Vol.2017-OS-141 No.2
2017/7/26

AnT の一括依頼並列処理機能

村岡 勇希[1]　佐藤 将也[1]　山内 利宏[1]　谷口 秀夫[1]

概要：マイクロカーネル OS では，OS 機能の一部を OS サーバとして実現している．このため，OS サーバを複数のプロセッサへ分散して実行することで，並列実行可能である．AP プロセスから OS サーバへの処理依頼は，高い利便性と低処理オーバヘッドを生かし，完了型のインタフェースを利用する．したがって，AP プロセスが一度に依頼できる処理は 1 つである．このため，AP プロセスが依頼する処理は，並列実行可能な処理であっても並列実行できない．そこで，AP プロセスが一度に複数の処理を OS サーバに依頼できれば，処理を並列実行できる．本稿では，マイクロカーネル構造を持つ ***AnT*** オペレーティングシステムにおいて，AP プロセスが一度に複数の処理依頼を行える一括依頼並列処理機能を提案し，評価結果を報告する．

1. はじめに

計算機性能の向上に伴い，提供されるサービスは高度化し，多様化している．また，サービスの提供を支えるシステムは複雑化している．このため，多様なサービスを支える高い適応性と複雑化したシステムによる不具合に耐える高い堅牢性を有するオペレーティングシステム（以降，OS）が求められている．高い適応性と堅牢性を有する OS のプログラム構造としてマイクロカーネル構造 [1][2] がある．

マイクロカーネル OS は，OS 機能をカーネルと OS サーバで実現している．AP プロセスが OS 機能を利用する際には，サーバプログラム間通信を利用した処理依頼を行う必要がある．マイクロカーネル OS では，OS サーバをプロセッサ毎に分散することで，OS 処理を分散できる．

マイクロカーネル構造を持つ ***AnT*** オペレーティングシステム（An operating system with adaptability and toughness）（以降，***AnT***）では，マルチコア環境で複数の OS サーバを同時起動し，OS 処理を分散して並列実行することにより高スループットを実現する OS 処理分散機能 [3][4] を有する．また，OS サーバは，非完了型のインタフェースで処理を別の OS サーバに依頼できるため，複数の処理を並列実行できる．これに対し，AP プロセスから OS サーバへの処理依頼は，高い利便性と低処理オーバヘッドを生かし，完了型のインタフェースを利用する．したがって，AP プロセスが一度に依頼できる処理は 1 つである．そこで，AP プロセスが一度に複数の処理を OS サーバへ依頼できれば，依頼した処理を並列実行できる [5][6].

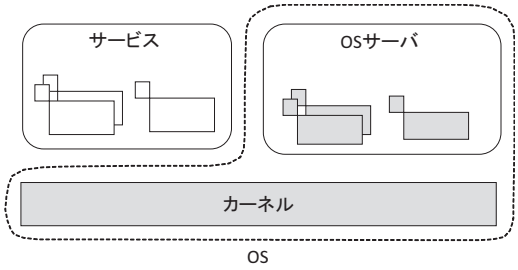


図 1　***AnT*** の基本構造

本稿では，***AnT*** において，AP プロセスが一度に複数の処理依頼を行える一括依頼並列処理機能を提案する．また，基本性能と並列処理性能の評価結果を述べる．

2. *AnT* オペレーティングシステム

2.1 基本構造

AnT は，マイクロカーネル構造を持つ OS である．プログラムは，OS とサービスからなる．OS は，カーネルと OS サーバからなる．サービスは，AP プロセスからなる．カーネルは，システムの動作を保証するための最小限の機能を有する．例えば，メモリ管理機能を有する．OS サーバは，OS 機能をプロセス化した部分である．例えば，ファイル管理機能を有する．

2.2 複写レスデータ授受機能

AnT は，プロセス間の通信を高速化するため，コア間通信データ域（ICA: Inter-core Communication Area）を利用した複写レスデータ授受機能を持つ．プロセス間の複

[1] 岡山大学 大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University

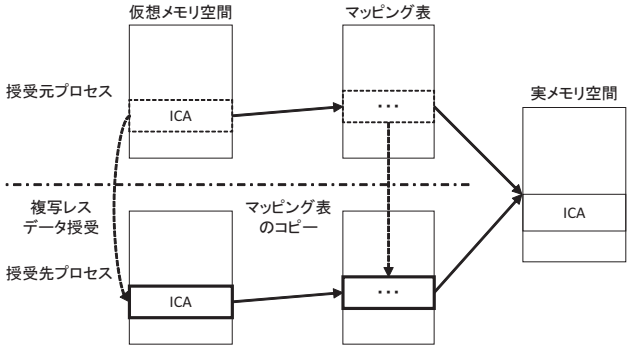


図 2 複写レスデータ授受

写レスデータ授受の様子を**図 2** に示す．ICA の特徴として，以下の 3 つがある．

（1）ページを単位とし，n ページ分の領域の確保と解放
（2）確保した領域（n ページ）の実メモリ連続の保証
（3）２仮想空間の間での領域の貼り替え

ICA は，ページを単位として管理される領域であり，ICA へのアクセスは，プロセス毎の仮想空間のマッピング表を通して行なわれる．ここで，マッピング表への書き込みを貼り付けと呼び，マッピング表からの削除を剥がしと呼ぶ．ICA を利用したプロセス間でのデータ授受は，授受するデータを格納した ICA をデータ授受元プロセスの仮想空間から剥がし，データ授受先プロセスの仮想空間へ貼り付けることで行なわれる．これらの操作をまとめて ICA の貼り替えと呼ぶ．

