある。例えばワークステーションの分野における数年前と現在の仕様を比べると表−1のようになる。ここで () 内は MC68020 のシステムを1としたときの性能比である。

表−1 ワークステーションの性能の推移

		1986/1Q		1988/2Q		1991/1Q		1993 (予想)	
CPU	名称	MC68020		MC68030		R3000		R4000	
	クロック MHz	16.6		25		25		50	
	性能 MIPS	2.3	(1)	5.3	(2.3)	25	(10.8)	80	(34.8)
	キャッシュ KB	8	(1)	64	(8)	128	(16)	1024	(128)
メモリ	集積度 Mbit/chip	1		1		4		16	
	容量 MB	8	(1)	16	(2)	64	(8)	256	(32)
	アクセスタイム ns	100	(1)	80	(1.25)	80	(1.25)	60	(1.7)
	バンド幅 MB/s	16.7	(1)	50	(3)	100	(6)	260	(15.6)
ディスク	容量 MB	150	(1)	280	(1.9)	1280	(8.5)	3000	(20)
	アクセスタイム ms	30	(1)	24	(1.25)	20	(1.5)	18	(1.7)
	バンド幅 MB/s	1	(1)	2	(2)	2.4	(2.4)	3	(3)

ところが良く見ると全ての項目が等しく改善されているわけではない。つまり、CPU の性能向上、メモリ容量の増加、ディスク容量の増加に比べて、

a) メモリのアクセスタイムが速くなっていない

b) ディスクのアクセスタイムが速くなっていない

c) ディスクのバンド幅が大きくなっていない

ことがわかる。

特にハードディスク装置に代表される二次記憶装置のアクセス速度と CPU の速度比は RISC 技術などによる CPU の高性能化に伴って一段と大きくなる傾向にある。このままではシステムとしてのバランスが悪くなり、せっかくの高性能な CPU の能力を活かせない。

また、ワークステーションのようにユーザが直接扱うコンピュータでは実際に操作したときの使用感が重要で、例えば、バックグラウンドでコンパイルなどを行っているときにインタラクティブな応答性が悪くなるようではいけない。

本論文ではこれらの問題を改善するために行った、いくつかの Unix カーネルの改良の試みとその効果について述べる。なお、本論文中での Unix とはバークレー版 4.3BSD を元にしている。

2. バッファキャッシュの動的割り当て

メモリ素子の集積度の向上と価格の低下により、ワークステーションに実装可能な主記憶容量は年々増加しているので、ファイルアクセスを高速化するために、この大容量の主記憶をキャッシュとして有効利用することが考えられる。

そのひとつの方法は最近のカーネルで用いられているような仮想記憶管理とファイル I/O の融合、つまり、メモリマップによるファイルアクセスである[7]。しかし、そのためにはカーネルの大幅な変更が必要になるため、我々は従来からあるバッファキャッシュを拡張する方法を取った。

オリジナルのバークレー版の実装では主記憶容量の約 10% 程度をバッファキャッシュに固定的に割り当てていたが、これを動的に割り当てるように変更して、主記憶のほぼ全体をバッファキャッシュとして使えるようにした[2][9]。我々はこれを**ダイナミックバッファ**と呼んでいる。

このダイナミックバッファにより次のような点が改善される。

a) 主記憶内にキャッシュできるデータ量が増加するので二度目以降のファイルの読みだしが大幅に高速化される。

b) ディレクトリ、i-node、間接ブロックなどのキャッシュによって、ファイルの書き込み速度が改善される。

c) 一度に使用できるバッファ量が増加するので、非同期書き込み[5]により write システムコールの時間が短縮される。

d) 最適なバッファキャッシュの量をメモリ容量によって個別に決める必要がない。

このようにバッファキャッシュの管理部分を変更することは容易で効果も大きいが、実際には次のようないくつかの副作用が発生し、その対策を行わないと実用にはならない。

a) オリジナルのバッファキャッシュ操作アルゴリズムにはバッファキャッシュの数が比較的少ないことを前提にしているものがあり、そのままでは非常に効率が悪いものがある。

b) Unix では部分的に変更されたデータブロックなどは毎回ディスクには書き込まず（遅延書き込み[5]）通常 30 秒に 1 回程度行われる sync 操作によってまとめてディスクに掃き出すようになっているが、バッファキャッシュの数が多くなると、 sync に非常に長い時間がかかるようになる。

c) バッファキャッシュにあまり多くの物理メモリを専用に割り当てると仮想アドレス空間用の物理メモリが不足して性能が低下してしまうため、負荷に応じて物理メモリの動的な再割り当てを行う必要がある。

