Vol.2018-OS-144 No.6
2018/7/30

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

寺本 大風[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配置して実行することで，OS の処理性能を向上できる．特に，同一の機能を持つ複数の OS サーバを分散配置することで，OS 処理の負荷を分散でき，処理スループットの向上が見込める．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，多くの AP プロセスから OS へ依頼された処理を負荷分散実行できる要求箱通信方式を提案し，評価結果を報告する．

1. はじめに

近年，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．マルチコア環境において，複数のプロセスが単一の OS サーバに対して処理依頼を行った場合，OS サーバの処理がボトルネックとなり，並列処理の効果が失われる場合がある．これを防ぐための方法として，負荷の集中する OS サーバを複数生成し，それらを他のコアに分散配置することにより，負荷を分散する方法がある [5][6][7]．しかし，OS サーバを各コアに分散配置したとしても，依頼元プロセスが意識して依頼先 OS サーバを選択する必要がある．

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[4] 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 処理の並列実行を実現している．

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，依頼元プロセスが OS サーバを意識しないで処理依頼することで負荷分散を可能にする要求箱通信方式を提案し，評価結果を報告する．

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

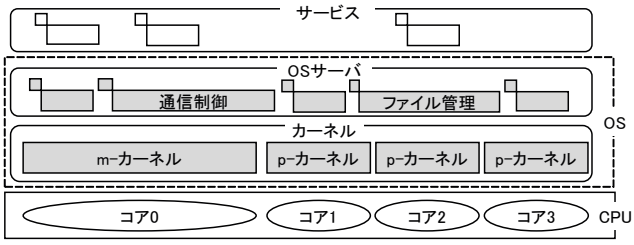

図 1　***AnT*** の基本構造

2. *AnT* オペレーティングシステム

2.1 基本構造

AnT は，マイクロカーネル構造を有している．***AnT*** の基本構造を**図 1** に示し，以下に説明する．マイクロカーネル構造 OS では，OS 機能をカーネルと OS サーバで分担して実現している．カーネルは，スケジュール処理などのプロセス実行制御機能を持つ．OS サーバは，OS 機能をプロセス化した部分であり，ファイル管理機能や通信ネットワーク制御機能を持つ．***AnT*** におけるカーネルは各コア毎にカーネルを配置する構造 [8] 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たコア毎に独立なスケジュール管理が可能となる．また，コア毎に並列なサーバプログラム間通信が可能になり，複数のコア上で複数のサーバプログラム間通信が可能となる．

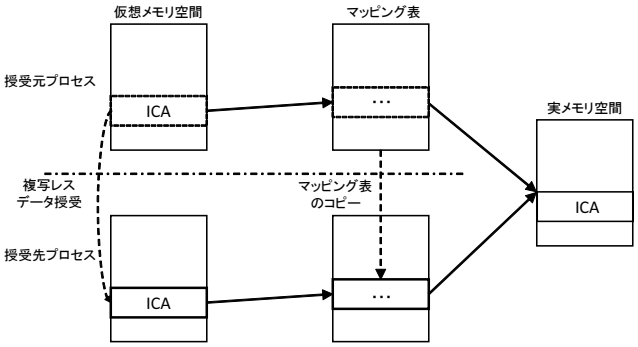

図 2 複写レスデータ授受

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
（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．