2.3 サーバプログラム間通信機構

サーバプログラム間通信 [7] の基本機構を**図 3** に示す．この機構は，ICA を利用することにより，プロセス間でデータ複写レスでの通信を実現している．具体的には，OS サーバへ渡す引数や通信制御の情報（以降，依頼情報）を制御用の ICA（以降，制御用 ICA）に格納し，扱うデータをデータ用の ICA（以降，データ用 ICA）に格納する．カーネルは，プロセス毎に通信のための依頼キューと結果キューを持つ．サーバプログラム間通信の流れを以下に述べる．

（1）依頼元プロセスが処理依頼を行うと，カーネルは，依頼先プロセスの依頼キューに依頼情報を格納した制御用 ICA を登録し，依頼先プロセスへ制御用 ICA を貼り替える．
（2）依頼先プロセスは，依頼キューから依頼情報を格納した制御用 ICA を取得し処理を実行する．
（3）依頼先プロセスは，依頼元プロセスの結果キューに結果情報を格納した制御用 ICA を登録し，依頼元プロセスへ制御用 ICA を貼り替える．
（4）依頼元プロセスは，結果キューから結果情報を格納した制御用 ICA を取得し処理を終了する．

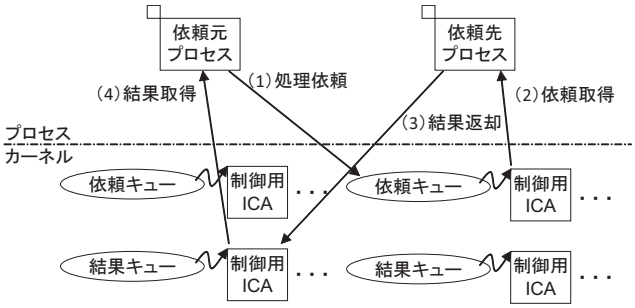


図 3 サーバプログラム間通信の基本機構

3. 一括依頼並列処理機能

3.1 AP 依頼処理の並列処理化

AP プロセスから OS サーバへの処理依頼は，高い利便性と低処理オーバヘッドを生かし，完了型のインタフェースを利用する．したがって，AP プロセスが一度に依頼できる処理は 1 つである．これに対し，OS サーバは，依頼された処理を非完了型のインタフェースで実行できる．

そこで，AP プロセスが一度に複数の処理を依頼できる機能を実現する．これにより，OS サーバは，複数の処理依頼を受け取ることができ，非完了型のインタフェースを利用して複数の処理を並列実行できる．

3.2 基本機構

一括依頼並列処理機能とは，複数の処理依頼を一度に行える機能である．これにより，AP プロセスが依頼した処理の処理時間を短縮できる．本機能の様子を**図 4** に示す．AP プロセスは，3 つの処理を依頼する場合，制御用 ICA を 3 個確保する．次に，3 つの処理の依頼情報を制御用 ICA1，2，および 3 へそれぞれ格納し，OS サーバ A へ一度に処理依頼を行う．OS サーバ A は，これらの制御用 ICA を非完了型インタフェースで OS サーバ P，Q，および R へ処理依頼する．OS サーバ A は，非完了型インタフェースで処理依頼を行うため，AP プロセスが依頼した処理は，並列実行される．

3.3 制御機構

本機能を実現する制御機構は，既存のサーバプログラム間通信機構への変更を最小化かつ局所化して実現する．

まず，既存のサーバプログラム間通信機構を利用して，AP プロセスが一度に 1 つの処理を依頼する処理流れを**図 5**（A）に示し，以下に説明する．

（1）AP プロセスは，処理依頼を行うために制御用 ICA を確保する．
（2）AP プロセスは，確保した制御用 ICA に各処理の依頼情報を格納する．
（3）AP プロセスは，処理依頼のシステムコールを発行

