分散オペレーティングシステムSolelcにおけるファイル管理機構の構成と性能評価

水口孝夫[†] 芝 公仁[††]
毛利公一[†††] 大久保英嗣[†††]

我々が開発している分散オペレーティングシステムSolelcでは，各計算機の抽象化機構が協調して動作し，位置透過な資源管理を可能とする環境を構築する．この環境上でカーネルを動作させることで，カーネルの機能は分散環境を意識することなく実現可能であり，単一のカーネルでネットワーク上の複数の計算機を管理することができる．また，Solelcでは，カーネルやプロセスの処理をスレッド単位で任意の計算機に配置することが可能である．これらの特徴を利用し，ファイルサービスに関するシステムコール処理を細分化し，各処理をその利用形態に応じて適切な計算機上で動作させる．これにより，処理の分散・並列化が促進され，効率的なファイル管理を実現することができる．本論文では，Solelcにおけるファイル管理機構の構成法とその性能評価について述べる．

The Construction of File Management in Solelc Distributed Operating System and Its Performance Evaluation

Takao Mizuguchi,[†] Masahito Shiba,[††] Koichi Mouri[†††]
and Eiji Okubo[†††]

We have been developing Solelc distributed operating system. In Solelc, cooperative work of the abstraction mechanism on each computer provides an environment for location transparent resource management. On this environment, functions of kernel are realized without considering the distributed background. Therefore, a kernel can manage plural computers. Additionally, threads executing codes of kernel or process work on a voluntary computer. Making use of these characteristics of Solelc, system call processings for file services can be executed on appropriate computers, and effective file management can be realized. In this paper, the construction of file management in Solelc and its performance evaluation are described.

1. はじめに

現在，我々は，単一のカーネルによって複数の計算機を管理可能な分散オペレーティングシステムSolelcの開発を行っている．Solelcでは，システム内の各計算機上で計算機資源を抽象化する機構が動作し，それらの機構がネットワークを介して協調することにより，システム全体の計算機資源の抽象化を実現している．特にメモリに関しては，各計算機の抽象化機構がメモリの一貫性制御を行うことで，複数の計算機でアドレス空間を共有する．これにより，カーネルやプロセスのコードをすべての計算機で共有し，それらを実行するスレッドを任意の計算機上で動作させることが可能となる．抽象化機構によるメモリ一貫性制御は暗黙的に行われるため，カーネルやプロセスは分散環境を意識する必要がなく，位置透過に動作可能である．カーネルは，抽象化機構が構築する環境上で動作することで，複数の計算機の資源を同時に管理可能となり，システム全体を考慮した資源管理が実現される．また，プロセスもこの環境上で動作するため，カーネルの機能をすべての計算機から同様に利用することが可能となり，効率的な計算機資源の利用が実現される．

Solelcでは，カーネルやプロセス全体を複数の計算機で共有するため，複数のカーネルスレッドや同一プロセスに属する複数のユーザスレッドを異なる計算機

† 立命館大学大学院理工学研究科
Graduate School of Science and Engineering, Ritsumeikan University
†† 龍谷大学理工学部
Faculty of Science and Technology, Ryukoku University
††† 立命館大学理工学部
Faculty of Science and Engineering, Ritsumeikan University

に配置することも可能である．スレッド単位で複数の計算機に分散させて CPU 処理能力を最大限に利用したり，特定の計算機の資源を頻繁に利用するスレッドを当該計算機に固定して動作させるなど，さまざまな処理形態に応じたスレッド配置が実現可能である．これによって，全体として効率的な処理を実現することができる．

本論文では，以上の特徴を持つ Solelc 上に実現したファイル管理機構の構成とその性能評価について述べる．Solelc におけるカーネルでは，分散環境を意識しない設計が可能であることから，従来の単一計算機での動作を前提としたファイル管理が適用可能である．しかし，カーネルの機能を動作させる計算機を無作為に選択し，これにスレッドを配置する手法では，メモリ一貫性制御によるページ転送が頻繁に発生してしまうなどの理由から，十分な性能を得ることが困難である．そこで，本ファイル管理機構では，Solelc のスレッド配置の自由度の高さを活かし，システムコール処理を細分化し，各処理をその処理形態に応じて適切な計算機に配置する．本手法により，複数の計算機から共有されたアドレス空間における効率的な分散・並列処理が実現される．

我々は，Solelc の開発において，位置透過な資源管理を実現した[1),2)]．オペレーティングシステム（以下，OS と記す）が管理する計算機資源であるメモリ，CPU，割込み，周辺デバイスを抽象化することで，カーネルが複数の計算機を同時に管理可能となり，システム全体を考慮した資源管理が実現された．OS の資源管理において最も重要なものの 1 つであるファイル管理は，OS の基本的な機能の多くを使用することで実現される．また，プロセスに対して提供されるシステムコールは，ファイル管理の機能を使用して実現されるものが多い．したがって，ファイル管理の効率化手法およびその性能を検証することにより，OS が実現する多くの機能の性能向上が期待できる．

以下，本論文では，2 章で Solelc の概要，3 章でファイル管理機構の概要について述べる．次に，4 章および 5 章では，ファイル管理機構の構成要素であるファイルマネージャとワーカについてそれぞれ述べる．さらに，6 章で本システムの性能評価，7 章で関連研究について述べ，提案手法の有効性について議論する．

