システムソフトウェアと
オペレーティング・システム 87−9
（２００１．　６．２８）

分散オペレーティングシステムSolelcにおける ファイル管理手法

水口 孝夫 †　芝 公仁 †　大久保 英嗣 ††

† 立命館大学大学院理工学研究科　†† 立命館大学理工学部

現在, 我々は, 分散オペレーティングシステム Solelc を開発している. Solelc では, 単一のオペレーティングシステムでネットワーク上の複数の計算機を管理する. 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ため, オペレーティングシステムの機能は分散環境を意識することなく実現可能である. これまでに, この特徴を活かしたファイル管理を実現してきた. しかし, ファイル管理における処理の効率化に関しては十分に検討されていない. 本稿では, Solelc の資源管理を考慮し, ファイル管理をより効率的に行う手法について述べる.

A Processing Scheme for File Management on Solelc Distributed Operating System

Takao Mizuguchi †　Masahito Shiba †　Eiji Okubo ††

†Graduate School of Science and Engineering, Ritsumeikan University
††Faculty of Science and Engineering, Ritsumeikan University

We have been developing Solelc distributed operating system. Plural computers which Solelc works on are managed by an operating system. Cooperative work of the abstraction mechanism on each computer provides an environment for location transparent resources management. In this environment, functions of operating system are realized without considering the distributed environment. Making use of these characteristics of Solelc, the file management system has been constructed with ease. For efficient processing, however, the location where requests should be handled has to be considered. In this paper, an efficient processing scheme for file management is described.

1 はじめに

近年, 計算機の低価格化と高速ネットワークの普及に伴い, 複数の計算機をネットワークで相互に接続したシステムを利用する機会が多くなってきている. しかし, 従来のシステムでは, 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は, その OS が動作する計算機上のものに限定されている. そのため, 複数の計算機を同時に使用する場合, 計算機ごとに資源管理が行われ, システム全体を考慮した資源管理が困難となっている. また, これらの計算機は環境に差異があり, それぞれ処理能力や接続されるデバイスが異なる. そこで, 我々は, これらの様々な環境を持つ計算機群を統一して管理する OS の構築を考えている.

我々が開発している分散 OS Solelc では, 従来の OS と異なり, 単一の OS でネットワーク上の複数の計算機を管理する [1]. 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動作を行うことにより, 位置透過な資源管理を可能とする環境を構築する. すべての計算機から共有される単一仮想アドレス空間を構築し, その共有アドレス空間上で OS を動作させることにより, 任意の計算機で動作している機能がすべてのプロセスから利用可能となる. したがって, すべての計算機に同一の環境が実現され, プロセスは任意の計算機で他のすべての計算機と同様の機能を利用することが可能となる.

Solelc においては, 抽象化機構によって計算機資源の抽象化が行われるため, OS の機能は分散環境を意識することなく実現可能である. これまでに, この特徴を活かしたファイル管理を実現してきた [2]. しかし, ファイル管理における処理の効率化に関しては十分に検討されていない. そこで, 今回, Solelc の資源管理を考慮し, データの流れや処理の分散・並列化の面でファイル管理をより効率的に行う手法について検討した.

以下, 本稿では, 2 章で Solelc の概要を述べ, 3 章でファイル管理の効率化について述べる. 次に, 4 章で効率化手法を用いたファイル管理の処理の流れについて述べ, 5 章で評価を行い, 最後に 6 章で本稿のまとめと今後の予定について述べる.

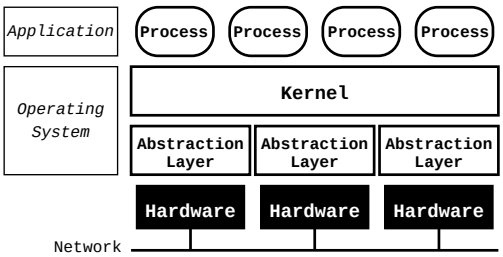

図 1 システムの全体構成

2 Solelc の概要

我々が開発している分散 OS Solelc では, 従来の OS と異なり, 単一の OS でネットワーク上の複数の計算機を管理する. 各計算機の抽象化機構がネットワークを介して協調動作を行うことにより, 計算機資源の位置透過性を実現する. Solelc において, プロセスは, 位置透過に OS の機能を利用することができる.