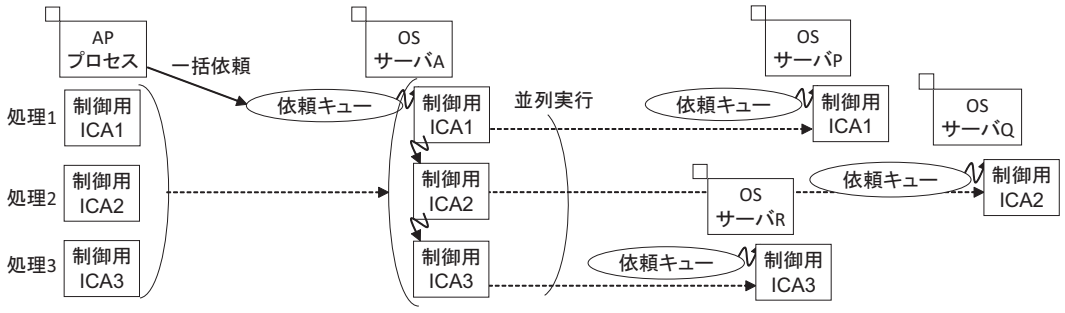


図 4 一括依頼並列処理機能

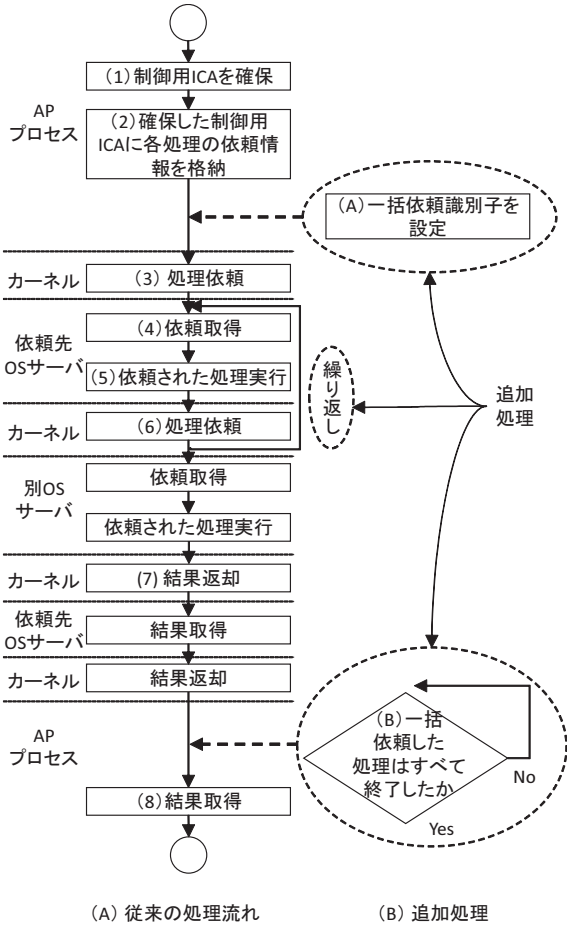


図 5 一括依頼並列処理機能の処理流れ

し，カーネルに処理が切り替わる．カーネルは，依頼先 OS サーバの依頼キューに制御用 ICA を登録する．

（4）依頼先 OS サーバは，AP プロセスから依頼された処理を取得するため，自身の依頼キューから制御用 ICA を取得する．

（5）依頼先 OS サーバは，依頼された処理を実行する．

（6）依頼先 OS サーバは，依頼された処理を実行するために別の OS サーバへ処理依頼を行う．

（7）別 OS サーバは，処理終了後に依頼先 OS サーバへ結果返却を行う．依頼先 OS サーバは，結果を取得し，AP プロセスへ結果返却を行うために，AP プロセスの結果キューに制御用 ICA を登録する．

表 1 インタフェースの機能と形式

機能	形式	戻り値
一括依頼	ncallsync(n, p); n: 処理依頼する制御用 ICA の個数 p: 制御用 ICA リストの先頭の制御用 ICA のアドレス	成功: 0 失敗: -1

（8）AP プロセスは，処理の結果を取得する．

なお，既存のサーバプログラム間通信機構では，（2）から（8）までの処理を繰り返し，複数の処理を実行する．

上記の処理にいくつかの新しい処理を追加することで一括依頼並列処理機能を実現する．

ここで，複数の処理結果を取得する方法として，複数の処理結果を一括して 1 回で取得する方法（一括結果取得）と結果取得を個別に複数回取得する方法（個別結果取得）がある．結果取得の回数が増えるとオーバヘッドが大きくなるため，一括結果取得を実現する．

本機能の処理流れを図 5 に示す．本機能を実現するために，1 つのプロセスが一括依頼した処理依頼の数を管理し，制御する一括依頼識別子を導入する．この識別子は，制御用 ICA に設定する．この識別子の値は，依頼した処理の数 (つまり，制御用 ICA の数) を示す．この識別子を利用した処理は，処理 (A) と (B) である．具体的には，処理 (A) は，制御用 ICA の一括依頼識別子に値を設定する．処理 (B) は，一括結果取得の処理である．具体的には，各処理の結果返却が行われた際に返却された結果の数が一括依頼識別子の値と等しくなった時に結果をまとめて取得する．

処理 (A) は，依頼元プロセスが行う処理として追加する．処理 (B) は，処理依頼における結果取得処理を改変するため，カーネルが行う処理として追加する．このため，本機能を実現する制御機構は，OS サーバへの改変無しに実現できる．

3.4 利用インタフェース

利用インタフェースの機能と形式を**表 1** に示す．依頼元プロセスは，一括依頼を行う制御用 ICA の個数（n）とキュー構造にした制御用 ICA のリストの先頭に登録されている制御用 ICA のアドレス（p）を引数として ncallsync()

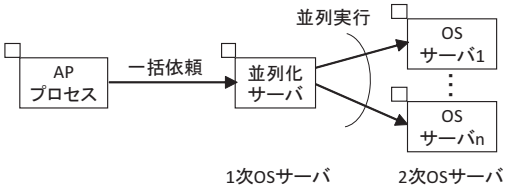


図 6 並列化サーバ

を発行する．これにより，依頼先の OS サーバの依頼キューへ複数個の制御用 ICA が一度に登録される．依頼した処理がすべて実行終了した後，各処理の結果情報が格納された制御用 ICA のリストを依頼元プロセスへ返す．

3.5 並列化サーバの導入

一括依頼並列処理機能は，一括依頼を受け取った OS サーバ（1 次 OS サーバ）が非完了型のインタフェースを駆使して，複数存在する次の OS サーバ（2 次 OS サーバ）に依頼することで並列処理の効果を発揮する．一方，OS サーバは，ファイル操作機能や通信制御機能といった OS 機能の一部を実現している．

そこで，これらの異なる機能を実現している OS サーバ群でも一括依頼並列処理機能を利用できるために，並列化サーバを導入する．この様子を**図 6** に示す．並列化サーバを 1 次 OS サーバとし，ファイル操作機能や通信制御機能といった OS 機能の一部を実現している OS サーバを 2 次 OS サーバに位置付ける．

4. 評価

4.1 観点と評価環境

評価は 2 つの観点で行う．一つは，基本評価として，処理依頼と結果返却における処理オーバヘッドを明らかにする．具体的には，1 個の処理依頼と結果返却を繰り返す場合（callsync()×n），ncallsync() により n 個の処理依頼と結果返却を行う場合，および並列化サーバを利用した場合について，1 コアで実行される場合の処理オーバヘッドを明らかにする．さらに，並列化サーバを利用した場合については，3 コアを利用して処理の並列化を高めた効果も比較する．もう一つは，並列処理性能評価として，CPU 処理を 2 並列で行える場合，およびファイル読み込み処理とデータ送信処理を並列に行えるファイル転送処理の場合について，提案機能の効果を明らかにする．

