

AnTの要求箱通信機能を用いた処理の多重化方式

寺本 大風¹ 佐藤 将也¹ 谷口 秀夫¹

概要:

AnT オペレーティングシステム (以降, **AnT**) は, マルチコアに対応したマイクロカーネル構造 OS であり, OS サーバを分散配置できる. また, AP プロセスが OS サーバの識別子を意識することなく処理を依頼できる要求箱通信機能を実現している. 本稿では, この機能を利用して, AP プロセスの 1 度の処理依頼で同じ処理を複数回実行する多重化方式を提案する. 具体的には, 多重度に合わせて影 OS サーバを複数用意し, 処理を実行する. これにより, AP プロセスに変更を加えることなく, 同じ処理を多重に実行できる.

1. はじめに

1 つのプロセッサ内に複数の命令実行部を有するマルチコアプロセッサが普及している. マルチコア環境では, プロセッサ処理を各コアに分散配置することで, プロセッサ処理の並列化を実現可能である. マイクロカーネル構造 [1][2][3] を有するオペレーティングシステム (以降, OS) では, OS 処理をプロセス (以降, OS サーバ) とカーネルで分担して実現している. このため, OS サーバを各コアに分散配置することで OS 処理を分散できる [4][5][6].

AnT (An operating system with adaptability and toughness) (以降, **AnT**) は, マイクロカーネル構造を有し, マルチコア環境に対応した OS [7] である. **AnT** は, OS サーバを別のコアに移譲する機能を持っており, これを用いてプロセスを各コアに分散することで, OS 処理の分散実行を実現している. また, AP プロセスが OS サーバの識別子を意識することなく処理を依頼することで, 複数の OS サーバに並列に処理依頼が可能な要求箱通信機能 [8] を実現している.

本稿では, 要求箱通信機能を利用して, 1 度の処理依頼で同じ処理を複数回実行する処理の多重化方式を提案し, 評価結果を報告する. 多重化方式を用いることで, 依頼プロセスに変更を加えることなく, 依頼プロセスからの依頼を複数回実行できる.

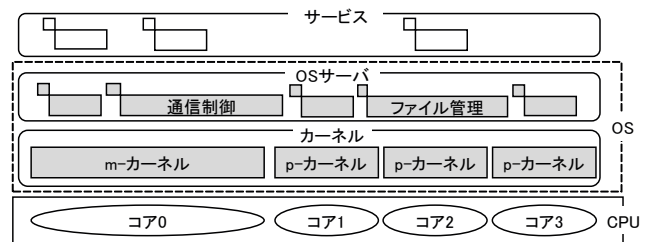


図 1 **AnT** の基本構造

2. **AnT** オペレーティングシステム

2.1 基本構造

AnT の基本構造を図 1 に示す。

AnT は各コア毎にカーネルを配置する構造となっており, マスタカーネル (以降, m-カーネル) とピコカーネル (以降, p-カーネル) の 2 種類のカーネルからなる. m-カーネルは, 全てのカーネル機能を有する. 一方で, p-カーネルは, プロセス実行制御機能, コア間通信制御機能, およびサーバプログラム間通信機能のみを有する. この構造によって, 各コアに個別にスケジューラを配置し, コア毎に独立なスケジュール管理を行う. また, 複数のコア上で複数のサーバプログラム間通信ができる.

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

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

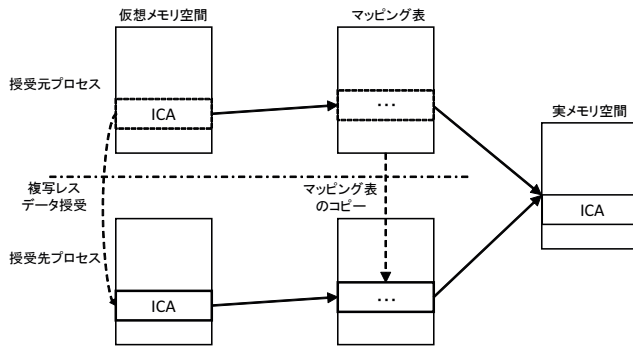


図 2 複製レスデータ授受

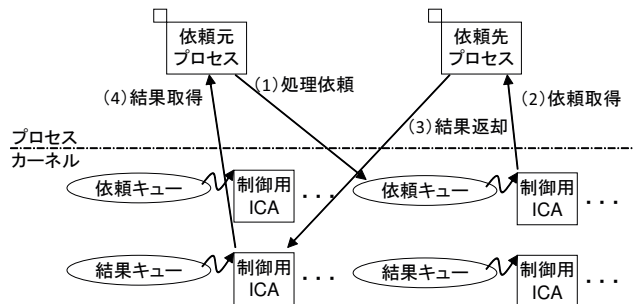


図 3 サーバプログラム間通信の基本機構

- (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、ページを最小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の基本機構 [9] 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製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