2.1 Solelc の構成

Solelc の構成を図 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. 下位層の抽象化層は, すべての計算機で 1 つずつ動作し, カーネルが動作するための環境を提供する. 上位層のカーネルは, 抽象化層が提供する機能を使用してシステム全体の資源を管理する. また, カーネルは, プロセスの実行環境を構築し, システムが持つ資源を各プロセスに適切に分配する.

計算機資源の抽象化を行う際に必要となる機能として, メモリ管理, プロセス・スレッド管理, イベント管理, デバイス管理の 4 つがある. それぞれの概要を以下に示す.

- メモリ管理
  すべての計算機から共有される仮想アドレス空間を構築する. Solelc における仮想アドレス空間の構成を図 2 に示す. 仮想アドレス空間は, 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. 一方, 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.

- プロセス・スレッド管理
  カーネルやプロセスのコードを実行するスレッドとして CPU を抽象化する. 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

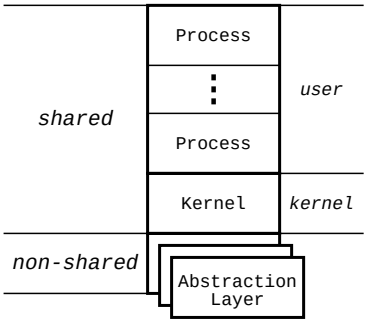

図 2　アドレス空間の構成

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. カーネルスレッドは, 共有領域内のすべてのメモリに読み書きすることが可能である. 一方, ユーザスレッドは, 自身が属するプロセスのメモリにのみアクセス可能である. プロセスは複数のユーザスレッドを持つことができ, 同一のプロセスに属するおのおののスレッドをそれぞれ異なる計算機で動作させることも可能である.

- イベント管理
  CPU が発生させる内部割り込みや新たな計算機の追加など, 非同期に発生する事象は, イベントとして抽象化され, カーネルに通知される. 各計算機上で発生したイベントが転送されカーネルに渡される様子を図 3 に示す. イベントは, その種類ごとに作成されるイベントキューに繋がれ管理される. イベントを処理するカーネルスレッドからの要求により, イベントキューの先頭のイベントがカーネルスレッドに渡される.

- デバイス管理
  任意の計算機からデバイスドライバを使用可能とすることで, 任意のデバイスへのアクセスを実現する. カーネルからの要求を受けた抽象化層は, 指定されたデバイスを持つ計算機にこの要求を転送する. また, デバイスドライバの処理結果も, 要求元の抽象化層に転送されカーネルに渡される.

抽象化層が計算機資源の抽象化を行うことにより, プロセスは, 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. また, カーネルも位置透過

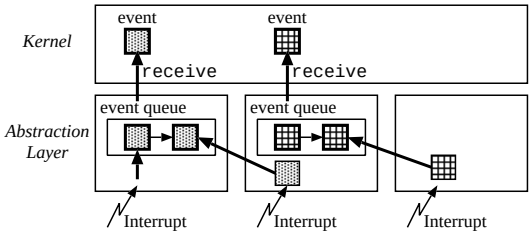

図 3　イベント管理

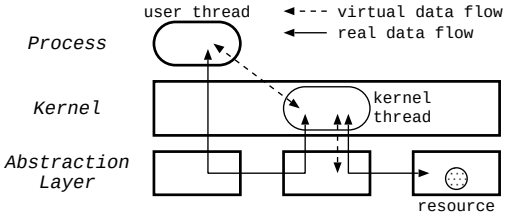

図 4　サービスの提供

にふるまい, 任意の計算機上の資源を利用可能である. Solelc におけるカーネルは, 従来のシステムにおけるカーネルと同様の役割を果たす. ただし, 抽象化層が実現する環境上で動作し, 複数の計算機を同時に管理するという点において, 従来のものとは異なる.