4.2 基本評価

ncallsync() と callsync() について，依頼する処理の数による処理依頼と結果取得にかかる処理時間の違いを明らかにする．具体的には，AP プロセスが OS サーバへ処理依頼を行い，OS サーバから AP プロセスへ結果返却が行われるまでの処理時間を測定する．この評価の処理流れを**図 7** に示す．

表 2 評価環境

OS	送信側：***AnT***，受信側：FreeBSD 6.3-RELEASE
CPU	Intel Core i7-2600 3.4 GHz 4 コア
メモリ	8,192 MB
SSD	Intel(R)Solid State Drive, 540s Series, 120GB
NIC	Intel PRO/1000 MT Desktop Adapter

複数の処理を依頼するとき，ncallsync() と callsync() では，処理依頼の実行回数（つまり，システムコール発行回数）が異なる．ncallsync() は，複数の処理依頼を一度に行えるため，依頼する処理の数に関わらず，システムコール発行回数は 1 回である．一方，callsync() は，一度の処理依頼で 1 つの処理を依頼するため，システムコール発行回数は，依頼する処理の数と同じである．

また，確保または解放する制御用 ICA の数について，ncallsync() は複数の処理依頼を一度に行うため，複数の制御用 ICA を確保または解放する必要がある．callsync() は，一度に 1 つの処理依頼を行うため，1 個の制御用 ICA を確保し，この制御用 ICA を使いまわす．

また，3.5 節で述べた並列化サーバから n 個の OS サーバへ非完了型のインタフェースで処理依頼を行う場合についても測定した．依頼する処理の数は，1 から 10 個とした．（C）において，OS サーバの数は，10 個である．これは，AP プロセスが依頼する処理を多くの OS サーバで実行させるために，各 OS サーバへ依頼する処理の数を 1 個までとしたためである．使用するコア数は，(A) と (B) の場合，1 である．また，（C）では，OS サーバに処理を並列実行させるために，コア数を 1 または 3 とした．コア数 3 の場合は，コア 0 上で AP プロセスと並列化サーバが走行し，コア 1 とコア 2 上で OS サーバが 5 個ずつ走行する．

依頼する処理の数による処理依頼と結果取得にかかる処理時間について，測定結果を**図 8** に示す．図 8 から以下のことが分かる．

（1）依頼する処理の数が 2 以上のとき，callsync()×n の処理時間より ncallsync() の処理時間の方が短い．このため，2 個以上の処理を依頼するとき，ncallsync() の方が有効である．

（2）callsync()×n と ncallsync() 並列化サーバ有 コア数 1 の処理時間を比較すると，callsync()×n の方が処理時間が短い．並列化サーバ有の場合，並列化サーバの処理や OS サーバへの処理依頼が増えるため，処理時間が長くなっている．このため，並列化サーバ有の場合，callsync()×n より処理時間が長くなる．

また，それぞれの処理依頼にかかる時間を分析するため，依頼する処理の数が 10 個のときのそれぞれの処理時間を 8 個の部分処理に分けて測定した．各部分処理の処理時間を**図 9** に示す．図 9 から以下のことが分かる．