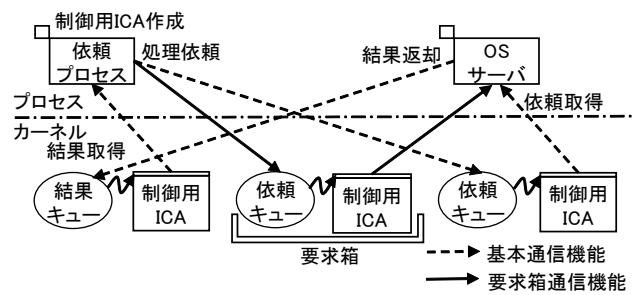


図 4 要求箱通信機能

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。

3. 要求箱通信機能 [8]

3.1 考え方

AnT のサーバプログラム間通信機構を用いて OS 処理の負荷分散を行う場合、以下の問題がある。

(問題) 同一の機能を持つ複数の OS サーバに対して、並列な処理依頼が不可能

そこで、この問題を解決するため、以下の 2 つの対処を行う。

(対処 1) OS サーバを指定しない要求箱による処理依頼の実現

(対処 2) 依頼プロセスからの起床要求を必要としないポーリングによる依頼取得の実現

以上の 2 つの対処により、問題の解決が可能になる。これらの対処を行った通信方式を要求箱通信方式と呼ぶ。

3.2 基本機構

要求箱通信機能とは、依頼元プロセスが依頼先の OS サーバを選択することなく、複数の OS サーバに対して並列に処理依頼を行うサーバプログラム間通信機能である。本機能の処理の様子を図 4 に示す。要求箱通信機能では、依頼元プロセスは依頼先 OS サーバの依頼キューではなく、要求箱に対して処理依頼を行う。OS サーバは、自身の依頼キューと要求箱の依頼キューを定期的に確認し、依頼が登録されている場合、依頼を取得し、OS サーバ処理を行う。

3.3 要求箱

要求箱は、OS サーバのファイルシステムや通信処理サーバなど、OS サーバの種類 1 つにつき 1 つずつ生成される。要求箱のコア ID は、紐づけられた OS サーバと同じ処理内容を表すコア ID を持つ。コア ID の構造を図 5 に示す。コア ID は、処理内容、処理種別、処理主体、通番、およ

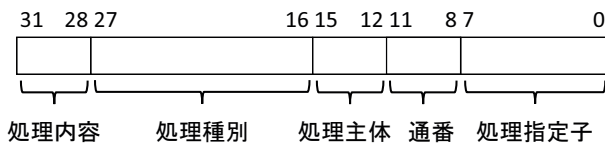


図 5 コア ID の構造

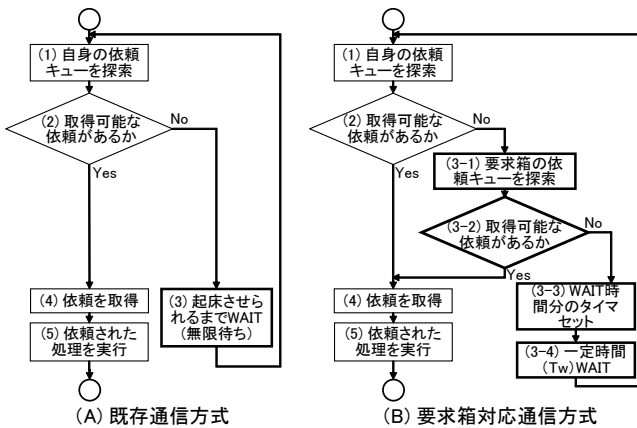


図 6 各方式における依頼取得処理の処理流れ

び処理指定子によって構成される。このうち、OS サーバの種類を処理種別、処理主体を用いて区別している。要求箱のコア ID は、OS サーバと同じ処理種別、処理主体を用い、通番によって区別する。要求箱の通番を 1 とし、要求箱から依頼を取得する OS サーバは 2, 3, 4 と順番に通番を割り当てられる。これにより、OS サーバは自身のコア ID から要求箱のコア ID を取得できる。

3.4 制御機構

既存の通信機能（以下、基本通信機能）と要求箱通信機能における依頼取得処理の処理流れを図 6 に示す。変更内容は、以下の 2 つである。

（変更 1）OS サーバの依頼取得の際、自身の依頼キュー探索に加え、要求箱の依頼キューを探索するように変更 (3-1)(3-2)

（変更 2）依頼取得待ちの WAIT 時間を無限待ちから一定時間 (T_w) 待ちに変更 (3-3)(3-4)

4. 要求箱通信機能を用いた処理の多重化方式

4.1 基本方式

処理の多重化方式とは、要求箱通信機能を用いて、依頼プロセスが 1 度の依頼で複数の OS サーバに処理依頼を行う通信方式である。この方式を用いることで、依頼プロセスに変更を加える必要なく OS サーバに同じ処理を複数回実行可能である。