Solelc において, カーネルの機能の提供は, 基本的に図 4 に示す方式に統一して実現されている. ここで, システムは, 3 台の計算機で構成されており, プロセスを実行するユーザスレッドとカーネルスレッドが異なる計算機で動作しているとする. また, カーネルがプロセスに機能を提供する際に必要となる資源も異なる計算機上に存在するものとする. 図の破線矢印は, カーネルから見た仮想的なデータの流れを表している. また, 実線矢印は, 計算機資源の抽象化により暗黙のうちに行われる実際のデータのやりとりを表している. ユーザスレッドがシステムコールを発行してから終了通知を取得するまでの手順は次のようになる.

(1) プロセスがシステムコールを発行する. システムコールは, 抽象化層を経て, そのシステムコールを処理すべきカーネルスレッドに転送される.

(2) カーネルスレッドがプロセスに対してサービスを提供する. このとき, カーネルスレッドは, 必要に応じて自身の計算機の抽象化層に要求を発行し, 適切な計算機資源にアクセスする.

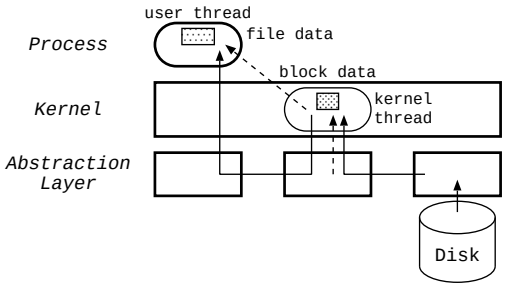

図 5　Solelc における従来のファイル管理

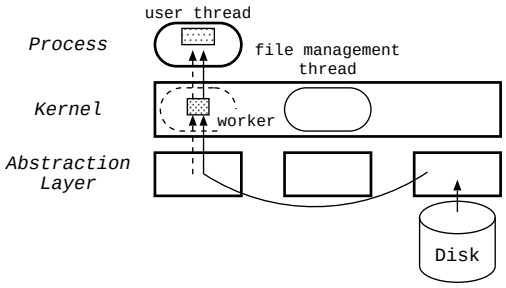

図 6　ファイル管理の効率化

(3) カーネルスレッドは, 処理の終了をプロセスに通知する.

Solelc において, カーネルの機能は, 分散環境を意識することなく実装可能である. このとき, カーネルスレッドは, ユーザスレッドや計算機資源の位置に依らずふるまう. しかし, 実際には, 計算機資源の抽象化により, 通信を伴う処理が頻繁に行われている. 例えば, 上記の手順 (1) において, プロセスは, カーネルスレッドの動作する計算機を意識することなくシステムコールを発行する. また, 手順 (2) において, カーネルスレッドは, 資源の位置を意識せず, 自身の計算機の抽象化層に要求を発行するのみである. しかし, いずれの手順においても, 抽象化層が資源の位置を管理し, 適切に通信を行うことで処理を実現している.

2.2　Solelc における従来のファイル管理

Solelc では, 前節で述べた Solelc の特徴を利用することで, ファイル管理を実現している. 具体的には, ファイル管理のイベントを処理する機能を持つカーネルモジュールと, 2 次記憶装置に対するブロックデータ入出力を管理するデバイスドライバを実装することで実現可能である. Solelc における従来のファイル管理については, カーネルモジュールとしてファイル管理モジュール, デバイスドライバとしてディスクドライバを実装している.

Solelc における従来のファイル管理では, ファイル管理モジュールは, 要求元ユーザスレッドやディスクの位置を意識することなくファイル管理を行っている. ファイル管理モジュールがディスクからブロックデータを読み出してプロセスに提供する様子を図 5 に示す. ここで, システム構成は, 図 4 の場合と同様であり, 対象となる計算機資源がディスクであるとする.

ファイル管理モジュールが図の破線矢印で示すようにデータを扱う一方で, 実際のデータの流れは実線矢印で示すようになる. すなわち, データがディスクからプロセスに到達するまでに, ネットワークを 2 回経由することになる. これは, ファイル管理モジュールが分散環境を意識せずに動作可能である反面, ブロックデータの流れが非効率となる可能性を含むことを意味する. 次章では, カーネルが適切に分散環境を意識することによって, ファイル管理を効率化する手法について述べる.