3. ディスク I/O のブロックマージ

ダイナミックバッファにより、キャッシュによるファイルアクセスの高速化は図れるが、最初の読み出しや、書き込みスループットなどを改善するためにはディスクのデータ転送速度自体を高速化する必要がある。また、バッファキャッシュに入り切らないような大きなファイルを扱う場合にはディスクのスループットが支配的になる。

ここで、ワークステーションで広く用いられている SCSI インタフェースのハードディスクについて考えてみると一般的に次のような特徴があることがわかる。

a) 論理ブロックでアクセスされるため、ヘッドのシーク時間、ディスクの回転待時間などを考慮したアクセス・スケジューリングが行いにくく、数Kバイト程度のアクセスを多数行う場合のデータ転送速度はあまり速くない。

b) 数百Kバイトと言った大容量の連続したデータ転送がひとつのコマンドで可能で、その場合の転送速度は比較的速い。

このため、SCSI ディスクの場合は、 Unix で一般に行われている 4〜8K バイト／ブロックのアクセスでは高速なデータ転送は行えない。

我々は連続したデータ転送の速い SCSI ディスクの特徴を生かすために、ディスク上の連続した領域をまとめて、1回の I/O 要求でデータ転送を行う機構をデバイスドライバに実装すると共に、ファイルシステムのパラメータを変更して可能な限りディスク上のデータブロックを連続して割り当てるようにした[8]。こうしても従来との互換性は保たれる。

さらに、読み出し時のスループットを高めるために、従来行われていた非同期の1ブロックの先読みを複数ブロックに拡張している。表－2にブロックマージの有無によるスループットの違いを示す。

表－2 ブロックマージの有無とスループットの実測例

	書き込み	読み出し
ブロックマージなし	327.7 KB/s	630.2 KB/s
ブロックマージあり	827.5 KB/s	765.6 KB/s

マージされたブロックがあまり大きくなると1回のデータ転送でディスクを占有する時間が長くなり他の I/O 要求の応答性が悪くなるため、マージされたブロックの最大値を 128K バイトに制限している。

4. ディスク I/O の優先度付きスケジューリング

ディスク I/O でもうひとつ重要なのは I/O 待時間の短縮である。データブロックの書き込みは通常終了を待たない（非同期書き込み）ために実際に I/O が終了するまでの時間はあまり問題とはならないが、ディレクトリや i-node などの管理ブロックなどは、システムクラッシュ時の影響を最小限にするためにデータ転送の終了を待つようになっている。（同期書き込み）これらの同期アクセスが非同期のデータ転送によって遅れると、そのまま処理時間の増加につながる。

従来の Unix では、ディスクのアクセスはその内容にかかわらず全て平等に、通常はシリンダ番号順にスケジューリングされるため、前述のダイナミックバッファなどによりディスクの I/O 待ち行列の長さが長くなると、この同期 I/O の遅れは無視できなくなる。

我々はこの同期 I/O の遅れを小さくするために、同期 I/O 要求に非同期 I/O 要求よりも高い優先度をつけ、非同期 I/O よりも先にサービスを行うようにした結果、表－3のように write システムコールの時間を短縮することができた[8]。

表－3 優先度付きスケジューリングの有無による実測例
（光磁気ディスクへの 1MB の書き込み）

	write の終了	I/O の終了
優先度なし	4.9 sec	5.0 sec
優先度あり	1.5 sec	5.2 sec

このようにI/O要求に優先度を付けると、同期I/Oの発生頻度が高い場合に非同期I/Oがサービスされなくなる可能性があるので、その対策が必要である。

また、ディスクの待ち行列を分割することにより、ヘッドのシーク距離が増加してスループットの低下が起きるが、無視できる程度であると考えられる。

5. NFSの高速化

Sun Microsystems社のNFS (Network File System) は業界標準のリモートファイルシステムとして広く使われているが、一般にファイル書き込みのスループットが低いという問題がある。