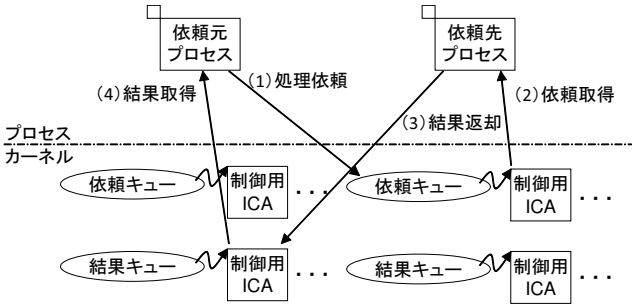

図 3 サーバプログラム間通信の基本機構

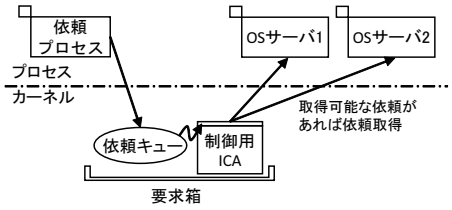

図 4 要求箱からの依頼取得

3. 要求箱通信機能

3.1 問題と対処

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

処理の集中する OS 処理の負荷分散を行うためには，同じ機能を持つ OS サーバを複数コア上に分散配置することが有効である．これにより，OS 処理を並列に実行することが可能になる．しかし，既存のサーバプログラム間通信機構では同じ OS 処理を並列に実行することができない．これは，依頼プロセスが依頼情報を制御用 ICA に登録する段階で，依頼先の OS サーバが 1 つに定められるためである．

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

要求箱とは，依頼プロセスからの依頼を格納するための領域である．依頼プロセスが要求箱の依頼キューに対して依頼を行う場合，依頼プロセスは OS サーバを指定することなく処理依頼が可能である．この様子を**図 4** に示す．依頼された処理に対応する各 OS サーバが，要求箱の依頼キューから依頼を取得することにより，複数の OS サーバによる並列な依頼取得が可能になる．

（対処 2）依頼プロセスからの起床要求を必要としない

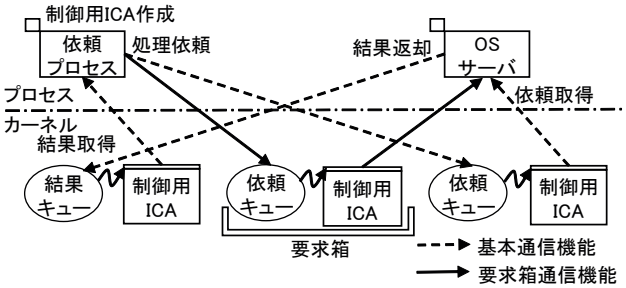

図 5 要求箱通信機能

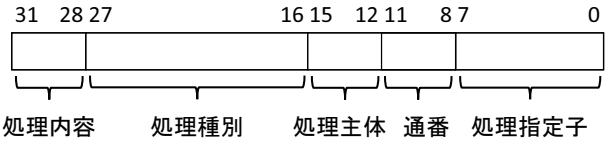

図 6 コア ID の構造

ポーリングによる依頼取得の実現

既存のサーバプログラム間通信機構では，依頼プロセスが OS サーバを指定することにより，OS サーバへの起床要求を可能にしている．しかし，（対処 1）を実現した場合，OS サーバを指定しないため，処理依頼時に OS サーバを起床できない．本方式では，OS サーバが一定間隔で依頼キューに対してポーリングを行うことで，依頼を取得する．これにより，依頼プロセスからの起床要求を必要としない依頼取得が可能になる．

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 5** 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 6** に示す．コア ID は，処理内容，処理種別，処理主体，通番，およ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．

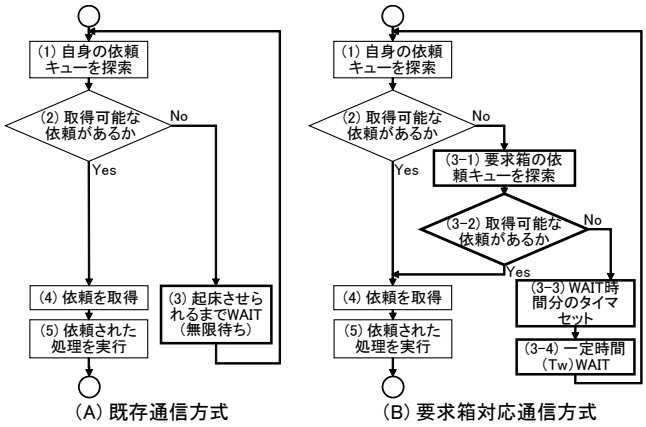

図 7 各方式における依頼取得処理の処理流れ

3.4 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．このため，サーバプログラム間通信機構の依頼取得処理のみに変更を加えることで実現する．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 7** 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)

基本通信機能では，OS サーバは依頼取得の際，自身の依頼キューを探索し，依頼が登録されていないかを確認する．要求箱通信方式では，自身の依頼キューを確認した結果，依頼が登録されていない場合，要求箱の依頼キューを探索する．

