高スループットを実現するOS処理分散法の実現

佐古田 健志[1]　山内 利宏[1]　谷口 秀夫[1]

概要：計算機ハードウェア性能と通信速度の向上により，映像や画像といった大容量のデータを高速に送受信することが可能になっている．このようなデータの送受信に用いる通信路には，高スループットが求められる．このため，複数の低速度な通信路を利用し，高スループットを実現する制御法の研究が行なわれており，モノリシックカーネル構造のオペレーティングシステムにおいて既に実現されている．この制御法は，マルチコアプロセッサ上で動作する環境において，NIC を複数搭載することにより実現できる．しかし，複数の NIC ドライバ処理へのパケット分割によるプロトコル制御のオーバヘッドが大きくなるため，プロトコル制御を含む通信制御処理がボトルネックになり，スループットの向上に制約がある．そこで，本稿ではマイクロカーネル構造の OS において NIC ドライバ処理と通信制御処理を実行する OS サーバを複数起動し，各コアに分散することで高スループットを実現する OS 処理分散法を提案する．また，マイクロカーネル構造である ***AnT*** オペレーティングシステムにおける OS 処理分散法の実現方式について述べ，評価結果を報告する．

Implementation of Distribution Method of OS Processing for High Throughput

Takeshi Sakoda[1]　Toshihiro Yamauchi[1]　Hideo Taniguchi[1]

1. はじめに

計算機ハードウェア性能と通信速度の向上により，映像や画像といった大容量のデータを高速に送受信することが可能になっている．このようなデータの送受信に用いる通信路には，高スループットが求められる．一方，10 Gigabit Ethernet を用いて高速な通信速度を得ることは可能になったものの，高価である．

上記の課題への対処として複数の低速度な通信路を利用し，高スループットを実現する制御法の研究が行なわれている [1]．つまり，複数の NIC（Network Interface Card）を利用した通信制御法である．この制御法は，モノリシックカーネル構造のオペレーティングシステム（以降，OS と呼ぶ）で実現されている [2]．この制御法は，マルチコアプロセッサ [3] 上で動作する環境（以降，マルチコア環境）において，NIC を複数搭載することにより実現できる．しかし，複数の NIC ドライバ処理へのパケット分割によるプロトコル制御のオーバヘッドが大きくなるため，プロトコル制御を含む通信制御処理がボトルネックになり，スループットの向上に制約がある．

一方，OS 構造の一つとしてマイクロカーネル構造 [4]~[7] がある．マイクロカーネル構造 OS は，割り込み処理や例外処理といった最小限の OS 処理をカーネルとして実現し，その他の通信制御処理や NIC ドライバ処理をプロセス（以降，OS サーバと呼ぶ）として実現する．このため，マルチコア環境において，これらの処理の OS サーバを各コアに分散することでスループットの向上が期待できる．しかし，既存のマイクロカーネル構造では，OS サーバを各コアに分散しても単一のカーネルがすべてのコアを管理しているため，コア間の独立性が低く，高スループットが得難い [5]．

そこで，本稿では，マイクロカーネル構造 OS において高スループットを実現する OS 処理分散法を提案する．具体的には，各コアにカーネルを配置させ，各コアを独立して管理させる．また，NIC ドライバ処理を実行する OS サーバ（以降，NIC ドライバサーバと呼ぶ）と通信制御処

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

理を実行する OS サーバ（以降，通信制御サーバと呼ぶ）を複数起動し，各コアに分散して動作させることで，NIC ドライバ処理だけでなく通信制御処理に関しても負荷を分散する．

また，マイクロカーネル構造である ***AnT*** オペレーティングシステム（An operating system with adaptability and toughness）[8] における OS 処理分散法の実現方式について述べ，評価結果を報告する．

2. 既存研究

2.1 モノリシックカーネル

低速度な通信路を組合せることで高スループットを実現する既存研究として Linux に実現されている Bonding 機能がある．Bonding 機能は，複数の通信路を束ねて一つの通信路として利用する機能である．Bonding 機能における負荷分散の実現は，4 つのモードとして実現されている．以下で 4 つのモードについて説明する．

（1）balance-rr(Round Robin)
接続されている NIC に対してラウンドロビン方式でパケット送信依頼を行なう NIC を分散する．パケットサイズと NIC の負荷状況は考慮されない．