処理の多重化方式では、要求箱通信機能で用いた要求箱に加えて、以下の OS サーバを用いる。

- (1) 影 OS サーバ（親）

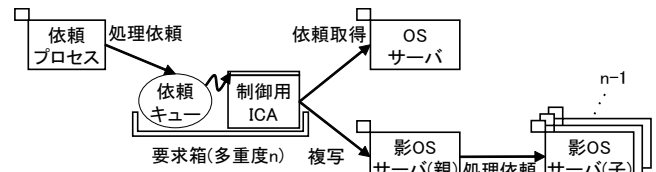


図 7 処理の多重化

影 OS サーバ（親）は、依頼プロセスが要求箱に依頼した内容から依頼情報を取得し、多重度に応じて影 OS サーバ（子）に処理依頼を行う OS サーバである。要求箱から依頼取得を行うため、要求箱と同じ処理種別をもつ。

- (2) 影 OS サーバ（子）

影 OS サーバ（子）は、多重処理を実行する OS サーバであり、OS サーバと同じ処理を行う。

この様子を図 7 に示し、処理の流れを以下に述べる。

- (1) 依頼プロセスは、要求箱に制御用 ICA を登録する。

(2) OS サーバは、要求箱から制御用 ICA を取得し、処理の実行と依頼プロセスへの結果返却を行う。

(3) 影 OS サーバ（親）は、要求箱から制御用 ICA の複製を取得する。その後、多重度に応じて、 n 個の影 OS サーバ（子）へ取得した制御用 ICA と同じ処理内容の処理依頼を行う。

(4) 影 OS サーバ（子）は、処理を実行し、影 OS サーバ（親）へ結果返却する。

(5) 影 OS サーバ（親）は、結果取得後、制御用 ICA を開放し、処理を終了する。

影 OS サーバ（親）が影 OS サーバ（子）からの結果を取得した際、影 OS サーバ（親）は依頼プロセスに対して結果返却を行わない。したがって、依頼プロセスは影 OS サーバによって多重化された処理を意識しない。

4.2 課題と対処

4.2.1 課題

処理の多重化方式を実現するためには、以下の課題がある。

（課題 1）複数の OS サーバが要求箱にある 1 つの制御用 ICA を取得する方法

本方式では、1 つの制御用 ICA に対して、複数の OS サーバが依頼取得と処理の実行を行う。しかし、要求箱に格納されている制御用 ICA は依頼情報を格納した領域への先頭アドレスであるため、複数の OS サーバがデータの格納や取り出しを行った場合、データの不整合が発生する。このため、既存の通信機能では複数の OS サーバが 1 つの制御用 ICA を取得し、処理を実行することはできない。

（課題 2）多重度の設定方法

多重度は、依頼プロセスと OS サーバは意識しないが、影 OS サーバ（親）は意識する必要がある。また、指定した多重度で処理を多重化できるようにするため、多重度は自