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

基本通信機能では，AP プロセスが依頼先 OS サーバに処理依頼を行ったとき，OS サーバが WAIT 状態の場合，OS サーバを起床させる．しかし，要求箱通信機能では，AP プロセスは依頼先 OS サーバを意識していないため，OS サーバを起床させることができない．このため，OS サーバは AP プロセスからの起床要求を待たず，定期的に起床して要求箱の依頼キューに依頼が登録されていないかを確認する必要がある．待ち時間を設定する際，OS サーバはタイマを設定する．OS サーバが一定時間待ちではなく依頼元プロセスからの起床要求によって起床させられた際は，起床させた AP プロセスによってタイマが削除される．

AnT では，OS サーバをコア ID によって識別している．各 OS サーバのスケジュールキューはサーバ情報表によっ

表 1　基本通信評価に利用した計算機

OS	***AnT***
CPU	Intel Xeon E5-2630 v3 (2.4 GHz) 8 コア
メモリ	32GB

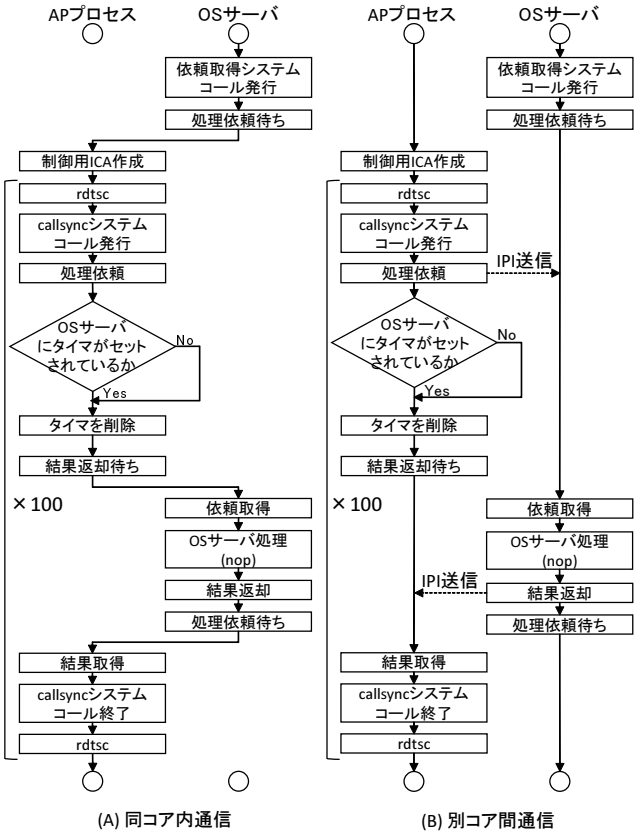

図 8　基本通信における処理流れ

て管理されており，コア ID を指定することで当該プロセスのサーバ情報表にアクセスできる．OS サーバは依頼取得の際，自身のコア ID を用いて自身の依頼キューを探索している．このため，要求箱の依頼キューから依頼を取得する場合，要求箱のコア ID が必要となる．3.3 節で述べた通り，要求箱のコア ID は依頼取得を行う OS サーバと同じ処理種別と処理主体を持っており，通番は 1 に定められている．これにより，OS サーバ自身のコア ID 情報から要求箱のコア ID を特定し，依頼を取得できる．

要求箱通信機能では自身の依頼キューと要求箱の依頼キューの両方に依頼が登録されている場合，自身の依頼キューから優先して依頼を取得する．これは，要求箱からの依頼取得が優先された場合，各 OS サーバが自身の依頼キューの状況に関わらず要求箱から依頼取得を行ってしまい，負荷分散ができないためである．