2. Solelc の概要

Solelc では，各計算機の抽象化機構がネットワークを介して協調することにより，計算機資源の位置透過な利用を可能とする環境を構築する．この環境上でカーネルを動作させることによって，単一のカーネルでネットワーク上の複数の計算機を管理することが可能である．また，プロセスは，任意の計算機から位置透過にカーネルの機能を利用することが可能である．さらに，カーネルやプロセスのコードを実行するスレッドをその処理形態に応じて適切な計算機に配置することで，効率的に処理を行うことも可能である．

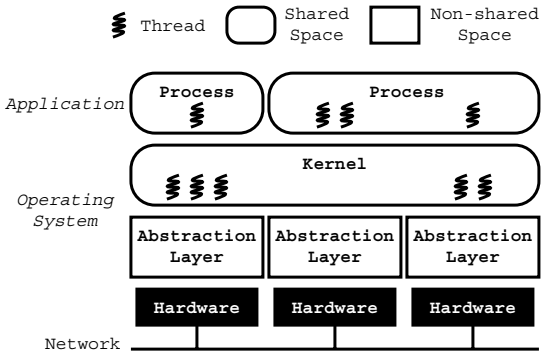


図 1 Solelc の全体構成
Fig. 1 Overall structure of Solelc.

2.1 システムの構成

Solelc の全体構成を**図 1** に示す．Solelc では，OS が抽象化層とカーネルの 2 つの層に階層化されている．

抽象化層は，すべての計算機上で 1 つずつ動作し，計算機資源を抽象化するための機能を実現している．さらに，各計算機の抽象化層が協調することによって，資源の位置を管理し，カーネルが位置透過に資源管理を行うための環境を構築している．抽象化層によって抽象化される計算機資源には，メモリ，CPU，割込み，周辺デバイスの 4 つがある．各資源は，以下のように抽象化される．

- メモリ
  各計算機の物理メモリを，すべての計算機から共有される単一の仮想アドレス空間として抽象化し，位置透過にメモリアクセスを行うことを可能とする．本アドレス空間は，非共有領域と共有領域の 2 つから構成される．非共有領域は，計算機ごとに異なる内容を持ち，抽象化層が配置される．共有領域は，すべての計算機で同一の内容を持ち，カーネルやプロセスが配置される．
- CPU
  CPU 資源を，これを使用する単位となるスレッドとして抽象化し，スレッドを任意の計算機上で動作可能とする．スレッドは，共有領域内のコードを実行するためのものであり，プロセスのコードを実行するユーザスレッドと，カーネルのコー

ドを実行するカーネルスレッドの 2 つに分けられる．

- 割込み
  割込みを，各計算機で発生した事象をカーネルに通知するイベントとして抽象化し，任意の計算機でイベントの取得を可能とする．
- 周辺デバイス
  周辺デバイスは，それら各々に対応したデバイスドライバに抽象化される．抽象化層は，任意の計算機からデバイスドライバを使用可能にすることによって，位置透過なデバイス操作を実現する．

さらに，抽象化層は，抽象化層間で協調処理を行うための通信機能を持つ．抽象化層間の通信には専用のプロトコルが使用される．各抽象化層は，互いに協調処理を行うことによって以下の 2 つを実現する．

- システムで使用可能な資源に，システム全体で一意な整数値である識別子を付与する．
- カーネルからの資源操作の要求を，対象となる資源を持つ計算機の抽象化層に転送する．

カーネルは，システム全体で 1 つであり，抽象化層の機能を使用してすべての計算機を同時に管理する．抽象化層による計算機資源の抽象化は暗黙的に行われるため，カーネルは分散環境を意識することなく位置透過に動作可能である．また，抽象化層による計算機資源への識別子の付与により，カーネルは任意の計算機からすべての資源を操作可能となる．資源操作の要求は抽象化層によって転送されるため，カーネルは，自身が動作する計算機の抽象化層に対して要求を発行することで，すべての資源を操作することが可能である．これにより，カーネルは，システム全体を考慮した資源管理が可能となる．

カーネルが実現する機能は，従来の OS のそれと同様のものである．すなわち，プロセスの実行環境を構築し，資源を適切に複数のプロセスに分配する．また，ファイル操作などのサービスを実現し，これをプロセスに提供する．

プロセスも抽象化層により構築される環境上で動作するため，位置透過に動作可能である．また，任意の計算機からカーネルの機能を同様に使用することが可能であり，効率的な資源の利用が実現される．

2.2 スレッドの処理形態と配置

Solelc では，カーネルやプロセスが共有領域に配置されるため，それらのコードを実行するスレッドを任意の計算機に配置することが可能である．ここで，各々のスレッドは，他のスレッドとの関連や特定デバイスの操作など，さまざまな処理形態を持ち，それぞれ特性が異なる．したがって，スレッドを配置する計算機を無作為に決定するのではなく，各々のスレッドの処理形態に応じて適切な計算機に配置することで，システム全体の処理の効率化を図ることができる．スレッド間の関連の強さに着目すると，スレッドの処理形態として，以下の 2 つがあげられる．