3　ファイル管理の効率化

Solelc におけるカーネルの機能は, 抽象化層によって計算機資源の抽象化が行われるため, 分散環境を意識することなく実現可能である. しかし, 処理効率に着目すると, 処理を行う計算機を考慮する必要があり, そのためには抽象化層の動作を意識しなければならない. 今回, Solelc の資源管理を考慮し, データの流れや処理の分散・並列化の面でファイル管理をより効率的に行う手法について検討した.

Solelc における従来のファイル管理では, ファイルデータは, ディスクからプロセスに到達するまでに, 一旦ファイル管理モジュールの領域に置かれる. ファイル管理モジュールは, ディスクおよび要求元ユーザスレッドの位置を意識することなくふるまうため, ファイル管理モジュールを実行するカーネルスレッドがディスクおよびユーザスレッドと異なる計算機で動作しているとき, 2 段階の通信を伴う. そこで, ファイル管理モジュールが要求元ユーザスレッドの動作する計算機のワーカスレッドに処理の一部を依頼することを考えた. ファイル管理の効率化を考慮した構成を図 6 に示す. システム構成は, 図 5 の場合と同様であるとする. カーネルスレッドであるファイル管理スレッドおよび

ワーカスレッドがファイル管理モジュールの処理を分担して実行することで, 全体として従来と同様のサービスを実現する. ここで, ユーザスレッドからの要求の管理, ファイル状態の管理などについては 1 箇所で行うべきである. 一方, 実際にファイルデータを扱う処理については, システム全体の処理状況を管理することで, 複数箇所で並列に実行可能となる. したがって, 分担は大きく次のようになる.

- ファイル管理スレッド
  - ファイル操作要求の受付
  - ファイル状態の管理
  - 処理状況管理
- ワーカスレッド
  - ファイルデータおよびブロックデータの入出力処理

ユーザスレッドと異なる計算機で動作するファイル管理スレッドに代わり, ユーザスレッドと同一の計算機で動作するワーカスレッドがファイル操作を行うことで, ファイルデータの流れの効率化を図る. 本手法では, データがディスクからプロセスに到達するまでのネットワーク経由回数が 1 回で済むことになる.

プロセスからの read システムコールの処理において, ファイル管理スレッドからワーカスレッドへの処理の移行は, 次の手順を辿ることで実現される.

(1) ファイル管理スレッドは, ユーザスレッドと同一の計算機で動作するワーカスレッドに対し, ファイル操作を依頼する.

(2) 依頼を受けたワーカスレッドは, プロセスに要求されたデータを含むブロックを適切なディスクから読み出す.

(3) さらに, プロセスの領域にデータをコピーし, 処理の終了をプロセスに通知する.

この手法により効率化できる点として, 例えば次のようなものが挙げられる.

- ブロックデータのコピーに伴う通信の削減
  従来のファイル管理では, ファイル管理モジュールからプロセス領域へのデータコピーの際, メモリの一貫性制御のための通信を伴う場合があった. 一方, 本手法を用いたファイル管理では, ファイル操作を行うワーカスレッドがユーザスレッドと同一計算機上に存在するため, データコピーを行う際に通信を伴わない.
- 処理の並列・分散化
  ワーカスレッドを用いることで, ファイル操作を各計算機で独立して実行できるため, 複数計算機上の複数ユーザスレッドからの要求を並列に処理することが可能となる. また, ファイルの管理と操作の機能を分散させることにより, 複数の要求の処理による応答性能の低下を抑えることができる.

本手法を用いたファイル管理を実現するには, 従来の抽象化層およびカーネルの機能を見直し, 適切に拡張しなければならない. 以下, ファイル管理に関連する機構とその拡張について述べる.

3.1 抽象化層

抽象化層では, 計算機固有の資源を管理する機能と, カーネルが指定した計算機の抽象化層にイベントを転送する機能が必要となる.