4. 評価

4.1 評価内容

基本性能の評価として，要求箱通信機能の実装によるオーバヘッドを明らかにする．また，OS サーバを各コアに分散配置させた際の効果を明らかにする．

通信処理性能の評価として，NIC ドライバを分散配置した際の分散効果を評価する．

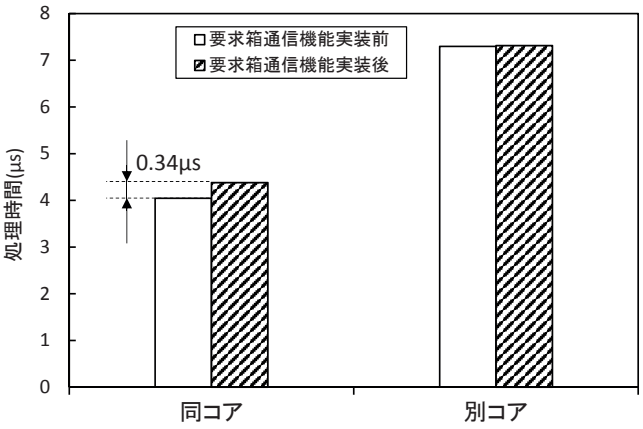

図 9　基本通信

4.2 基本性能

4.2.1 基本通信性能

評価に使用した計算機を**表 1** に示し，基本通信における各プロセスの処理流れを**図 8** に示す．AP プロセスが OS サーバに依頼を発行する callsync() システムコールを発行してから，OS サーバによる結果返却を受け取って callsync() システムコールを終了するまでの処理時間を測定し，100 回繰り返す． 平均処理時間を**図 9** に示す．図 9 より，以下のことが分かる．

（1）AP プロセスと OS サーバを同コアに配置した場合，要求箱通信機能の実装により処理時間が 0.34μ s 遅くなっている．これは，要求箱通信機能を実装した場合，依頼取得の際に依頼キューに依頼が登録されていない時，無限待ちではなく一定時間待ち (Tw=5ms) に遷移し，周期タイマのセットが必要になるためである．

（2）AP プロセスと OS サーバを別コアに配置した場合，要求箱通信機能を実装していない場合と実装済みの場合で処理時間は同じである．これは，OS サーバがタイマをセットしている間，AP プロセスが依頼取得を行っているためである．同コアでの通信の場合，OS サーバは結果返却後，自身の依頼キューと要求箱の依頼キューを確認し，WAIT 状態に遷移するまでプロセス切替えを行わない．しかし別コアでの通信の場合，OS サーバの結果返却が終わった段階で AP プロセスが RUN 状態になるため，OS サーバの依頼取得時の周期タイマセットによるオーバヘッドが隠蔽される．

（3）同コアと別コアの両方において，要求箱通信機能を実装済みの場合では，OS サーバがタイマをセットするため，AP プロセスが処理依頼を行う際，OS サーバにセットされているタイマを削除する．しかし，別コアの場合における要求箱通信機能実装前と実装後を比較すると，処理時

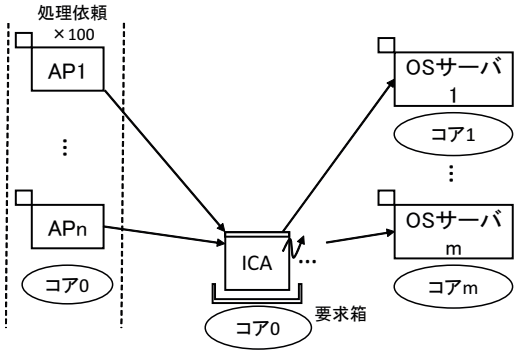

図 10 AP プロセスと OS サーバの関係

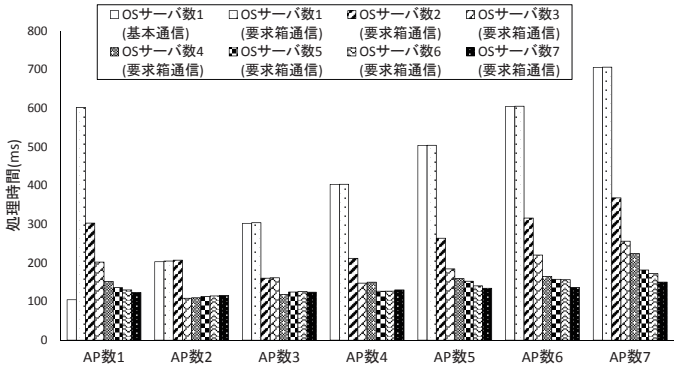

図 11 複数コアにおける分散効果

間が同じであるため，タイマ削除によるオーバヘッドが非常に小さいことが分かる．

4.2.2 複数コアにおける分散効果

複数コアに OS サーバを分散配置し，基本通信と要求箱通信の性能を評価する．

AP プロセスと OS サーバの関係を**図 10** に示す．評価に使用した計算機は 4.2.1 項と同じである．AP プロセスは，callsync システムコールを用いて，要求箱に処理依頼を 100 回行う．OS サーバは，1ms の PU 処理を行う．なお，Tw=5ms とした．各 AP プロセスと要求箱は全て同じコア (コア 0) に配置し，各 OS サーバは全て別コア (コア 1〜コア 7) に配置した．評価結果を**図 11** に示す．図 11 より，以下のことが分かる．