- スレッド間で互いに強く関連する処理
  Solelc では，カーネルやプロセスは共有領域に配置され，すべての計算機から参照可能である．共有変数への頻繁なアクセスや，カーネルによるプロセス領域へのアクセスなど，複数のスレッドが同一領域に頻繁にアクセスする場合，それらのスレッドが異なる計算機に配置されると，メモリの一貫性制御によるページ転送が頻繁に発生し，著しく処理効率が低下する．したがって，スレッド間で互いに強く関連する処理については，それらのスレッドを同一計算機上で動作させることが望ましい．
- 他のスレッドとの関連が弱い処理
  一方，共有変数へのアクセスが少なく，スレッドが他の処理から独立して動作する場合，処理効率は，主に CPU の処理能力に依存する．したがって，このような処理については，それぞれのスレッドを複数の計算機に分散させることで，システム全体の CPU 資源の効率的使用につながり，スレッド実行の効率化を図ることが可能である．

カーネルやプロセスの処理をこのように分類することができれば，Solelc の特徴を活かして効率良く処理を行うことが可能である．プロセスの振る舞いについては，実行されるまでその処理の特性が不明であることが多いため，処理形態を判断することは困難である．一方，カーネルのそれについては，あらかじめその特性が判明しているため，処理形態を判断することも容易である．以降の章では，このような分類に基づいた効率的なファイル管理の構成法について述べる．

3. ファイル管理機構の概要

Solelc におけるカーネルでは，分散環境を意識しない設計が可能であり，任意の計算機からすべての計算機資源を使用可能である．このことから，単一計算機での動作を前提としたファイル管理手法を適用することで，容易にファイルサービスを実現可能である．しかし，前章で述べたようなスレッドの処理形態を考慮せず，ファイル管理の機能を動作させる計算機を無作為に選択してこれにスレッドを配置する手法では，次のような点から，十分な性能を得ることが困難である．

- メモリ一貫性制御によりページ転送が頻繁に発生してしまう．
- 複数の計算機の資源を同時に使用できない．

そこで，本ファイル管理機構では，Solelc の特徴の 1 つであるスレッド配置の自由度の高さを活かしたファイル管理の効率化を図る．具体的には，システムコール処理を細分化し，各処理をその処理形態に応じて適切な計算機に配置する．これにより，メモリ一貫性制御によるページ転送を抑制し，処理効率の低下を抑制することができる．また，独立した処理は他の処理と並列に実行可能であるため，処理の分散・並列化が促進され，処理効率の向上を図ることができる．

例として，ファイルの内容を取得するためのインタフェースとしてユーザに提供される read システムコールについて考える．read システムコールの処理を大きく機能ごとに分類すると以下のようになる．

(1) システムコールの受付
(2) ファイル状態の管理
(3) ディスクからのブロックデータの読出し
(4) プロセス領域へのファイルデータの書込み
(5) システムコール処理の終了通知

この手順において,(2) の処理を行うすべてのスレッドは，ファイル管理データを共有し，これを頻繁に更新する．このことから，この処理は，複数の計算機に分散させず，いずれか 1 台の計算機に配置されることが望ましい．一方,(3) および (4) の処理は，必要な情報が与えられれば，当該スレッド固有のデータ参照のみで完結する．このため，複数の計算機に分散させ，並列に処理を実行することが可能である．

また，一般的に，read システムコールを発行したユーザスレッドは，カーネルから read システムコールの終了通知を受けた直後，読み出したデータにアクセスすることが多い．すなわち,(4) の処理において，データを書き込む対象となるプロセス領域上のバッファは，処理を行うワーカと read システムコールを発行したユーザスレッドとの間で共有されるという見方ができる．したがって,(4) の処理は，read システムコールを発行したユーザスレッドが動作する計算機で行われることが望ましい．

このような考察から，システムコール処理を以下の 2 種類の機能に分割する．

- 1 台の計算機に固定され,(1) および (2) の処理を実行する．続いて，システムコール発行元ユーザスレッドの動作する計算機に対して残りの処理を依頼する．
- すべての計算機に配置され，処理の依頼に従って (3)〜(5) の処理を実行する．

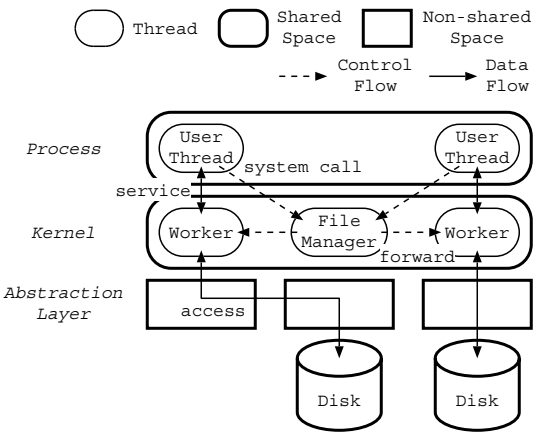


図 2 ファイル管理機構の構成
Fig. 2 Structure of file management.