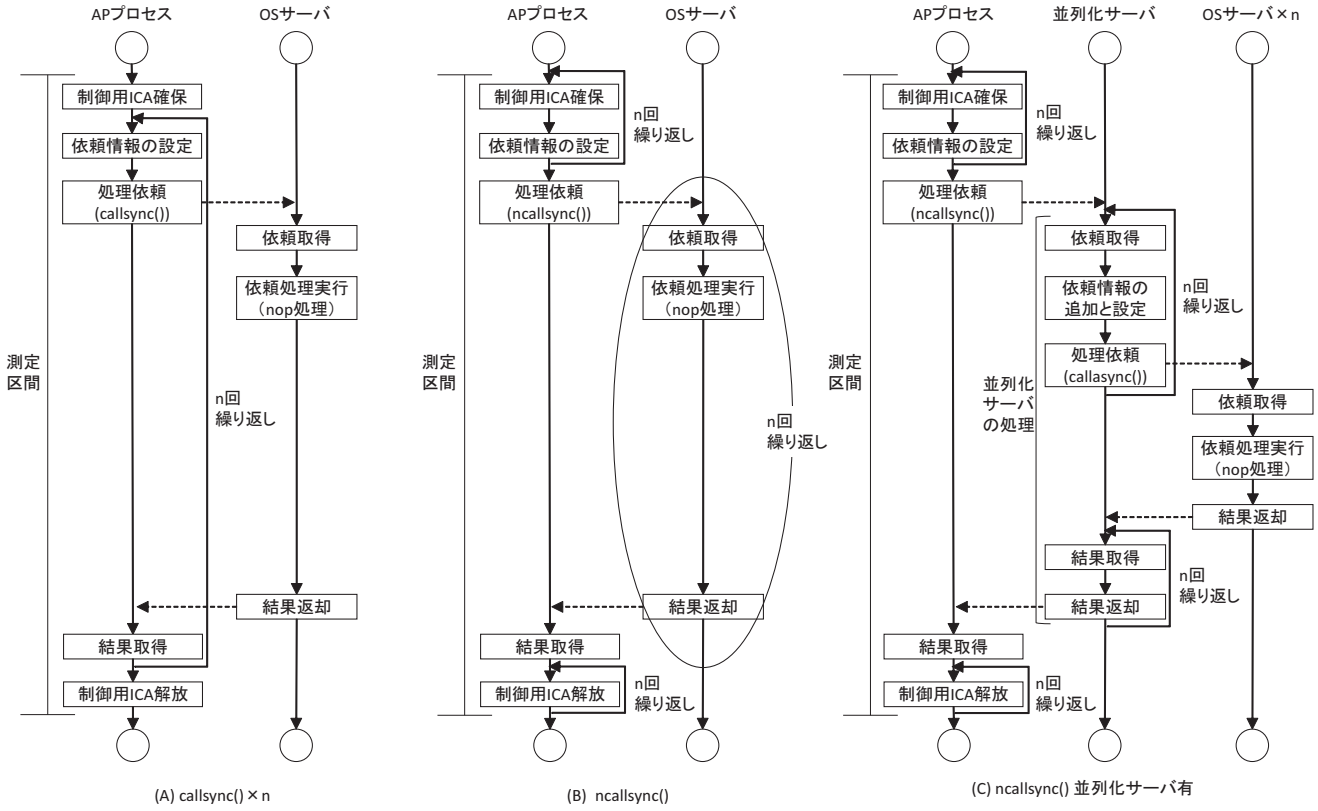


図 7 基本性能の評価の処理流れ

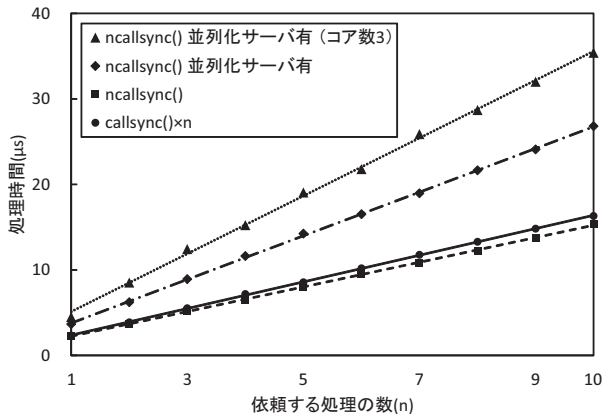


図 8 依頼する処理の数による基本性能の処理時間

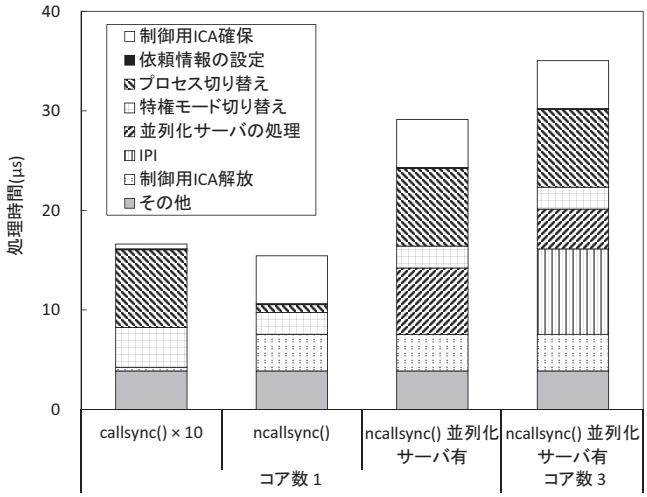


図 9 依頼する処理の数が 10 のときの各部分処理の処理時間

（3）callsync()×10 と ncallsync() の処理時間を比較すると，ncallsync() の方が短い．この差は，システムコール発行回数の違いにより生じている．システムコール発行回数は，callsync()×10 が 12 回，ncallsync() が 21 回であり，ncallsync() の方が多い．しかし，発行するシステムコールの内訳が異なっており，callsync()×10 は，制御用 ICA の確保 1 回，処理依頼 10 回，制御用 ICA 解放 1 回である．これに対して，ncallsync() は，制御用 ICA 確保 10 回，処理依頼 1 回，制御用 ICA 解放 10 回である．1 回あたりの制御用 ICA 確保と解放の合計処理時間より 1 回あたりの処理依頼の処理時間の方が長いため，ncallsync() の方が処理時間が短い．

（4）ncallsync() 並列化サーバ有のとき，コア数 1 の場合とコア数 3 の場合の処理時間を比較すると，コア数 1 の方が短い．この差は，IPI (Inter-Processer Interrupt) の有無と並列化サーバの処理時間の差により生じている．IPI は，処理依頼や結果返却の際に別コアのプロセスを起床させるために必要な処理であるため，コア数 3 の場合のみ実行される処理である．並列化サーバの処理時間の差について，コア数に関わらず，並列化サーバ自身が行う処理の処理時間は，同じである．差が生まれるのは，並列化サーバが処

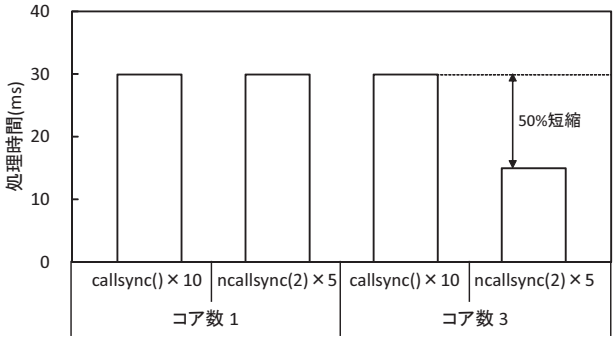


図 10 CPU 処理の処理時間

理依頼を行う OS サーバの処理時間である．コア数 1 のとき，10 個の OS サーバは，依頼された順に逐次的に処理を実行する．コア数 3 のとき，OS サーバは，2 つのコアに分かれて走行しているため，OS サーバの処理を並列実行できる．このため，並列化サーバの処理時間は，コア数 3 の方が短い．しかし，IPI の処理時間が OS サーバの処理時間より長いため，コア数 1 の方が処理時間が短い．