（1）要求箱通信において，AP 数に関係なく，OS サーバ数の増加に伴い，処理時間が減少する．これにより，OS サーバを分散配置することで OS サーバ処理の負荷分散が可能であるといえる．

（2）AP プロセスが 1 つの場合，OS サーバの数に関わらず，基本通信が要求箱通信よりも処理時間が短い．これは，要求箱に依頼が存在しない場合，OS サーバが WAIT 状態に遷移するためである．AP プロセスは OS サーバからの結果を取得するまで次の処理依頼を行えないため，OS サーバが結果返却を行った後すぐに依頼キューを確認した場合，要求箱に依頼が登録されておらず，依頼取得を行えず，Tw の WAIT 状態に遷移する．このため，基本通信の場合よりも処理時間が長くなる．

（3）要求箱通信において，OS サーバ数が AP プロセス数よりも 2 つ以上多い場合，分散効果は発揮されない．例えば，AP プロセス数が 2 の場合，OS サーバ数が 3 の場合と 4 の場合で処理時間は同じである．これは，AP プロセスが要求箱に登録する依頼の数よりも OS サーバの数の方が多い場合，要求箱の参照時に依頼を取得できない OS サーバが発生するためである．

（4）AP プロセスが 2 つ以上の場合，OS サーバが 1 つの際の基本通信と要求箱通信の処理時間は同じである．これは，複数の AP プロセスが要求箱に依頼を登録することで，常に要求箱の依頼キューに依頼が登録されている状態になり，OS サーバが WAIT 状態にならないためである．これにより，複数の AP プロセスが処理依頼を行う環境では，OS サーバの数に関わらず，要求箱通信を用いることによるオーバヘッドが発生しないことがわかる．

表 2 通信処理性能評価に利用した計算機

	送信側計算機	受信側計算機
OS	***AnT***	FreeBSD 6.3-RELEASE
CPU	Intel Core i7-2600 (3.4 GHz) 4 コア	
メモリ	8GB	
NIC	RTL8139（2 枚）(100Mbps)	Intel PRO/1000 MT(1Gbps)

4.3 通信処理性能

4.3.1 評価環境

評価に利用した計算機を**表 2** に示し，評価の処理流れを**図 12** に示す．送信側計算機にデータ送信要求を行う AP プロセスを複数用意し，受信側計算機には AP プロセスと同じ数のサーバを用意する．送信側計算機の AP プロセスと受信側計算機のサーバは対応しており，AP1 が送信したデータはサーバ 1 が受信する．2 つの NIC ドライバを 1 コアと 2 コアに分散配置して通信処理を行う．基本通信機能では，AP プロセスが通信制御サーバに処理依頼を行い，通信制御サーバは 2 つの NIC ドライバに対して交互 [10] に処理依頼を行う．要求箱通信機能では，AP プロセスが通信制御サーバに処理依頼を行い，通信制御サーバは要求箱に対して処理依頼を行い，各 NIC ドライバは自身が依頼を処理していない場合，要求箱から依頼取得を行う．なお，Tw=5ms とする．

AP プロセスと受信側計算機のサーバの処理流れを**図 13** に示す．AP プロセスはデータをサーバに送信し，サーバは AP プロセスから送信されたデータを受信したのち，AP プロセスに対して 21B のデータを送信する．サーバが recv を発行する直前から AP プロセスへの send を終了するまでの処理時間を測定し，10 回繰り返す．10 回の測定結果の内，2 回目から 10 回目までの合計の処理時間を測定する．