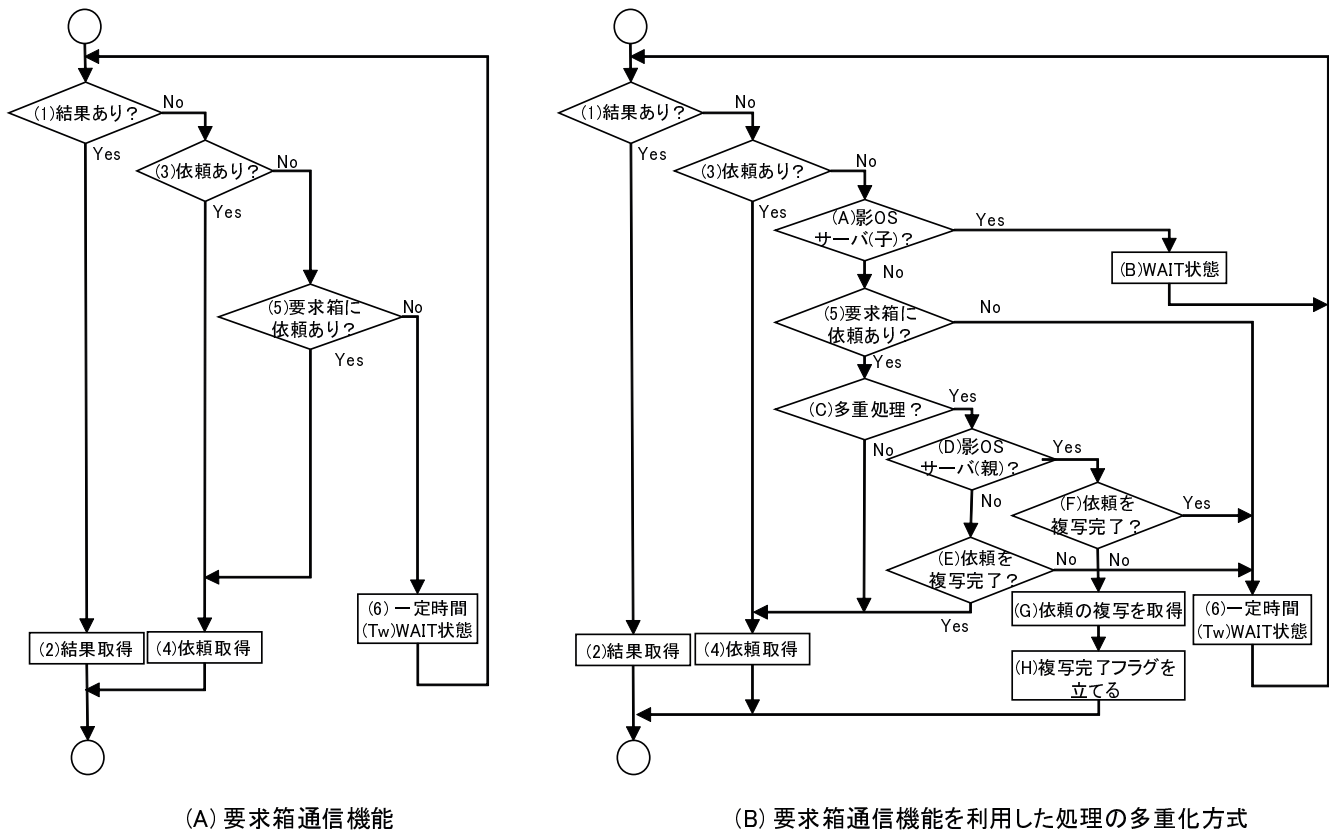


図 8 依頼取得処理流れの変更点

由に設定できるようにする必要がある。

4.2.2 対処

4.2.1 項の課題に対して、以下の対処を行う。

(対処 1) 影 OS サーバ (親) による制御用 ICA 情報の複写
制御用 ICA の複写を影 OS サーバ (親) が行い、OS サーバが影 OS サーバ (親) より先に制御用 ICA を取得しないように制御することで対処する。影 OS サーバは複写を行った後、複写元の制御用 ICA に対して複写完了済みフラグを立てる。OS サーバが要求箱からの依頼を試みた時に、複写完了済みフラグが立っていない場合、OS サーバは要求箱からの依頼取得を行わない。

(対処 2) 要求箱に多重度を保持させる

図 5 で示した、要求箱のコア ID の下位 6 ビットを多重度として扱う。これにより、多重度を影 OS サーバ (親) が認識でき、かつ影 OS サーバ (親) に関係なく自由に設定できる。

以上の対処を行うことで、OS サーバと影 OS サーバのそれぞれが依頼を取得することが可能になり、多重化方式が実現可能になる。

4.3 依頼取得の処理流れ

OS サーバの依頼取得の処理流れを図 8 に示す。要求箱通信機能における依頼取得処理を図 8 の (a) に、多重化方式における依頼取得の処理流れを (b) に示す。多重化

方式における依頼取得の処理流れは、要求箱通信機能の処理流れから、要求箱に関する部分の処理を変更する。変更箇所は (A) ~ (H) である。

処理 (A) と処理 (B) では、自身が影 OS サーバ (子) の場合は、要求箱の制御用 ICA を取得する必要がないため、要求箱から制御用 ICA を取得しないように制御している。処理 (C) では、多重処理を行わない場合は、既存の要求箱通信機能を利用できるように制御している。

また、影 OS サーバ (親) が要求箱にある制御用 ICA の複写を取得したことを確認するために、複写完了フラグを導入する。このフラグが立っていれば、影 OS サーバ (親) が要求箱にある制御用 ICA の複写を取得したことを示す。複写完了フラグを利用した処理は、処理 (D) ~ (H) である。自身が影 OS サーバでないなら、処理 (E) に遷移し、制御用 ICA に複写完了フラグが立っていれば依頼を取得し、立っていない場合は一定時間 WAIT 状態に遷移する。自身が影 OS サーバであるなら、処理 (F) に遷移し、要求箱の依頼キューに登録された制御用 ICA を複写し、一定時間の WAIT 状態に遷移する。影 OS サーバであるか否かの識別には、コア ID を用いる。複写完了フラグの導入により、OS サーバが要求箱にある制御用 ICA を取得するかの判定および影 OS サーバ (親) が要求箱にある制御用 ICA の複写を取得するかの判定が可能になる。

表 1 評価に利用した計算機

OS	AnT
CPU	Intel Core i7-2600 (3.4 GHz) 4 コア
メモリ	32GB
HDD	7200rpm, 160G, WD1600AAJS 2 個