以上の考察に基づいたファイル管理機構の構成を**図 2** に示す．上記のうち前者をファイルマネージャ，後者をワーカに実行させる．すなわち，ファイルマネージャは，Solelc の管理下にあるいずれか 1 台の計算機に配置され，システム全体のファイルサービスの管理など，主に共有変数を操作する機能を持つ．また，ファイルマネージャは，並列実行可能な処理の要求をユーザスレッドの動作する計算機のワーカに通知する．ワーカは，すべての計算機にそれぞれ 1 つ以上配置される．ファイルマネージャからの通知を受けたワーカは，その要求内容に従って処理を行う．

以上のように，本ファイル管理機構では，システムコールの処理を 1 箇所で行うべき部分と複数の計算機で並列に実行可能な部分に分割し，それぞれ異なる機能として実現している．これによって，処理を分散・並列化させ，CPU 資源を効率的に使用することが可能である．また，ワーカを適切な計算機に配置することで，メモリの一貫性制御によるページ転送を抑制し，ユーザスレッドを効率良く動作させることも可能としている．

さらに，本ファイル管理機構のような構成をとることにより，カーネルに機能を追加することを考えたとき，必要とされる処理形態に応じて柔軟に機能を実現可能である．たとえば，ファイル操作の排他制御などの機能は，集中的に管理を行うことが要求されるため，ファイルマネージャの機能として実装されることが望ましい．また，共有領域上にキャッシュしたデータを使用するといった場合，複数の計算機で共有していることを利用し，ワーカの機能として実装することで効率的なキャッシュの利用を実現することが可能である．

4. ファイルマネージャ

複数のスレッドで共有される変数を頻繁に更新する処理については，複数の計算機で分散して実行した場合，メモリの一貫性制御のためのページ転送が発生して効率が低下する．したがって，このような処理は，なるべく 1 台の計算機で実行されることが望ましい．ファイルサービスにおいては，ファイルの状態などの共有情報の管理がこれに相当する．ファイルマネージャは，Solelc の管理下にあるいずれか 1 台の計算機上で動作し，ファイルサービスにおける管理の機能を担う．ファイルマネージャの機能を以下にあげる．

- システムコールの受付
  ファイル管理の機能は，システムコールとしてプロセスに提供される．ファイルマネージャは，open，read，write，close などのファイルサービスに関するシステムコールを取得する機能を持つ．ユーザスレッドがシステムコールを発行すると，その旨がユーザスレッドが動作する計算機の抽象化層にイベントとして通知され，ファイルマネージャが動作する計算機に転送される．ファイルマネージャは，抽象化層からイベントを受け取り，システムコールとして解釈する．
- ファイルのパス名の解決
  ファイルマネージャは，open システムコールのパラメータとしてユーザスレッドにより指定されたパス名から，ファイル属性などの管理情報を保持するブロックデータの存在するディスクを特定する．続いて，目的のディスクのデバイス ID とブロック番号を基に，抽象化層を介してブロックデータを読み出す．パス名を解釈しながら順にブロックデータを読み出すことで，目的のファイルの情報を取得する．
- ファイル状態の管理
  プロセスが使用する資源は，プロセスごとに資源管理データとしてカーネルによって管理される．ファイルマネージャは，それらの資源のうちファイルの状態を**表 1** に示すような情報により管理する．これらの情報に加え，オープンされている全ファイルエントリ，inode の内容など，複数のファイルをオープンした際に共有される情報を管理することで，基本的なファイル操作の機能が実現される．ユーザスレッドからのシステムコールが open や close などファイルアクセスの前処理/後処理の要求であった場合，ファイルマネージャは，資源管理データに当該ファイルのエントリを追加/削除する．また，システムコールが read や write などファイルアクセスそのものの要求であった場合，後述するようにディスクおよびプロセス領域へのアクセス要求を適切な計算機のワーカに通知し，当該ファイルに対応する資源管理データを更新する．
- ワーカへの要求通知
  ユーザスレッドからのシステムコールが read や write などである場合，ファイルマネージャは，プロセス領域やディスクへのアクセスをワーカに要求する．前章で述べたように，プロセス領域へのアクセスは，ユーザスレッドが動作する計算機のワーカによって行われることが望ましい．ファイルマネージャは，システムコールを発行したユーザスレッドの情報から適切な計算機を選択し，これに要求を通知する．
- システムコール処理の終了通知
  ワーカに処理要求を通知せずファイルマネージャのみでシステムコールの処理を完了した場合，ファイルマネージャがユーザスレッドに対しシステムコール処理の終了を通知する．open システムコールの処理については，終了通知とともに目的のファイルのディスクリプタをユーザスレッドに返す．

表 1 資源管理データ
Table 1 Data for resource management.

Name	Description
dev	ディスクのデバイス ID
blksize	ディスクのブロックサイズ
ino	inode 番号
size	ファイルサイズ
offset	ファイル先頭からのオフセット値

5. ワ ー カ

スレッド間で変数を共有せず互いに独立して実行可能な場合，それらのスレッドは，1 台の計算機ですべて実行されるよりも，複数の計算機で並列に実行されることが望ましい．ファイルサービスにおいては，ディスクへのアクセスなどがこれに相当する．ワーカは，Solelc 管理下の各々の計算機上で動作し，このような処理を複数の計算機で並列に実行する．ファイルサービスにおいて，ファイルマネージャが管理の機能を担うのに対し，ワーカは，その管理下における処理の機能を担う．ワーカの機能を以下にあげる．