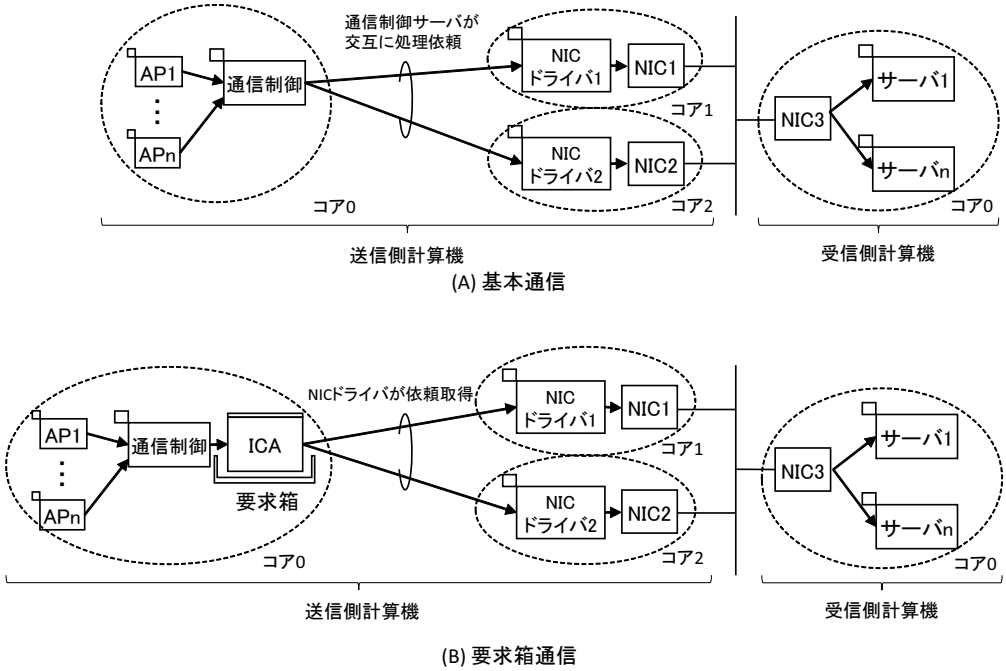

図 12 通信処理性能評価の処理流れ

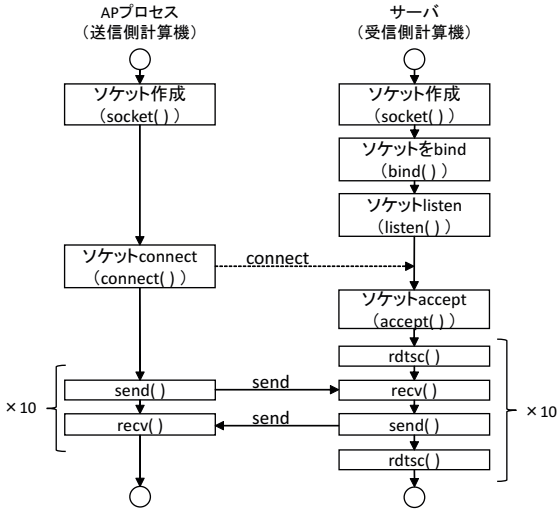

図 13 AP プロセスとサーバの処理流れ

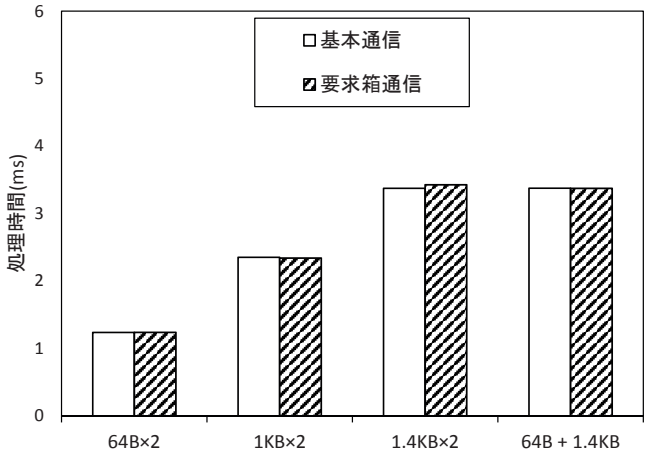

図 14 データ送受信の処理時間 (AP 数が 2 の場合)