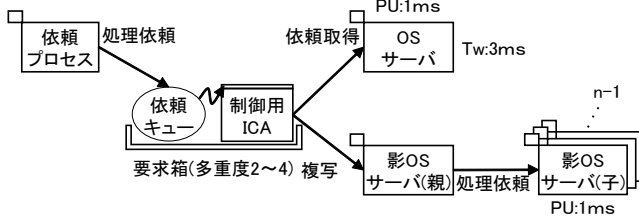


図 9 多重度による影響の評価

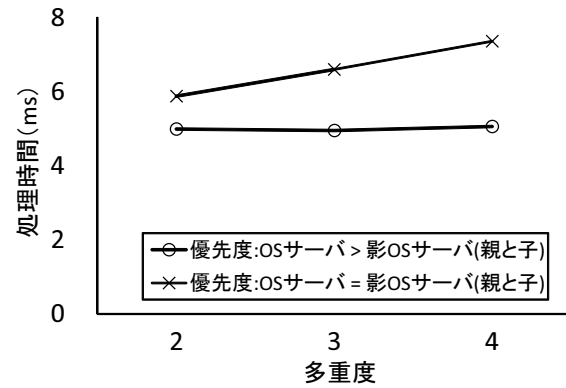


図 10 各多重度における処理時間

5. 評価

5.1 評価内容

評価に利用した計算機を表 1 に示す。基本評価として、1ms の PU 処理を行う OS サーバを用意し、多重度を変化させた場合の処理時間への影響を明らかにする。また、基本通信機能と多重化方式について、ファイル書き込み速度を比較する。

5.2 多重度の処理時間に与える影響

5.2.1 評価環境

依頼プロセスが要求箱へ処理依頼を行い結果取得するまでの処理時間を測定する。測定の様子を図 9 に示す。OS サーバは、1ms の PU 処理を行う。Tw は、3ms に設定した。測定結果は、100 回測定した処理時間の平均値を用いる。

5.2.2 結果と考察

評価結果を図 10 に示し、以下に考察を述べる。

(1) OS サーバと影 OS サーバ (親と子) の優先度が同じ場合、多重度が増加するにつれて、処理時間は長くなる。これは、OS サーバ処理と同時に影 OS サーバ (親と子) 処理が行われるためである。

(2) OS サーバが影 OS サーバ (親と子) より優先度が高い場合、多重度に関わらず、処理時間にほとんど変化はない。これは、OS サーバが影 OS サーバ (親と子) より優先して実行されるためである。

以上の評価より、多重度の増加は OS サーバの処理時間に影響を与えるものの、影 OS サーバの優先度を OS サーバよりも低くすることにより回避できるといえる。

5.3 ファイル書き込み処理時間

AnT において、ファイル操作に関連する OS サーバは、ファイル管理サーバ (FS)、ブロック管理サーバ (BLK)、およびディスクドライバサーバ (DK) の 3 つである。ファ

イル操作を要求するプロセス (AP) は PU 処理を実行し、FS と BLK を経由し、DK に対して指定したファイルへの書き出しを要求する。

基本通信機能との比較、および複数コアに処理を分散した場合の多重化方式の効果を明らかにするため、以下の 4 つの処理形態について測定する。各処理形態を図 11 に示す。

(処理形態 P) 基本通信を用いて、FS1 に処理依頼する。

(処理形態 Q) AP プロセス (コア 0) が、要求箱に処理依頼をし、それを FS1 (コア 1) のみが依頼取得する。

(処理形態 R) 多重化方式を用いて、2 つのディスクへの書き込み処理をコア 1 とコア 2 に分散する。

(処理形態 S) 多重化方式を用いて、2 つのディスクへの書き込み処理をコア 1 のみで行う。

5.3.1 基本通信との比較

測定結果を図 12 に示す。処理形態 (P), (Q), (R) の 3 つの処理形態について、4KB の書き込み単位で 30 回のデータの書き込みを行い、処理時間を評価する。また、AP プロセスの処理時間が多重化方式に与える影響を評価するために、AP プロセスが 0ms, 2ms, 4ms, 6ms の PU 処理を行う場合について、同様に評価を行った。(A) は、AP プロセスの PU 処理とファイル書き込み処理の処理時間を示し、(B) のグラフは (A) のグラフから AP プロセスの PU 処理時間を引いた値を示している。

(1) 図 (a') では、Tw = 0 の場合、処理形態 (P) は、処理形態 (Q), (R) よりも処理時間が 30ms 短い。これは、処理形態 (P) では、基本通信によって要求箱を介さずに処理依頼を行っているため、依頼先の OS サーバが WAIT 状態の場合でも、依頼先 OS サーバを起床させて依頼を取得させることができるためである。

(2) 図 (a') において、処理形態 (Q), (R) は、Tw の増加に伴って処理時間が約 4 倍に増加している。これは、Tw の設定時間の増加により、OS サーバが WAIT 状態に入る時間が長くなり、OS サーバ処理の実行が遅くなるためである。