- イベントテーブル
  プロセスの発行したシステムコールは, 抽象化層ではイベントとして管理される. 各計算機の抽象化層では, イベントテーブルにより, イベント処理を行うカーネルスレッドの動作する計算機の情報を管理する. 抽象化層は, このテーブルを参照し, 適切な計算機にイベントを転送する.
- イベントキュー
  適切な計算機に転送されたイベントは, 抽象化層内でイベントの種類毎にイベントキューに繋がれ, 管理される. カーネルスレッドが抽象化層に要求すると, 抽象化層はイベントキューからイベントを取り出し, カーネルスレッドに渡す.
- 処理要求の転送
  カーネルスレッドからの要求により, 抽象化層は, ユーザスレッドの動作する計算機に要求を転送する. 転送された要求は, 計算機指定イベントとして転送先計算機の抽象化層内で管理される.

- 計算機指定イベントキュー
  各計算機の抽象化層が計算機指定イベントキューを持ち, おのおのの計算機指定イベントを管理する. カーネルのワーカスレッドが抽象化層に要求すると, 抽象化層はこのキューからイベントを取り出し, ワーカスレッドに渡す.

- デバイスリスト
  カーネルが発行したドライバへの要求は, 抽象化層により適切な計算機に転送される. 各計算機の抽象化層では, デバイスリストにより, デバイスを持つ計算機の情報を管理する. 抽象化層は, このリストを参照することで, 指定されたデバイスの存在する計算機を知ることができる.

3.2 カーネル

抽象化層により計算機資源の位置透過性が実現されるため, カーネルでは, プロセスにサービスを提供する機能のみを実現する. ただし, 計算機を指定して行う処理については, それぞれの計算機でスレッドを動作させる必要がある.

- ファイル管理スレッド
  ファイル管理モジュールを実行し, ユーザスレッドからのファイル操作要求を処理する. ディスクアクセスおよびファイルデータのコピーを伴う read, write システムコールの処理を行う際, 要求元ユーザスレッドの動作する計算機のワーカスレッドに処理を依頼する. ファイル管理スレッドは, ファイル操作要求の受付およびファイル状態管理を行う. ファイル管理スレッドが管理する情報としては, ファイルディスクリプタ, モードおよびサイズなどがある. また, ワーカスレッドへの処理の依頼に伴い, システム全体の処理状況の管理を行う.

- ワーカスレッド
  抽象化層から計算機指定イベントを取得し, ファイル管理スレッドにより依頼された処理を行う. 具体的には, プロセスからの read および write 要求に対し, プロセスとディスクの間でファイルデータの入出力処理を行う. また, ユーザスレッドに処理の終了を通知する.

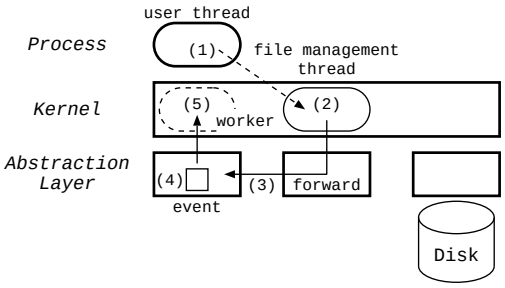

図 7 ワーカスレッドへの処理要求

4 処理の流れ

プロセスが read システムコールを発行してからワーカスレッドに制御が移行するまでの手順を以下に示す. また, 図 7 にその様子を示す.

(1) プロセスが read システムコールを発行する. このシステムコールは, 抽象化層を経て, ファイル管理モジュールに転送される.

(2) ファイル管理モジュールは, 取得した要求が read 要求であることを知り, 要求の転送を抽象化層に依頼する.

(3) 抽象化層は, ユーザスレッドの動作する計算機へ要求を転送する.

(4) 要求を取得した計算機の抽象化層は, 計算機指定イベントとしてこれを計算機指定イベントキューに繋ぐ.

(5) ワーカスレッドからの指示を受け, 抽象化層は, 計算機指定イベントキューからイベントを取り出し, ワーカスレッドに渡す.