これは，1 回目の測定では，サーバ側が accept の終了後すぐに rdtsc 命令を発行するため，AP プロセスの connect 処理が測定区間に含まれる恐れがあるためである．

4.3.2 考察

2 つの AP プロセスが同じデータ長 (64B，1KB，1.4KB) を送信した場合，および異なるデータ長 (64B と 1.4KB) を送信した場合について，処理時間を**図 14** に示す．図 14 から以下のことが分かる．

（1）図 14 において，データの送信サイズが 64B × 2，1KB × 2，および 1.4KB × 2 の時，基本通信と要求箱通信のそれぞれの場合で，データ送受信全体の処理時間はほぼ同じである．これは，データのサイズが等しい場合，それぞれの NIC にかかる負荷が等しいため，基本通信の場合と要求箱通信の場合の NIC ドライバの動作状況が同じになるからであると考えられる．

（2）図 14 において，データの送信サイズが 1.4KB × 2 の場合と 64B + 1.4KB の場合で，データ送受信時間はほぼ同じである，これは，64B の AP プロセスのデータ送受信が終了した後にも，1.4KB のデータ送信を行う AP プロセスの送受信が完了していないためである．

この時，各 AP プロセスが同じデータ長 (1.4KB) を送信した場合と異なるデータ長 (64B と 1.4KB) を送信した場合について，受信側計算機のサーバが発行した recv と send のタイムスタンプ値を**図 15** に示す．図 15 より，以下のことが分かる．

（3）図 15 において，データ送受信全体の処理時間は基本通信と要求箱通信でほぼ同じである．一方で，64B のデータ送信を行うプロセス 1 の送受信時間は，基本通信機能

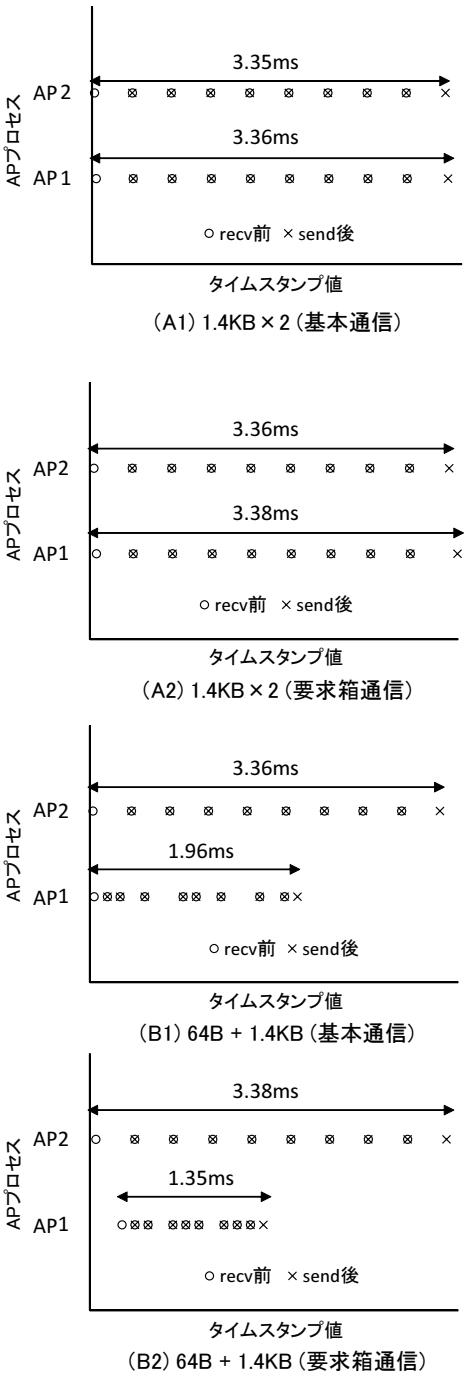

図 15 各プロセスにおけるデータ送受信間隔

が 1.96ms であるのに対し，要求箱通信は 1.35ms となり，0.61ms 短い．これは，基本通信の場合，64B の送信依頼と 1.4KB の送信依頼を NIC ドライバの負荷を考慮せずに交互に処理依頼を行っているのに対し，要求箱通信機能では負荷の小さい NIC ドライバが要求箱から自発的に依頼取得を行うためである．これにより，要求箱通信では OS サーバに効率的に処理依頼を分散可能であることが分かる．