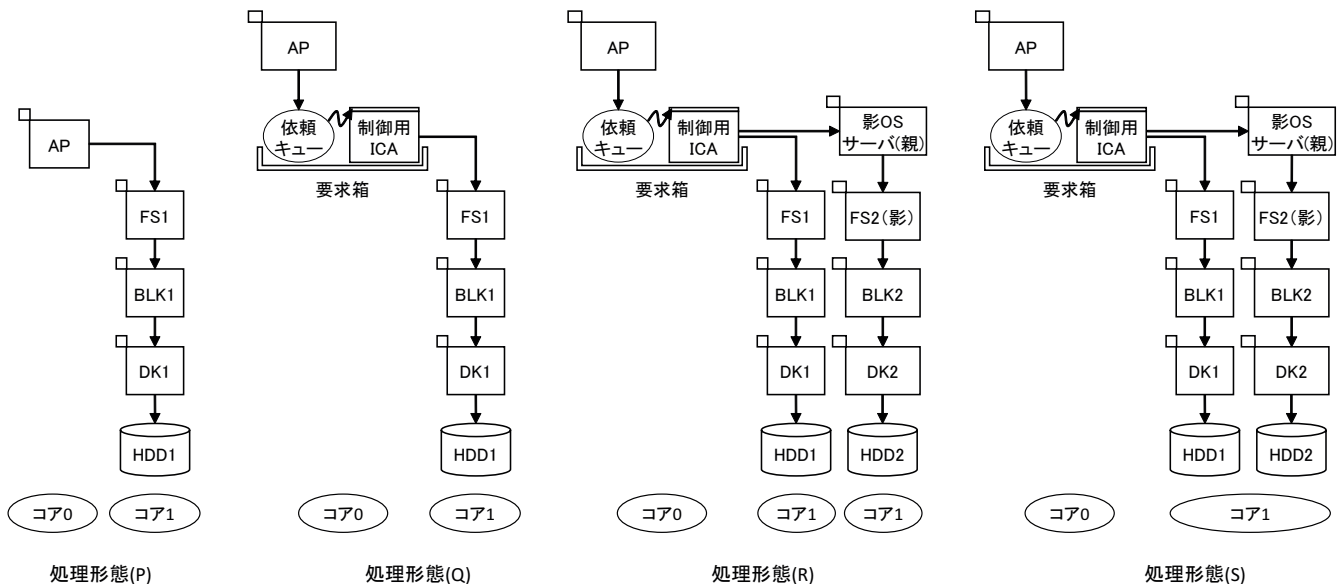


図 11 ファイル書き込み評価における処理形態

(3) 図 (a')-図 (d') より, AP プロセスの PU 処理時間が長大化するにつれて, 処理形態 (P) と処理形態 (R) の処理時間の差が減少していることが分かる. これは, T_w が AP プロセスの PU 処理時間によって相殺されるためである.

(4) 処理形態 (Q) と処理形態 (R) を比較すると, 処理形態 (S) では, 2つのディスクに対して処理依頼を行っているにも関わらず, 処理時間がほぼ同じである. これは, OS サーバを各コアに分散させたことで, ファイル書き込み処理を複数コアで並列に実行できたためである.

図 12 の (B) から, 処理形態 (P) と処理形態 (R) の処理時間の比を算出する. また, 処理時間の長い処理に対して, 多重処理方式の効果が高くなることを示すため, 8KB 単位での書き込みについて同様の評価を行い, 同様に処理形態 (P) と処理形態 (R) の処理時間の比を求めた. この結果を図 13 に示す.

図 13 から, 以下のことが分かる.

(1) $T_w = 1$, PU 処理時間=0ms の場合について, 書き込み単位が 4KB の場合の処理時間比が 3.03 であるのに対し, 8KB の場合は 2.40 と小さい. これは, 8KB 単位でのデータ書き出しが 4KB のデータ書き出しよりも時間がかかるため, T_w による待ち時間を相殺するためである.

(2) $T_w = 1$, PU 処理が 4ms の場合, 書き込み単位が 8KB の場合では, 処理時間比が 1.01 と, 1 に収束しているのに対し, 書き込み単位が 4KB の場合は 1.11 と収束しきっていない. これは, AP プロセスの PU 処理時間と 8KB のファイル書き出し時間の和が, 4KB 書き込みの場合よりも長いためである.

以上の評価より, 処理時間の長いプロセスに対しては, 多重処理機能が有効であることが分かる.

5.3.2 OS サーバの分散配置による処理時間への影響

処理形態 (R) と (S) を比較し, OS サーバを各コアに分散配置することによって多重化方式に与えられる影響を評価する. 測定内容は 5.3.2 項と同じく, 4KB 単位で 30 回のファイル書き込み処理の処理時間を比較する. 測定結果を図 14 に示す. 測定結果より, 以下のことがわかる.

(1) $T_w = 1\text{ms}$ の場合, 処理形態 (R) の方が処理形態 (S) よりも処理時間が 14ms 程度短い. これは, 処理形態 (R) では, OS サーバを分散することで, OS サーバ処理を並列に実行できるためである.