（2）balance-xor
送信先計算機の MAC アドレスを 16 ビットの定数と XOR 計算する．計算結果と応じてパケット送信依頼を行なう NIC を分散する．

（3）balance-atb
各 NIC の負荷を 10 秒ごとに確認し，負荷の状況に応じてパケット送信依頼を行なう NIC を分散する．

（4）balance-tlb
（3）の機能に加え，パケット受信時の ARP パケット受信を契機に NIC の負荷を確認し，負荷の状況に応じてパケット受信する NIC を分散する．この際，パケット受信する NIC の MAC アドレスと IP アドレスの対応表を書き換える．このため，パケット送信に関しては，（3）の機能と同様の機能を有する．一方で，パケット受信に関しては，（3）の機能とは異なり，パケット受信においても負荷分散を実現できる．

上記で述べた Bonding 機能の 4 つのモードにおける通信ネットワーク処理の構成を**図 1** に示す．

図 1 は，NIC を複数搭載することにより実現している．しかし，複数の NIC ドライバ処理へのパケット分割によるプロトコル制御のオーバヘッドが大きくなるため，通信制御処理の負荷は大きい．このため，プロトコル制御を含む通信制御処理がボトルネックになり，スループットの向上に制約がある．

2.2 マイクロカーネル

マルチコア環境でマイクロカーネル構造 OS を動作させた場合，通信制御処理を含む多くの OS 処理を OS サーバとして実現しているため，通信制御サーバを複数起動させることで，通信制御処理の負荷を分散することが可能である．しかし，MINIX のように単一のカーネルですべてのコアを管理するマイクロカーネル構造 OS では，OS サーバを各コアに分散させても，各コアが独立して動作できないため，高スループットを実現しにくい．

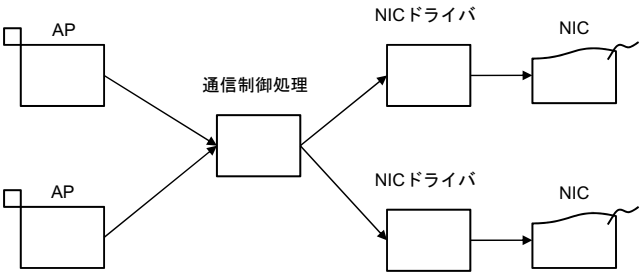


図 1 Bonding 機能における通信ネットワーク処理の構成

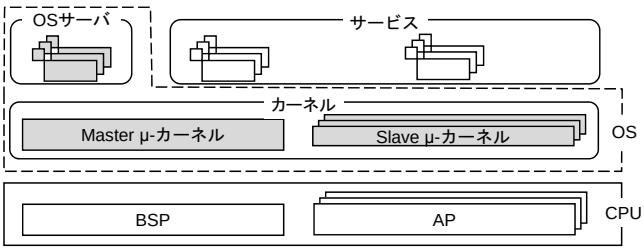


図 2 Multi Microkernel の基本構造

一方で，マルチコア環境に対応し，各コアを独立して管理する構造になっているマイクロカーネル構造 OS として Multi Microkernel[9] がある．

Multi Microkernel の基本構造を**図 2** に示す．OS はカーネルと OS サーバからなる．カーネルは，Master マイクロカーネルと Slave マイクロカーネルからなる．Master マイクロカーネルは，カーネルに必要な全機能を有し，システム全体を管理する．Slave マイクロカーネルは，AP の実行のみに機能を制限されたカーネルであり，Slave マイクロカーネルが管理する資源以外の資源を利用する場合，Master マイクロカーネルを経由してのみ利用可能になる．

Multi Microkernel は，OS 処理を OS サーバとして実現している．しかし，Multi Microkernel では，OS サーバを分散できない．これは，OS サーバへの処理依頼の機能を Master マイクロカーネルのみが有しており，Slave マイクロカーネルからは OS サーバへ処理依頼できないためである．つまり，OS 処理を利用するためには，必ず Master マイクロカーネルを経由する必要があるといった制約があり，OS 処理を分散できていない．そこで，マイクロカーネル構造 OS の特徴である OS 処理を OS サーバとして実現する利点を生かしながら，OS 処理を各コアへ分散し，独立