4.3 CPU 処理の並列処理性能

CPU 処理の並列処理性能を明らかにする．この評価では，CPU 処理を 3 ms 実行する OS サーバを 2 つ用意し，ncallsync() と callsync() をそれぞれ利用して処理依頼を行う場合の処理時間を測定した．この評価の処理流れは，OS サーバの依頼処理実行（nop 処理）を 3 ms の CPU 処理に置き換えたものである．

各 OS サーバへ，それぞれ 5 回処理依頼を行う．callsync()×n は，合計 10 回処理依頼を行う（callsync()×10）．ncallsync() は，それぞれの OS サーバへの 2 つの処理依頼を一度に行い，これを 5 回行う（ncallsync(2)×5）．

CPU 処理の処理時間について，測定結果を**図 10** に示す．図 10 から以下のことが分かる．

（1）コア数 1 のとき，callsync()×10 の場合と ncallsync(2)×5 の場合は，どちらの場合も処理時間がほぼ同じである．callsync()×10 は，処理依頼を逐次実行するため，3 ms × 10 = 30 ms に近い値となった．ncallsync(2)×5 は，AP プロセスが一度の処理依頼で 2 つの処理を依頼するため，並列化サーバは，処理依頼を並列実行できる．しかし，OS サーバが同一コア上で走行しているため，CPU 処理を並列実行できず，逐次実行する場合とほぼ同じ処理時間になった．callsync() と ncallsync() は，4.2 節で述べたオーバヘッドの差があるものの，このオーバヘッドは，CPU 処理の処理時間と比較して十分小さい．

（2）コア数 3 のとき，callsync()×10 の処理時間は，コア数 1 のときと比較して，IPI が発生するため，この分だけ長い．このオーバヘッドは，CPU 処理の処理時間と比較して十分小さい． ncallsync(2)×5 の場合の処理時間は，約 15 ms である．ncallsync(2)×5 は，一度の処理依頼で 2 つの処理を依頼する．OS サーバは，それぞれ異なるコアで走行しているため，CPU 処理を並列実行できた結果になった．このため，ncallsync(2)×5 は，2 つの処理を並列実行でき，callsync()×10 の場合やコア数 1 の場合と比較して，約 50%処理時間を短縮できた．

上記（1）(2) より，OS サーバが別コアで走行する場合，本機能により CPU 処理の並列実行を実現できることを示した．

4.4 I/O を伴う処理の並列処理性能

I/O を伴う処理の並列処理性能を評価するために，ファイル転送処理の処理性能を評価する．ファイル転送処理は，ファイル読み込み処理とデータ送信処理からなる．具体的には，外部記憶装置（DK）からファイルを読み込み，読み込んだデータを別の計算機へ送信する処理である．この処理では，ファイル読み込み処理とデータ送信処理を独立して実行できる場合，並列実行できる．

ファイル転送処理を行う AP プロセスは，ファイル操作処理と通信制御処理を処理依頼する．I/O 処理の発行回数や一度に処理するデータサイズによる影響を評価するために，I/O 処理の発行回数とデータサイズを変えて測定した．具体的には，ファイル読み込み処理では，1 または 2 KB のファイル読み込みを行う．データ送信処理では，1 KB のパケットを送信する．このとき，読み込んだファイルサイズが 1 KB の場合，1 回のファイル読み込み処理に対してデータ送信処理を 1 回実行する．2 KB の場合，1 回のファイル読み込み処理に対してデータ送信処理を 2 回実行する．ファイル転送処理を並列実行する効果を分かりやすくするために，1 回の測定でファイル転送処理を合計 51 回行った．これにより，4.2 節で述べたオーバヘッドによる影響を小さくできる．

本評価において，callsync() を利用する場合と ncallsync() を利用する場合について，ファイル転送処理の処理時間を測定する．それぞれを使用したファイル転送処理の処理流れを**図 11** に示す．ファイル操作処理に関する OS サーバは，3 種類である．ファイル管理サーバ（FS）は，i ノードを管理する．ブロック管理サーバ（BLK）は，ブロックキャッシュを管理する．ディスクドライバプロセス（DK Driver）は，DK を管理する．通信制御処理に関する OS サーバは，2 種類である．ネットワークプロトコル制御サーバ（INET）は，ソケット生成やパケット送信処理におけるプロトコルヘッダの設定を行う．NIC ドライバプロセス（NIC Driver）は，NIC を管理する．ここで，使用する OS サーバは，並列化サーバも含めて，すべて優先度は同じで最も高く設定している．

各 OS サーバが処理を行う順を**図 12** に示す．callsync() を利用する場合，AP プロセスは，FS と INET へそれぞれ

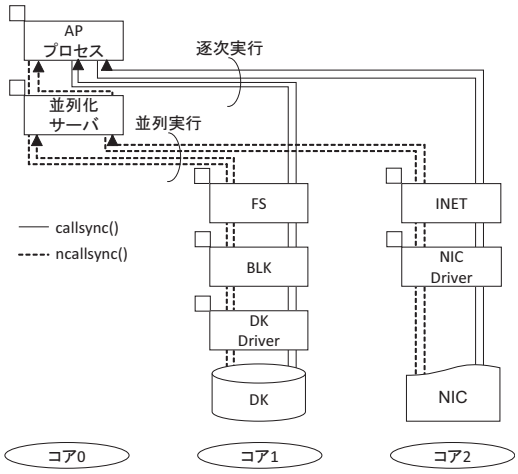
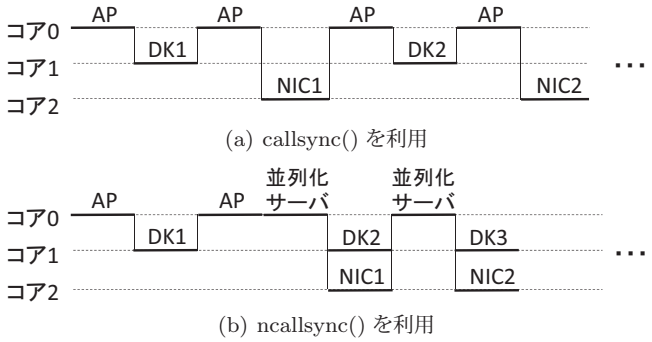