- 対象となるディスクへのアクセス
  カーネルは，ファイルのオープン時にそのパス名からファイルのデータを持つディスクを特定し，

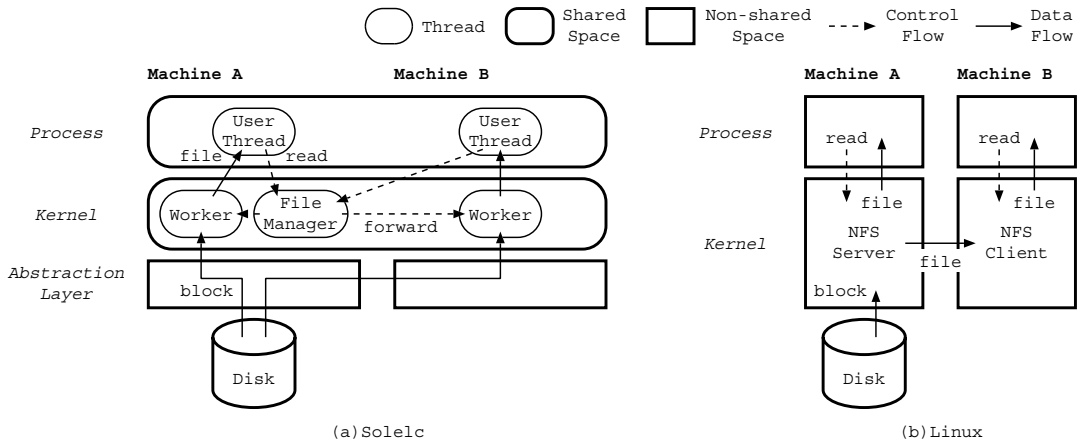


図 3 評価に用いたシステム構成
Fig. 3 System configuration for performance evaluation.

資源管理データを用いてそのディスクのデバイス ID などを管理する．ユーザスレッドからのシステムコールが read や write などである場合，ワーカは，それらの管理データを参照することでユーザスレッドの指定したファイルの存在するディスクを特定し，これにアクセスすることでファイルの読み書きを実現する．

- プロセス領域へのアクセス
  ユーザスレッドは，read や write などのシステムコールを発行する際，そのパラメータとしてバッファのアドレスを指定する．ワーカは，システムコールが read である場合，適切なブロックデータを読み出し，要求された部分をそのアドレスに書き込む．一方，システムコールが write である場合，そのアドレスからデータを読み出し，適切なディスクブロックに書き込む．
- システムコール処理の終了通知
  ファイルサービスに関するシステムコールの処理において，ファイルマネージャからワーカへ要求を通知した場合，システムコール処理の終了通知はワーカによって行われる．

6. 評　　価

Solelc では，Linux が提供しているものと同様のシステムコールをユーザに提供することが可能である．現在，ファイル管理については，プロセスにファイルサービスを提供するための open，read，close システムコールを実装済であり，ディスク内のファイルシステム Ext2fs[3] を解釈してプロセスにファイルを提供することが可能である．

本章では，プロセスによるファイル読出しについての実験を行い，Solelc のファイル管理機構の性能を検証する．また，Linux との比較を行うことによって，効率的なファイル管理が実現されていることを示す．なお，本章における性能評価には，Celeron 667 MHz を搭載した PC/AT 互換機を 100 Mbps のイーサネットで接続した環境を用いている．また，比較対象の Linux については，kernel-2.4.18 を用いている．

評価の具体的な方法としては，**図 3** に示すようなシステム構成において，プロセスが read システムコールを利用して各々の計算機からファイルデータを読み出し，そのデータにアクセスするのに要する時間を計測した．本実験では，図 3（a）に示すように，ファイルマネージャおよびディスクが計算機 A に配置され，ユーザスレッドが計算機 A，B それぞれに 1 つずつ配置される．Solelc においてユーザスレッドの負荷分散を考えたとき，各々の計算機に順にスレッドが配置されるのが自然であり，この構成は最も一般的なものであると考えられる．ユーザスレッドが読み出すファイルのデータはすべて計算機 A のディスクに格納されており，各計算機のワーカが抽象化層を介して同一のディスクからブロックデータの読出しを行う．一方，Linux では，図 3（b）に示すように，ディスクが計算機 A に配置され，プロセスが計算機 A，B それぞれに配置される．計算機 A，B がそれぞれ NFS のサーバ，クライアントの関係にあり，ディスク内のファイルシステムの内容は，NFS マウントによって計算機 A から計算機 B に提供される．このとき，計算機間のデータ転送については，図 3 において対比されるように，Solelc では特別な意味を持たないブロックデータを単位として行われるのに対し，Linux では NFS によってファイルという意味を持ったデータを対象とし

て行われる．また，Solelc，Linux のいずれにおいても，ディスク内のファイルシステムとして Ext2fs を用い，PIO 転送モード 4 によってディスクからブロックデータを読み出す．なお，ディスクブロックサイズは，1 ブロックあたり 1 キロバイトとなっている．

まず，本論文で提案する手法の効果を示す．比較対象として，Solelc のメモリ管理の特徴を考慮しない以下の 2 つの手法を用いる．

- 手法 A ··· 計算機 A ですべてのシステムコール処理を行う．
- 手法 B ··· システムコールが発行された計算機ですべての処理を行う．

提案手法と 2 つの手法を比較するために，それぞれの手法ごとに，以下の 2 つの場合における処理の所要時間を計測した．

(1) 計算機 A においてのみ read システムコールが発行された場合
(2) 計算機 B においてのみ read システムコールが発行された場合

また，計算機 A および B から同時に read システムコールが発行された場合において，以下の処理の所要時間を計測した．

(3) 計算機 A から発行された要求についての処理
(4) 計算機 B から発行された要求についての処理

3 つの手法ごとに上記の 4 つの処理時間を計測した結果を**表 2** に示す．(1) および (2) では，複数の要求が同時に発生しないため，それぞれの手法で処理の所要時間に大きな差はみられない．両方の計算機から同時にシステムコールが発行される (3) および (4) の場合，手法 A では，計算機 B からの要求についての処理でプロセス領域のページ転送が頻繁に発生する．そのため，特に (4) の所要時間が増加している．一方，計算機 A からの要求については，メモリページ転送が発生しないため，短時間で処理が完了する．手法 B では，両方の計算機でカーネルスレッドが資源管理データを頻繁に更新するため，カーネル領域のページ転送が頻繁に発生する．この結果，(4) の所要時間が非常に大きなものとなっている．ここで，計算機 B からの要求の処理中に発生するブロックデータの読出しの間，計算機 B の処理がブロックされるため，メモリ一貫性制御のためのページ転送が抑制される．このことから，計算機 A からの要求についての処理は妨げられることがなく，計算機 B からの要求より短時間で処理が完了する．