以上の手順により, ファイル操作処理をファイル管理モジュールからワーカスレッドに移行することができる. 続いて, ワーカスレッドによる処理の手順を以下に示す. また, 図 8 にその様子を示す.

(6) ワーカスレッドは, 指定されたファイルを含むブロックデータを取得するため, ブロックデータ読出し要求を発行する. この要求も抽象化層に通知される.

(7) 抽象化層は, デバイスリストを参照して必要なデバイスが存在する計算機に要求を転送する.

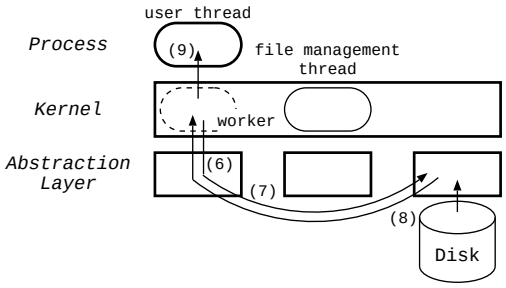

図 8　ワーカスレッドによる read 処理

(8) 要求を取得した計算機の抽象化層は, デバイスドライバを通してディスクからブロックデータを読み出し, これを要求元計算機に送信する.

(9) ブロックデータを取得した計算機のワーカスレッドは, 要求されたデータをプロセス領域にコピーし, プロセスに read 処理終了を通知する.

以上の手順により, ワーカスレッドを利用し, 処理の効率を考慮したファイル管理が実現される. プロセスは, 従来のファイル管理手法を用いた場合と同様にファイルサービスを利用することができる.

5　評価

本章では, プロセスからの要求の処理時間を計測し, これまで述べてきた効率化を行ったファイル管理手法と従来のファイル管理手法との比較を行う. 具体的には, プロセスが read システムコールを利用して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. 評価に用いた計算機は, Celeron 667MHz の PC であり, 100Mbps のイーサネットで接続されている. また, 今回の評価には RAM ディスクを用いている.

Solelc では, read システムコールを発行したユーザスレッド, ファイル管理スレッド, ディスクの位置関係により, ファイル管理の処理に要する時間が大きく異なる. ここでは, ファイル管理の効率化を図ることによる影響を最も大きく受けると考えられる以下の 2 つの場合について処理時間を計測した.

(1) ユーザスレッドとディスクが同一計算機上, ファイル管理スレッドのみ異なる計算機上に存在する. この場合, ファイル管理スレッドからワーカスレッドへ制御が移行すると, ワーカスレッドの動作する計算機内で処理が完結する. このことから, 最も効率化が図られると考えられる.

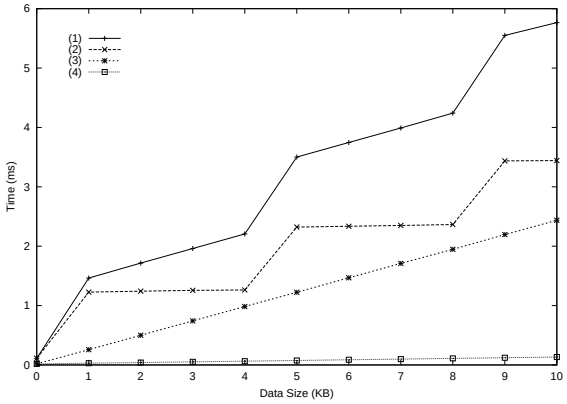

図 9　従来の手法での所要時間

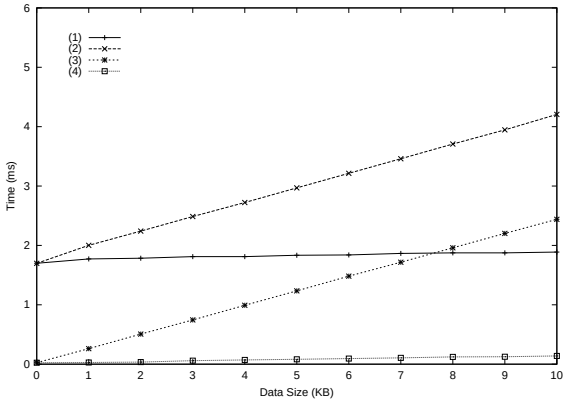

図 10　新たな手法での所要時間

(2) ファイル管理スレッドとディスクが同一計算機上, ユーザスレッドのみ異なる計算機上に存在する. (1) の場合と異なり, ワーカスレッドが他の計算機のディスクにアクセスする. これは, 従来のファイル管理方式と新たに提案した方式の利点および欠点が明確に現れる場合であると考えられる.