図 11 ファイル転送処理の処理流れ



(a) callsync() を利用

(b) ncallsync() を利用

図 12 ファイル転送処理の評価の処理流れ

逐次的に処理依頼を行うことでファイル転送処理を実行する．ncallsync() を利用する場合，AP プロセスは，並列化サーバへ一括依頼を行う．並列化サーバは，FS と INET への処理依頼を並列実行する．ここで，ファイル転送処理は，読み込んだデータを送信するため，ファイル読み込み処理を終えるまで，データ送信処理を行えない．このため，並列実行できない．並列実行するために，データ送信処理と次に送信するデータのファイル読み込み処理を一括依頼する．AP プロセスは，最初にファイル読み込み処理を FS へ処理依頼する．ファイル読み込み処理終了後，並列化サーバへ処理依頼し，読み込んだデータを渡す．並列化サーバは，データ送信処理依頼とファイル読み込み処理依頼を 50 回繰り返す．最後のファイル読み込み処理終了後，読み込んだデータを AP プロセスへ返却し，AP プロセスは，データ送信処理を INET へ処理依頼する（read()，(send()+read())×50，send()）．

使用するコア数は，1 または 3 である．コア数が 3 のとき，AP プロセスと並列化サーバがコア 0 上で走行する．ファイル操作処理に関する OS サーバはコア 1 上で走行し，通信制御処理に関する OS サーバはコア 2 上で走行する．

ファイル転送処理の処理時間について，測定結果を**図 13** に示す．図 13 から以下のことが分かる．

（1）コア数 1 のとき，callsync()×n の場合と ncallsync() の場合は，どちらも処理時間がほぼ同じである．callsync()×n の場合，ファイル読み込み処理と送信処理の処理依頼を逐次実行するためである．ncallsync() の場合，AP プロセスがデータ送信処理とファイル読み込み処理を並列化サーバへ一括依頼するため，並列化サーバは，処理依頼を並列実行できる．しかし，すべてのプロセスが同一コア上で走行するため，複数の CPU 処理を同時に並列実行できない．また，1 回のファイル読み込み処理時間に比べ，1 回のデータ送信処理時間は短い．このため，重複して実行できる処理が少なく，並列実行したことの効果が小さい．

（2）コア数 3 のとき，図 13（A）と（B）いずれの場合においても，ncallsync() の場合，処理時間は約 9.4 ms であり，callsync()×n の場合より処理時間が短い．これは，ファイル読み込み処理とデータ送信処理を並列実行できたためであると考えられる．

（3）（A）のコア数 3 のとき，callsync()×n に対する ncallsync() の処理時間短縮効果と（B）のコア数 3 のとき，callsync()×n に対しての ncallsync() の処理時間短縮効果を比較すると，（B）の方が効果が大きい．これは，ファイル読み込み処理 1 回に対してデータ送信処理を 2 回行うと，ファイル読み込み処理の処理時間とデータ送信処理の処理時間の差が小さくなり，並列実行により短縮できる処理時間の割合が増加したためである．具体的には，1 KB または 2 KB のファイル読み込み処理時間は，どちらも約 0.18 ms であり，1 KB のデータ送信処理時間は，約 0.02 ms である．このため，0.18 ms の処理と 0.04 ms(0.02×2) の処理を並列実行した．（A）では，逐次実行する場合と比較して処理時間を約 86%に短縮できたのに対し，（B）では，処理時間を約 78%に短縮できた．

上記（1）（2）（3）より，本機能が I/O を伴う処理の並列実行でき，全体の処理時間を短縮できることを明らかにした．ただし，この評価では，ファイル読み込み処理とデータ送信処理の処理時間の差が大きいため，並列実行の効果の確認が難しい．このため，INET サーバに 0.1 ms の CPU 処理を追加し，ファイル読み込み処理とデータ送信処理の処理時間の差が小さくなるように変更した．この変更を加えた場合のファイル転送処理時間について，測定結果を**図 14** に示す．図 14 から以下のことが分かる．