提案手法では，Solelc のメモリ管理の特徴を考慮し，手法 A および B においてみられたページ転送を抑制することで，効率化を実現している．(3) の処理時間では手法 A に劣るが，適切に計算機 B の処理に時間を分配することで，(4) の処理時間を削減している．特に，(2) と (4) を比較すると，読出しサイズが 1,000 キロバイトのとき，手法 A，B においてそれぞれ約 8.0 倍，30.3 倍もの処理時間を要しているのに対し，提案手法における所要時間の増加は 1.3 倍程度となっている．

表 2 各手法における所要時間
Table 2 Execution time of each processing scheme.

Processing Scheme		Data Size (KB)			
		0	100	500	1,000
提案手法	(1)	0.48	26.93	134.71	270.79
	(2)	0.40	48.57	240.41	505.71
	(3)	1.75	59.99	337.70	665.07
	(4)	1.10	59.22	336.90	664.01
手法 A	(1)	0.48	26.87	134.75	303.34
	(2)	0.42	52.33	256.93	511.19
	(3)	1.75	31.00	150.99	341.62
	(4)	1.30	236.75	1,699.86	4,070.86
手法 B	(1)	0.48	27.16	135.60	303.07
	(2)	0.42	49.70	240.97	480.69
	(3)	1.57	78.91	408.81	810.89
	(4)	1.11	718.19	6,632.02	14,576.03

(ms)

次に，Solelc と Linux を比較することで，提案手法の基本性能を示す．上記 (1) 〜 (4) についての Solelc，Linux における計測結果をそれぞれ**図 4**，**図 5** に示す．それぞれ読出しデータサイズ 50 キロバイトごとに計測し，所要時間をミリ秒単位で示している．異なる OS 間での絶対的な処理時間の比較に意味はないが，処理内容の違いから生じる処理時間の変化を比較することで，本ファイル管理機構の特徴に関して以下のような考察が得られる．

すべての場合について，Solelc における所要時間は読出しサイズにほぼ比例している．これは，プロセスから要求されたデータサイズによらず，カーネルが 1 キロバイトごとにディスクからブロックデータを読み出し，プロセス領域にファイルデータを書き込むためである．一方，Linux においては，ファイルデータが格納されているディスクブロックの連続状態によって 1 度にディスクから読み出されるブロック数が異なることや，ブロックデータの先読みを行うことから，所要時間の揺れが激しくなっている．

(1) および (2) の所要時間は，1 つのシステムコールの処理のみに費された時間であり，ファイル管理の基本性能を表している．(1) の場合，通信をともなわず，すべての処理が 1 台の計算機で完結する．(2) の場合，データ送受信のための通信が発生するため，(1)

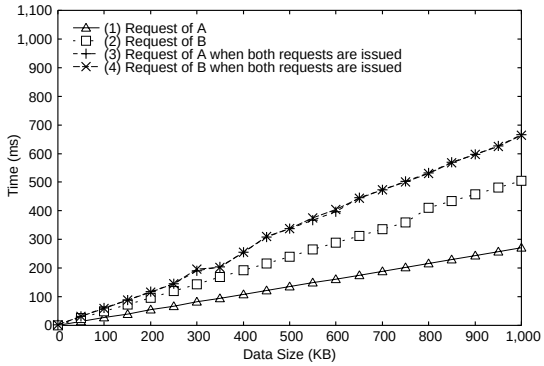


図 4 Solelc における所要時間
Fig. 4 Execution time on Solelc.

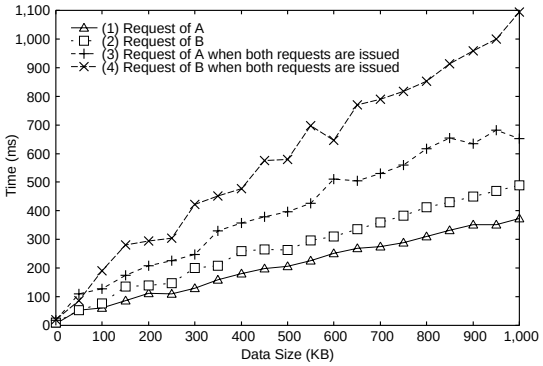


図 5 Linux における所要時間
Fig. 5 Execution time on Linux.