(2) $T_w = 3\text{ms}$, 5ms の場合, 処理形態 (R) と処理形態 (S) の処理時間はほぼ同じである. これは, OS サーバの WAIT 時間が長い場合, 影 OS サーバによって取得された依頼が WAIT 時間の間に実行されるためである. 以上より, 多重化方式において, T_w の値を小さく設定した場合に, OS サーバの分散が効果的であるといえる.

6. おわりに

マルチコアに対応したマイクロカーネル構造 OS である *AnT* において, 要求箱通信機能を利用した処理の多重化方式について提案し, 評価結果を述べた. 要求箱通信機能は, 依頼プロセスからの依頼を依頼先 OS サーバの依頼キューではなく要求箱の依頼キューに格納することにより, 複数の OS サーバに対して依頼プロセスが依頼先を意識することなく処理依頼が可能な機能である. 処理の多重化方式は, 要求箱通信機能を利用した通信方式である. 影 OS サーバ (親と子) を用いて, AP プロセスに変更を加えることなく処理依頼内容を複製することで, 1 度の処理依頼で多重に処理を実行する機能である.

評価では, 1ms の PU 処理を用いて, 多重化方式における,

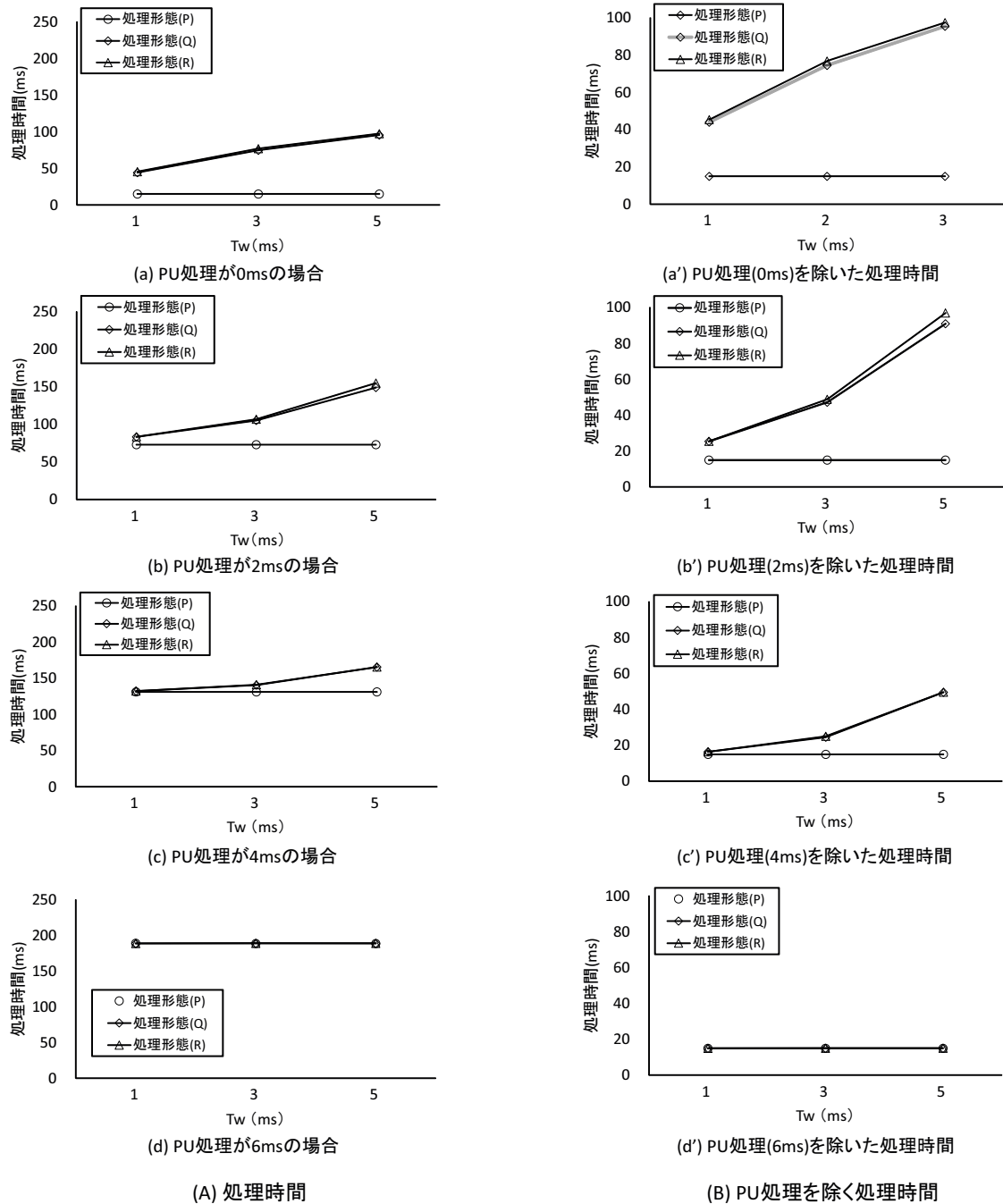
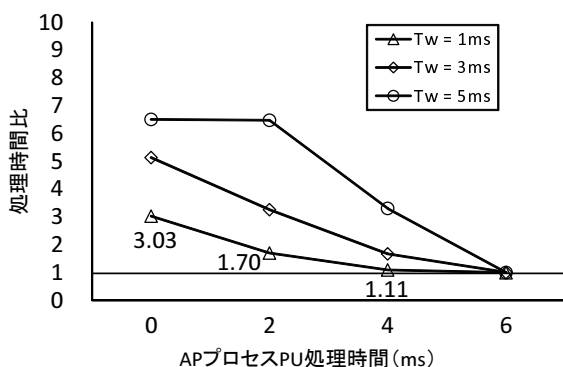