次に，1 つの AP は 1.4KB のデータを送信し，64B のデータを送信する AP の数を変化させた場合の処理時間を **図 16** に示す．図 16 より，以下のことが分かる．

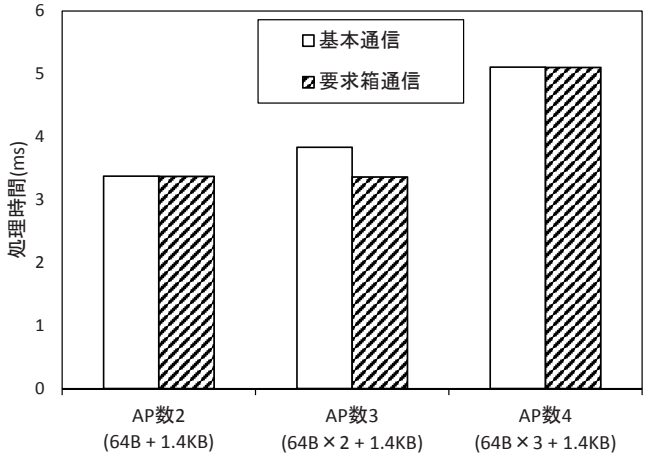

図 16 データ送受信の処理時間 (AP 数を変化させた場合)

（1）AP プロセス数が 3 の場合，要求箱通信機能は基本通信機能の場合よりも合計処理時間が短くなる．これは，基本通信機能では NIC ドライバの負荷状況を考慮しないため．1.4KB のデータを処理している負荷の大きい NIC ドライバに送信処理依頼を発行してしまうためであると考えられる．要求箱通信機能では負荷の小さい OS サーバが依頼取得を行うため，片方の NIC ドライバが 1.4KB の処理依頼を取得した場合，もう片方の NIC ドライバが 64B のデータ処理依頼を複数回取得することでどちらかの NIC ドライバに負荷が偏らないように分散される．

（2）AP プロセス数が 4 の時，合計処理時間は AP プロセス数 3 以下の場合と比べて長くなる．また，基本通信機能の場合と要求箱通信機能の場合で処理時間が同じになる．これは，AP 数の増加により，NIC ドライバの処理がボトルネックとなるためである．基本通信機能と要求箱通信機能の場合の両方で 2 つの NIC ドライバが常にデータを処理しているため，どちらの通信機能を用いた場合でも処理時間は同じである．

5. 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，OS サーバを各コアに分散することで OS 処理の負荷分散を可能にする要求箱通信機能について提案し，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できる．評価では，要求箱通信機能の実装による基本通信の処理時間への影響について評価した．要求箱通信機能の実装後の ***AnT*** は，実装前の ***AnT*** と比較し，基本通信の処理時間が $0.34\mu s$ 増加した．これは，要求箱通信機能を用いた場合，OS サーバが WAIT 状態に遷移する際にタイマを設定する必要があるためである．また，要求箱通信の分散効果を評価するため，通信ネットワーク処理にお

いて，2 つの NIC ドライバを用意し，データ送受信処理の性能を評価した．評価結果より，データの送信サイズが 64B+1.4KB の場合，64B のデータ送受信の処理時間が，基本通信機能では 1.96ms であるのに対し，要求箱通信では 1.35ms となり，0.61ms 短くなった．これにより，要求箱通信を用いた場合の方が NIC ドライバ処理を効果的に分散できることを示した．

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] 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).
[5] 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-Time Applications (OSPERT16) pp. 11–16(2016)
[6] 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).
[7] 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).
[8] Baumann, A., Barham, P., Dagand, P. E., Harris, T., Isaacs, R., Peter, S., Singhania, A. : The multikernel: a new OS architecture for scalable multicore systems, In Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles pp. 29–44(2009)
[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).
[10] Pattanaik, P. A., Roy, S., Pattnaik, P. K.: Performance study of some dynamic load balancing algorithms in cloud computing environment, 2nd International Conference on Signal processing and integrated networks (SPIN), pp. 619–624(2015)