の場合よりも処理に時間を要している．これらの基本性能を基に（3）および（4）の処理時間について検討することで，複数の要求を同時に処理することにより発生するオーバヘッドが明らかになる．

（3）および（4）の所要時間は，複数の計算機から同時にファイルの読出し要求が発行された場合のものであり，システム全体の資源管理の効率を表している．複数の要求を同時に処理するため，Solelc，Linux のいずれにおいても，（1）や（2）の場合と比較して処理に時間を要する．Linux においては，ブロックデータを読み出し，ファイルとして解釈するまでの処理がすべて計算機 A で行われる．このことから，計算機 A の負荷が増大し，計算機 B から発行された要求の応答性能が大きく低下する．この結果，読出しサイズが 1,000 キロバイトのとき，（2）および（4）の所要時間がそれぞれ 488.85 ms，1,094.74 ms となり，計算機 A の負荷の増大によって約 2.2 倍もの処理時間を要している．一方，Solelc においては，必要なディスクブロックの算出や，取得したブロックのうち要求された部分のバッファへの書込みなど，ブロックの集合からファイルを構成してユーザに提供する処理は各計算機で並列に行われる．したがって，Linux において発生した負荷の集中を抑制することが可能である．実際の所要時間としては，（3）および（4）のものがほぼ同一となっている．2 つのシステムコール処理が同時に行われる場合，いずれの処理についてもファイルマネージャの機能は計算機 A で実行される．このことから，ファイルマネージャの処理に時間を要するとき，（4）に比べ（3）の所要時間が増大するはずである．したがって，この結果は，ファイルマネージャの処理時間が十分に小さいことを意味している．また，読出しサイズが 1,000 キロバイトのとき，（2）および（4）の所要時間がそれぞれ 505.71 ms，664.01 ms であり，計算機 A の負荷の増大に対して所要時間は 1.3 倍程度となる．

Solelc においては，カーネルからのデバイス操作要求が抽象化層間で転送されることから，この転送時間を考慮することで図 3（a）と異なるディスク構成についても処理時間が予測可能である．たとえば，計算機 B にディスクが接続されており，このディスクからブロックデータを読み出す処理を考えたとき，計算機 A からの要求については処理時間が増加し，計算機 B からの要求については処理時間が減少するといった単純なものとなる．このことから，図 3（a）の構成における（4）が最も時間を要する処理であることが分かる．すなわち，他のディスク構成を考えたときは，これよりも短い時間で処理を完了することが予測される．

以上のように，Solelc のファイル管理は，実用可能な基本性能を有し，システム内の各計算機にユーザスレッドが配置されるという Solelc における一般的な構成において効果的である．また，2 台の計算機で構成されるシステムにおいては，処理時間の増加率が低いことを示した．計算機の台数を増加させた場合においても，システムコール処理の大部分を各計算機で並列に実行可能であることから，複数の計算機で負荷を分散し，効率的なファイル管理を実現している．

7. 関 連 研 究

位置透過なファイル操作をユーザに提供するファイル管理手法について，これまでにさまざまなシステムで研究が行われてきた．特に，ディスク内の物理ブロックに格納されたデータを論理的な木構造のファイルシステム[4] として認識し，複数のディスクを 1 つのディレクトリツリーに合成してユーザに提供する手法は，近年の多くのシステムで用いられている．

分散環境では，ネットワークに接続された複数の計

算機ですべてのディスクを共有し，ファイル操作の完全な位置透過性を実現することが望まれる．そのような要望に対して，他の計算機のファイルシステムを自身のファイルシステムの一部として操作可能とするNFS[5]を使用するシステムが多くみられる．NFSは，システム内のすべてのファイルを共有するものではなく，完全な位置透過性は実現していない．しかし，その適用範囲の広さから，非常に有用なシステムであるといえる．一方，完全な位置透過性を持つファイル操作をユーザに提供するシステムとして，Sprite[6]がある．Spriteでは，ファイルシステムの木構造をドメインと呼ばれる単位に分割し，各ドメインをそれぞれ異なる計算機で管理する．ディレクトリツリーを辿る際，プレフィックステーブルを基に目的のファイルの存在するドメインを検索し，そのドメインを管理する計算機に要求を通知することでパス名の解決を行う．このように，複数の計算機で協調することで，システム全体で 1 つのディレクトリツリーを構成し，完全な位置透過性を実現している．Solelcにおいても，システム全体で 1 つのディレクトリツリーを構成している．したがって，Solelcにおけるファイル操作は，完全な位置透過性を持つ．Solelcでは，ユーザに位置透過なファイル操作を提供するだけでなく，カーネル自体が位置透過にディスクを管理可能である．このことから，Solelcのファイル管理機構は，容易に設計可能であり，簡素なものとなっている．

分散環境におけるファイル操作は，ブロックやファイルデータのキャッシュまたは複製を保持することにより，高速化が図られる．キャッシュや複製の実現手法として，Spriteでは，ドメインを管理するサーバとそれを利用するクライアントの両方の計算機にキャッシュを保持する．これにより，ディスクアクセスと通信の回数を削減し，高速なファイル操作を実現している．Locus[7]では，ファイルの複製を複数の計算機のディスクに格納し，ファイル操作を並列に行うことを可能としている．ファイル操作の際に複製の同期を管理する計算機が介入することで，最新のファイルにアクセスすることが保証される．Locusでは，ファイル更新のディスクへの反映は，ファイル操作がすべて完了したのちに行われる．これにより，ファイル更新処理の原子性を実現している．また，複製管理によって高度なスケーラビリティを実現するシステムとして，Andrew[8),9)]がある．Andrewでは，1 台のサーバと複数のクライアントでクラスタを構成する．クライアントは，ファイルのオープン時に自身が属するクラスタのサーバからファイルを取得し，これを自身のディスクにキャッシュする．ファイルへの読み書きをキャッシュに対して行うことで，ファイル操作における通信回数が削減される．Andrewでは，このようなファイルへの読み書きの効率化をはじめ，さまざまな手法により通信負荷の低減を図り，スケーラビリティの大幅な向上を実現している．Solelcでは，キャッシュについては現在検討中であるが，共有領域上にキャッシュの領域を設けることですべての計算機で利用可能となり，さらなるファイル管理の効率化が期待できる．この際，計算機間のキャッシュの一貫性を考慮する必要があるが，Solelcにおいて実現されているメモリ管理が適用されるため，容易に一貫性を保持することが可能である．