（4）コア数 3 のとき，ncallsync() の場合，短縮した処理時間の割合が大きくなった．例えば，（A）では，処理時間を約 60%に短縮できた．

上記（4）より，並列実行する処理の処理時間の差が小さいと，短縮できる処理時間の割合が増え，本機能の効果が大きくなることを明らかにした．

5. おわりに

マイクロカーネル構造を有する ***AnT*** において，複数の処理依頼を一度に行える一括依頼並列処理機能について提

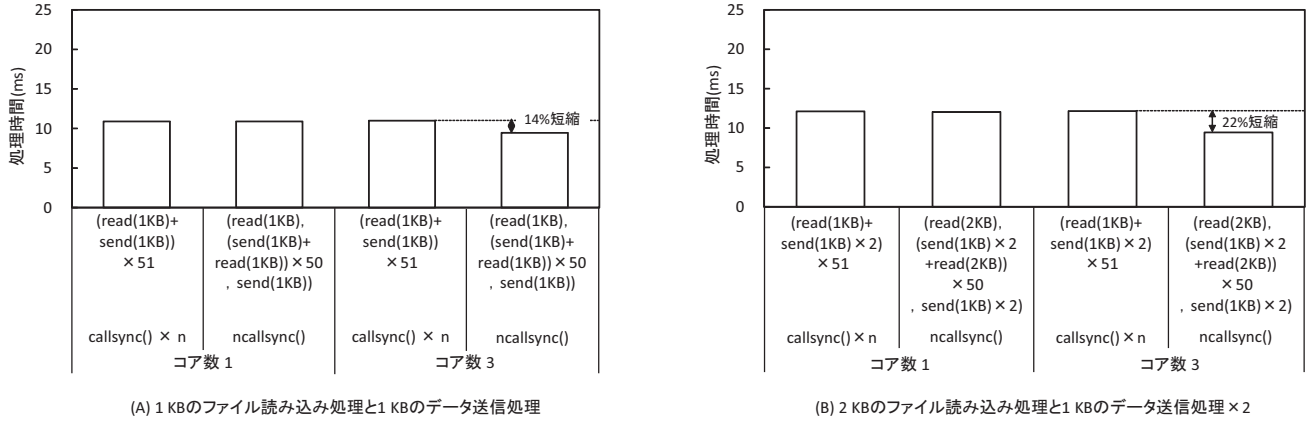


図 13 ファイル転送処理の処理時間

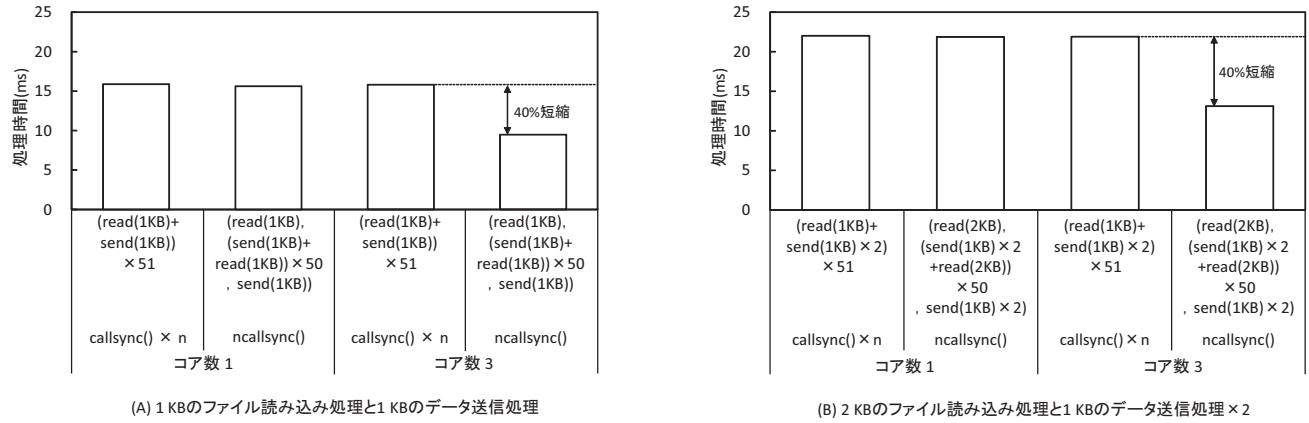


図 14 ファイル転送処理の処理時間（INET サーバに 0.1 ms の CPU 処理追加）

案し，評価結果を述べた．本機能の実現においては，従来のサーバプログラム間通信機構の変更を局所化することで，既存 OS サーバの変更無しで実現した．また，並列処理を促進するために，並列化サーバを導入した．

評価では，1 コア環境での制御オーバヘッドを明らかにした．制御オーバヘッドは，依頼する処理数に比例して増加する．しかし，制御オーバヘッドが最も大きい並列化サーバを導入した場合でも，依頼処理当たり 3 μs 程度である．コア数を増やして並列化サーバを別コアで走行させると，コア間通信のために，さらに 0.5 μs 程度増加する．また，OS サーバの処理が CPU 処理の場合，複数のコアを利用することで，並列化効果は非常に大きいことを示した．さらに，ファイル転送処理としてファイル読み込み処理とデータ送信処理を並列化した場合も，並列化効果を確認できた．ただし，1 回のファイル読み込み処理時間に比べ，1 回のデータ送信処理時間は短い（約 10 分の 1）ため，効果は大きくない．

参考文献

[1] Liedtke, J.: Toward real microkernels, *Communications of the ACM*, Vol.39, No.9, pp.70–77 (1996).
[2] Tanenbaum, A.S., Herder, J.N., and Bos, H.: Can we make operating systems reliable and secure?, Vol.39, No.5, pp.44–51 (2006).
[3] 佐古田 健志, 山内 利宏, 谷口 秀夫：高スループットを実現する OS 処理分散法の実現, マルチメディア, 分散, 協調とモバイル (DICOMO2013) シンポジウム論文集, Vol.2013, No.2, pp.1663–1670 (2013).
[4] 江原 寛人, 山内 利宏, 谷口 秀夫：ファイル操作に着目した OS 処理分散法, 情報処理学会研究報告, Vol.2015-OS-132, No.7, pp.1–7 (2015).
[5] Nomura A, Ishikawa Y, Maruyama N, Matsuoka S.: Design and Implementation of Portable and Efficient Non-blocking Collective Communication, *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pp.1–8 (2012).
[6] Seo, S., Latham, R., Zhang, J., and Balaji, P.: Implementation and Evaluation of MPI Nonblocking Collective I/O, *Proceedings of the 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pp.1084–1091 (2015).
[7] 岡本 幸大, 谷口 秀夫：***AnT*** オペレーティングシステムにおける高速なサーバプログラム間通信機構の実現と評価, 電子情報通信学会論文誌 (D), Vol.J93-D, No.10, pp.1977–1989 (2010).