図 12 処理時間 (4KB 単位の書き出しを 30 回)

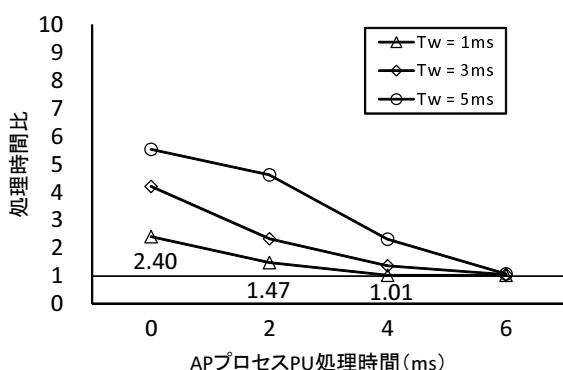
多重度が多重化方式に与える影響について評価した。ファイル書き込みの評価では, AP プロセスに 0ms, 2ms, 4ms, 6ms の PU 処理時間を追加し, 基本通信機能と多重化方式の処理時間を比較することで, 処理時間の長いプロセスほど多重化方式が有効に機能することを示した。また, ファイル操作を行う OS サーバを複製し, 各コアに分散配置させた場合の処理時間への影響について評価した。この評価では, T_w を小さく設定することで, OS サーバの分散による効果が出ることを示した。

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?, IEEE Computer Magazine, Vol.39, No.5, pp.44–51 (2006).
- [3] Heiser, G., Elphinstone, K.: L4 microkernels: The lessons from 20 years of research and deployment, ACM Transactions on Computer Systems (TOCS), Vol.39, No. 1, pp.1–29(2016).
- [4] Hamad, M., Schlatow, J., Prevelakis, V., Ernst, R. : A communication framework for distributed access control in microkernel-based systems, In 12th Annual Workshop on Operating Systems Platforms for Embedded Real-



(A) 書き込み単位が4KBの場合



(B) 書き込み単位が8KBの場合

図 13 処理時間 (処理形態 (R)/処理形態 (P))

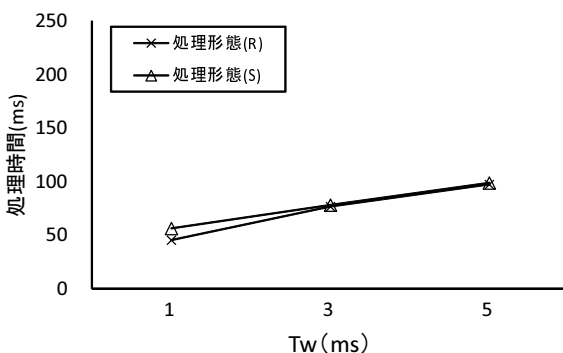


図 14 ファイル書き込みの処理時間 (書き込み単位: 4KB)

(2018).

- [9] 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).

Time Applications (OSPRT16) pp. 11–16(2016)

- [5] 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).
- [6] 江原 寛人, 栢田 圭祐, 山内 利宏, 谷口 秀夫: プロセス間通信を抑制しデータ共有するマイクロカーネル構造 OS 向けファイル操作機能の実現と評価電子情報通信学会論文誌 (D), Vol.J99-D, No.10, pp.1069–1079 (2016.10).
- [7] 井上 喜弘, 佐古田 健志, 谷口 秀夫: マルチコアプロセッサ上での負荷分散を可能にする *AnT* オペレーティングシステムの開発, 情報処理学会研究報告, Vol.2012-DPS-150, No.37, pp.1–8 (2012).
- [8] 寺本 大風, 佐藤 将也, 山内 利宏, 谷口 秀夫: OS サーバ処理の負荷分散を可能にする *AnT* の要求箱通信機能, 情報処理学会研究報告, Vol.2018-OS-144, No.6, pp.1–8