さらに，Solelcでは，カーネル自体が位置透過に資源を管理可能であることから，従来のシステムで用いられてきたファイルシステムを完全に位置透過なものとして使用することが可能である．独自のファイルシステムを設計・実装することで，ファイルの移動透過性も実現可能である．

8. おわりに

本論文では，分散オペレーティングシステム Solelcにおけるファイル管理機構の構成法とその性能評価について述べた．本ファイル管理機構では，システムコール処理を Solelc の処理特性に合わせて分割し，各処理を適切な計算機で実行させる．これによって，特定の計算機に負荷を集中させることなく，効率的なファイル操作をプロセスに提供することが可能である．今後は，ディスクブロックやファイルを単位としたキャッシュ管理手法や，独自のファイルシステムによる効率的なファイルサービスについて検討を行う予定である．

参考文献

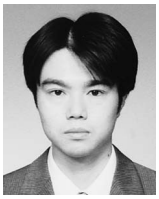
1) 芝 公仁，大久保英嗣：分散オペレーティングシステム Solelc の設計と実装，電子情報通信学会論文誌 D-I，Vol.J84–D–I，No.6，pp.617–626 (2001).
2) 芝 公仁，大久保英嗣：分散オペレーティングシステム Solelc におけるメモリ管理手法，情報処理学会論文誌，Vol.42, No.6, pp.1460–1471 (2001).
3) Ext2fs Home Page, http://e2fsprogs.sourceforge.net/ext2.html.
4) Rosenblum, M. and Ousterhout, J.K.: The Design and Implementation of a Log-Structured File System, *ACM Trans. Comput. Syst.*, Vol.10, No.1, pp.26–52 (1992).
5) Sandberg, R.: The Sun Network Filesystem: Design, Implementation, and Experience, *Proc.*

Summer 1986 USENIX Technical Conference and Exhibition (1986).
6) Ousterhout, J.K., Cherenson, A.R., Douglis, F., Nelson, M.N. and Welch, B.B.: The Sprite Network Operating System, *Computer Magazine of the Computer Group News of the IEEE Computer Group Society, ACM CR 8905-0314*, Vol.21, No.2 (1988).
7) Walker, B., Popek, G., English, R., Kline, C. and Thiel, G.: The LOCUS distributed operating system, *Proc. 9th ACM symposium on Operating systems principles*, pp.49–70 (1983).
8) Morris, J.H., Satyanarayanan, M., Conner, M.H., Howard, J.H., Rosenthal, D.S. and Smith, F.D.: Andrew: a distributed personal computing environment, *Comm. ACM*, Vol.29, No.3, pp.184–201 (1986).
9) Howard, J.H., Kazar, M.L., Menees, S.G., Nichols, D.A., Satyanarayanan, M., Sidebotham, R.N. and West, M.J.: Scale and performance in a distributed file system, *ACM Trans. Comput. Syst. (TOCS)*, Vol.6, No.1, pp.51–81 (1988).

(平成 14 年 12 月 21 日受付)
(平成 15 年 4 月 13 日採録)

水口 孝夫（学生会員）

昭和 52 年生．平成 14 年立命館大学大学院理工学研究科博士課程前期課程修了．現在，同大学大学院理工学研究科博士課程後期課程に在学中．オペレーティングシステム，分散システム等に興味を持つ．



芝 公仁（正会員）

昭和 49 年生．平成 14 年立命館大学大学院理工学研究科博士課程後期課程修了．同年龍谷大学理工学部助手となり，現在に至る．博士（工学）．オペレーティングシステム，分散システム，実時間システム等の研究に従事．電子情報通信学会，IEEE 各会員．

毛利 公一（正会員）

昭和 47 年生．平成 11 年立命館大学大学院理工学研究科博士課程後期課程修了．同年東京農工大学工学部情報コミュニケーション工学科助手．平成 14 年立命館大学理工学部情報学科講師となり，現在に至る．工学博士．オペレーティングシステム，実時間システム，マルチメディアシステム，インターネット等の研究に従事．電子情報通信学会，日本ソフトウェア科学会，ACM，IEEE-CS 各会員．

大久保英嗣（正会員）

昭和 26 年生．昭和 52 年北海道大学大学院工学研究科情報工学専攻修士課程修了．同年（株）日立製作所ソフトウエア工場入所．主として FORTRAN コンパイラの開発に従事．昭和 54 年京都大学工学部情報工学科助手．昭和 60 年同講師．昭和 62 年同助教授．平成 3 年立命館大学理工学部情報学科教授となり，現在に至る．工学博士．オペレーティングシステム，データベースシステム，分散システム，実時間システム等の研究に従事．電子情報通信学会，日本ソフトウェア科学会，システム制御情報学会，ACM，IEEE-CS 各会員．