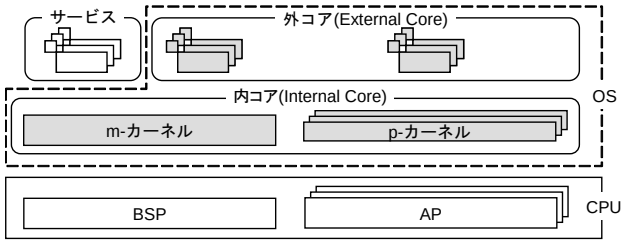


図 3 マルチコア ***AnT*** の基本構造

して実行できる新たなマイクロカーネル構造 OS が必要になっている．このため，我々は，各コアを独立して制御し，OS サーバを各コアに分散可能な独自のマイクロカーネル構造を持つマルチコア環境用の ***AnT*** オペレーティングシステム [10] を提案している．

3. *AnT* オペレーティングシステム

3.1 基本構造

マルチコア環境用の ***AnT*** オペレーティングシステム (以降，マルチコア ***AnT*** と呼ぶ) の基本構造を**図 3** に示す．OS は，内コアとプロセス (OS サーバ) として動作する外コアからなる．内コアには，m-カーネル（BSP（Boot Strap Processor）上で動作し最初に起動するカーネル）と p-カーネル（AP（Application Processor）上で動作し m-カーネルにより起動されるカーネル）がある．m-カーネルは，カーネルに必要な全機能を有する．一方，p-カーネルは，機能を例外・割り込み機能，サーバプログラム間通信機能，およびスケジューラ機能に絞り，軽量化を図っている．外コアは，適応したシステムに必須なプログラム部分であり，例えば，通信制御処理や NIC ドライバ処理を OS サーバとして提供する．OS サーバは，実行する処理に対応した一意な識別子（以降，コア ID と呼ぶ）を有している．このコア ID を用いて処理依頼先の OS サーバを識別する．

AnT の仮想空間の構成は，多重仮想記憶である．さらに，特徴的な領域として，コア間通信データ域（以降，ICA (Inter-core Communication Area)）がある．この領域は，内コア，外コア，およびサービスが相互のデータ通信に利用する空間である．

3.2 サーバプログラム間通信機構

AnT は，高速なサーバプログラム間通信機構 [11] を有しており，この様子を**図 4** に示す．図 4 に示すように，依頼元プロセスと依頼先 OS サーバがそれぞれに通信用の依頼キューと結果キューを有し，各キューへ処理依頼の情報を書き込んだ ICA（以降，制御用 ICA と呼ぶ）を繋ぎ換えることで高速な通信機構を実現している．さらに，マルチコア ***AnT*** では，マルチコア環境でのサーバプログラム