これはクライアントに応答が返った時点でサーバ側のデータは全て二次記憶上に書き込まれているというプロトコル上の制約から、通常は次のように同期書き込みを用いてサーバ・プロセス (nfsd) が実装されているためである。

	nfsd 1	nfsd 2	
(1)	要求を受信	要求を受信	
(2)	i-node をロック	・	
(3)	**ブロックの割り当て**	・	(同期書き込み)
(4)	**データの書き込み**	・	(同期書き込み)
(5)	**i-node の更新**	・	(同期書き込み)
(6)	i-node をアンロック	・	
(7)	結果を送信		
(8)	・	i-node をロック	
(9)	・	**ブロックの割り当て**	(同期書き込み)
(10)	・	**データの書き込み**	(同期書き込み)
(11)	・	**i-node の更新**	(同期書き込み)
(12)	・	i-node をアンロック	
(13)	・	結果を送信	

ディスクI/Oの間はファイルは変更に対してロックされているので、サーバプロセスは複数存在するにもかかわらず、1つのファイルに対する処理はシリアルに行われ、必ず3回のディスクアクセスが必要になる。また、NFS クライアントでは非同期 I/O を実現するために複数のAsyncデーモン (biod)を用いているが、書き込みスループットの向上には役立っていない。

しかし、nfsd も biod も複数個存在していることから、複数のデータ転送要求が並列に発行されているはずであり、それを利用することでスループットの向上が可能だと考えられる。

我々が用いたのは、次のように各サーバ・プロセスでは非同期書き込み、あるいは遅延書き込みを用いてI/O要求を処理した後に、複数のサーバプロセスの待ち合わせを行い、ディスクI/Oの終了を待ってそれぞれのサーバ・プロセスがクライアントに応答を返すという方法である[9]。ファイルがロックされている時間は遅延書き込み、非同期書き込みによりごく短くすることができる。

	nfsd 1	nfsd 2	
(1)	要求を受信	要求を受信	
(2)	i-node をロック	•	
(3)	**ブロックの割り当て**	•	(遅延書き込み)
(4)	**データの書き込み**	•	(非同期書き込み)
(5)	**i-node の更新**	•	(遅延書き込み)
(6)	i-node をアンロック	•	
(7)	•	i-node をロック	
(8)	•	**ブロックの割り当て**	(遅延書き込み)
(9)	•	**データの書き込み**	(非同期書き込み)
(10)	•	**i-node の更新**	(遅延書き込み)
(11)	•	i-node をアンロック	
(12)	**待ち合わせ**		
(13)	**ブロックをフラッシュ**		(同期書き込み)
(14)	結果を送信	結果を送信	

(12) の待ち合わせを行うのは非同期に到着する I/O 要求をできるだけ数多くまとめて (13) で1度にディスクに書き込むためである。

この方法により、表−4のように書き込みスループットが向上した。

表−4 待ち合わせ機構の有無と書き込みスループットの実測例

	書き込みスループット
待ち合わせなし	98.5 KB/s
待ち合わせあり	440.4 KB/s

また NFS では読み込み時の先読みブロックの数を増やすことで表−5のように読み出しのスループットを改善することができる。これは biod により複数の読み込み要求が並行してサーバに送られるため、ネットワークのバンド幅の利用率が高くなるためだと考えられる。

表−5 先読みブロック数と読み出しスループットの実測例

先読みブロック数	読み出しスループット
1ブロック	505.7 KB/s
4ブロック	787.7 KB/s

6. キャッシュを考慮した物理メモリ割り当て

最初に述べたように、 CPU の大幅な速度向上に比べて主記憶のランダム・アクセスタイムはそれほど短縮していない。このギャップを埋めているのが大容量のキャッシュであるため、キャッシュのヒット率が CPU の性能に与える影響は大きい。特に MIPS 社の R3000 のように物理ア

ドレスタグ、ダイレクトマップ方式のキャッシュでは、物理ページの割り当て方がキャッシュのヒット率を左右する[6]。

理想的には実際のキャッシュの衝突が最も少なくなるように物理ページを割り当てるのが望ましいが、実行中にキャッシュのヒット率を計測するのは困難なので、仮想アドレスをヒントにして物理ページを選ぶ方法を取った[8]。つまり、（ページ番号 mod キャッシュサイズ）が同じ物理ページを対応する仮想ページに割り当てることによって、キャッシュサイズの仮想アドレス空間の範囲内ではキャッシュの衝突を防ぐことができる。

物理メモリの割り当て方法による CPU 性能の違いを表－６に示す。

表－６ 物理メモリ割り当て方法と CPU 性能の実測例
(Linpack ベンチマークによる、単位は MFLOPS)