また, これら 2 つの場合の処理時間の内訳を知る目安となる以下の 2 つの位置関係についても計測した.

(3) ユーザスレッドとファイル管理スレッドが同一計算機上, ディスクが異なる計算機上に存在する.

(4) ユーザスレッド, ファイル管理スレッド, ディスクがすべて同一計算機上に存在する.

以上 4 つについて, Solelc における従来のファイル管理手法を用いた場合と新たに提案した手法を用いた場合の処理の所要時間をそれぞれ図 9, 10 に示す. それぞれ読出しデータサイズ 1KB ごとに計測し, 所要時間をミリ秒単位で示している.

図 9 において, (1) および (2) の場合の所要時間が 4KB ごとに大きく増加するのは, read システムコールの処理後, ユーザスレッドが読み出したデータにアクセスすると, メモリページ転送が発生するためである. Solelc では 4KB 単位でメモリページを管理しているため, 処理時間としてこのような形で現れる. その他の場合については, read システムコールの処理が完了した時点で読み出したデータを持つメモリページがユーザスレッドの動作する計算機上に存在するため, メモリページ転送は発生しない.

(3) および (4) の場合, ユーザスレッドとファイル管理スレッドが同一計算機上に存在する. そのため, 効率化を図りワーカスレッドに制御を移行することが従来の手法と同様の処理を意味し, 手法の違いによる影響がほとんど現れない. (3) と (4) の差は, ネットワークを介したブロックデータ取得に要する時間である.

(1) の場合, 従来の手法では, ユーザスレッドとディスクが同一計算機上に存在するにも拘らず, ファイル管理スレッドが他の計算機でファイル操作を行うため, ネットワークを介したファイルデータの通信に時間を要していた. 新たに提案した手法では, ユーザスレッドおよびディスクの存在する計算機のワーカスレッドにファイルデータ入出力を行わせることで, この問題を解決している. 図 10 から, ワーカスレッドへの制御の移行に時間を要するが, その後は単一計算機内で処理が完結していることがわかる.

(2) の場合, 新たな手法では, ワーカスレッドへの制御の移行に時間を要するため, 従来の手法からの効率化は行われない. 図 9, 10 からも, 新たな手法が逆にオーバヘッドになり, 非効率な処理になっていることがわかる.

今回の評価は, 単一ユーザスレッドからの要求の処理についてのみ行った. Solelc における従来のファイル管理手法では, 要求の処理をすべてファイル管理スレッドが行うため, 複数の要求を同時に処理できない. 一方, 新たに提案した手法では, ワーカスレッドによる処理は, 複数の計算機で並列に行うことが可能である. 処理の並列・分散化による効率化, およびその評価は今後の課題である.

6 おわりに

本稿では, 分散 OS Solelc の概要と, Solelc の特徴を考慮したファイル管理の効率化手法について述べた. 本手法を用いてファイル管理を行うことで, プロセスからのファイル操作要求の処理において, ファイルデータの流れの効率化を実現した. また, 処理の並列・分散化による応答性能の向上も期待できる.

しかしながら, 処理を分散させることにより, 制御の移行の際に通信を伴うため, プロセスがサイズの小さなファイルの読出しを要求するときなど, 逆に非効率となる場合も考えられる. 今後は, プロセスによるファイルの利用形態を考慮した効率化について検討を進める予定である.

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, 情報処理学会研究報告 2000-OS-84, pp.237–244 (2000).

[2] 水口孝夫, 芝公仁, 大久保英嗣: 分散オペレーティングシステム Solelc におけるファイルシステムの構築, 情報処理学会研究報告 2000-OS-84, pp.245–252 (2000).