実行回数	1回目	2回目	3回目	4回目	5回目	6回目	7回目	8回目
ヒントなし	5.600	5.849	5.689	5.766	5.714	5.500	5.686	5.569
ヒントあり	6.232	5.987	6.209	6.140	6.198	6.126	5.984	5.903

この方法の欠点は、プログラムの挙動を全く考慮していないために、必ずしもキャッシュ衝突が少なくなるとは限らないことである。

7. プロセス・スケジューリングの改善

Unix ではバックグラウンドで大量のコンパイルなどを行っている場合にインタラクティブな操作の応答性が悪くなる場合があるが、これは次のような Unix のプロセス・スケジューリング・ポリシーに原因がある。

a) 全てのプロセスが平等にスケジュールされるため、同時に多くのプロセスを発生するジョブの CPU 占有率が高くなる

b) 実行時間の短いプロセスの方が優先度が高いため、短いプロセスを数多く発生するジョブの CPU 占有率が高くなる

このため、バックグラウンド・ジョブ全体としては長時間 CPU を占有しているにもかかわらず、見かけ上の優先度が高くなり、フォアグラウンドのプロセスの実行速度が遅くなってしまう。

さらに、バックグラウンド・ジョブの見かけ上の優先度が、新しいプロセスの起動のたびに不規則に変動するために、実際の使用感はさらに悪くなる。

この問題を解決するにはジョブごとの CPU 占有率に基づいてジョブ内の各プロセスの優先度を決定すれば良いが、Unix には明確なジョブの概念がなくプロセスグループで代用しており、厳密にジョブの優先度を求めることは面倒なので、我々は、次のようにプロセスグループ内の親プロセス、子プロセス間で CPU 占有率を継承することにより、この現象を軽減することを試みた。

a) fork 時には親プロセスの CPU 占有率を子プロセスが継承する

b) exit 時には同じプロセスグループの子プロセスの CPU 占有率を親プロセスが引き継ぐ

こうすることで、多数の短いプロセスによるジョブ全体の優先度は単一の長いプロセスと同程度になり、見かけ上の優先度の変動も少なくなるので、インタラクティブな処理の応答性を高めることができた。

8. まとめと今後の課題

このようにオペレーティングシステム・カーネルに対して、いくつかの高性能化の試みを行ってきたが、それなりの効果はあったものと確信している。

しかし、このような対症療法的な修正は、全体のバランスを考えるとあまり好ましいものとは言えない。ダイナミックバッファによる副作用などはその良い例で、その解決に、実際には3年ほどかかっている。

なお、現在まだ未解決な問題点には次のようなものがある。

a) クライアント／サーバモデルなどのように、プロセス間に強い相互関係がある場合にはUnix の平等なスケジューリングでは不具合が生じる場合がある。

b) Unix では I/O やネットワークのスケジューリングに関しては特に何も考慮していないために、大量の低速 I/O を行うプロセスが一時的にせよ資源を占有してしまうことがある。

c) タイナミックバッファや I/O スループットの改善によってファイル I/O のデータ転送速度は向上したが、ファイルの生成／消去の効率はあまり改善されていない。

参考文献

[1] AT&T Bell Laboratories, *The UNIX System* , Prentice-Hall, 1987

[2] Rodriguez, R. Koehler, M. and Palmer, R., "A Dynamic UNIX Operation System", *Summer 1988 USENIX Conference Proceedings* , pp.305-319.

[3] Seltzer, M. Chen, P. and Ousterhout, J., "Disk Scheduling Revisited", *1990 USENIX Conference Proceedings* , pp.313-324.

[4] Bach, M., *The Design of the UNIX Operation System* , Prentice-Hall, 1986.

[5] Leffler, S. McKusick, M. Karels, M. and Quarterman, J., *The Design and Implementation of the 4.3 BSD UNIX Operation System* , Addsion Wesley, 1989.

[6] Kane, G. MIPS R2000 RISC Architecture, Prentice-Hall, 1987.

[7] Young, M. Tevabian, A. and Rashid, R., "the Duality of Memory and Communication in the Implementaion of a Multiprocessor Operation System", *The Eleventh ACM Symposium on Operation Systems Principles* , 1987.

[8] 手塚, 林,「Unix の高性能化を図り, 高速 MPU と大容量の主記憶に対応する」, 日経エレクトロニクス, 1990 年 4 月 30 日号, no.498, pp.171-189.

[9] 手塚,「NFS の高速化へのソフトウェア・アプローチ」, 第 17 回 jus UNIX シンポジウム論文集, pp.96-103