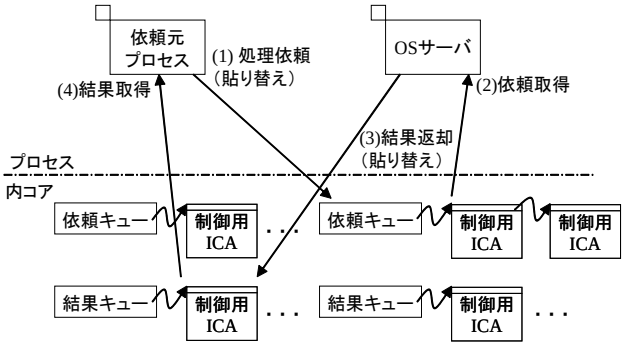


図 4 ***AnT*** のサーバプログラム間通信機構の処理流れ

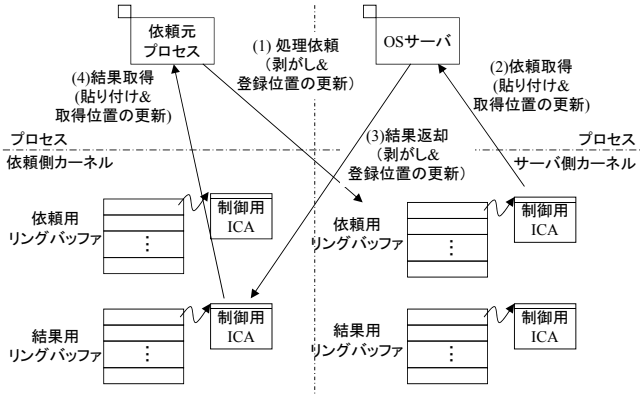


図 5 マルチコア ***AnT*** のサーバプログラム間通信機構の処理流れ

間通信機構を高速化している．この様子を**図 5** に示す．これは，マルチコア環境では，従来のキューを用いた通信機構である場合，複数のプロセスが同時にキュー操作を実行する可能性があるため，キュー操作に排他制御が必要になり，排他制御オーバヘッドが生じる．そこで，排他制御をできるだけ排除した高速なサーバプログラム間通信機構を実現している．具体的には，OS サーバ間の通信相手を制限し，通信時のキュー操作を 2 排他制御にしている．さらに，リングバッファを用いた制御機構により，排他制御を行なわない制御用 ICA の貼り替えを実現している．また，ICA の授受方式を改善し，各プロセスの動作するコア上のカーネルがサーバプログラム間通信時における ICA の貼り替えを行なっている．このマルチコア環境での高速なサーバプログラム間通信機構を m-カーネルだけでなく，p-カーネルも利用できるようにすることで，すべてのプロセスは，動作するコアに関わらずに各コアに分散した OS サーバへ処理依頼可能になる．

3.3 処理分散機能

マルチコア ***AnT*** では，マルチコア環境に適したプロセス移動（プロセスマイグレーション）機能を実現し，プロセス移譲機能と名づけている [12]．プロセス移譲機能は，

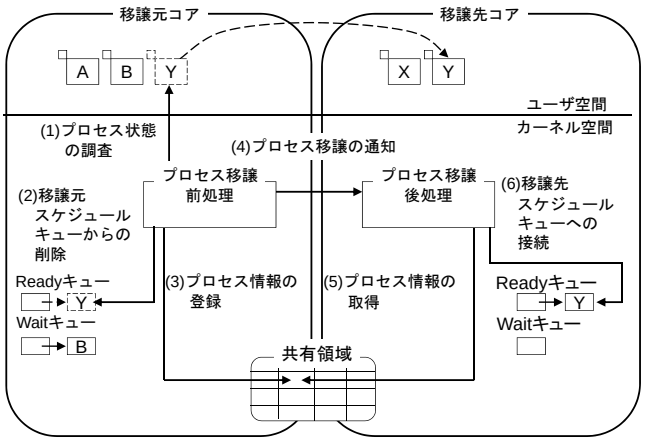


図 6 プロセス移譲機能の処理流れ

移譲を実行するコア（以降，移譲元コアと呼ぶ）上で走行しているプロセスを任意のコア（以降，移譲先コアと略す）に移譲する機能である．プロセス移譲機能の処理流れを**図 6** に示し，以下に説明する．

（1）移譲可能なプロセス状態の判定を行なう．

（2）移譲対象プロセスが RUN 状態の場合，プロセスの状態を READY 状態に変更し，プロセスのコンテキストを保存する．また，移譲対象プロセスが READY 状態，または WAIT 状態の場合，それぞれのスケジュールキューからプロセスを削除する．

（3）移譲対象プロセスのプロセス情報を共有領域へ書き込む．

（4）IPI を用いて移譲先コアへプロセス移譲を通知する．

（5）移譲対象プロセスのプロセス情報を共有領域から読み込む．

（6）(2) の処理により，すべてのプロセスの状態は READY 状態か WAIT 状態のどちらかに変更されているため，該当するスケジュールキューへプロセスを接続する．

また，分散したプロセスが別コアの OS 処理を利用する方式として，m-カーネルへの処理依頼と m-カーネル専用機能の共有化を実現し，要求処理速度に応じて適用している．

m-カーネルへの処理依頼は，p-カーネル上のプロセスが発行したカーネルコール処理を m-カーネルに処理依頼するものである．

m-カーネル専用機能の共有化は，p-カーネルからも m-カーネル専用機能を利用可能にするものであり，共有化するにあたり，m-カーネル専用機能に排他制御を用いる．

4. 提案手法

4.1 考え方

プロトコル制御を含む通信制御処理への処理依頼が集中するため，高スループットを実現しにくい．そこで，通信制御サーバを複数起動させることで，通信制御サーバへの処理依頼を分散可能にする．このために，通信制御サーバを複数起動させる方式が必要になる．さらに，複数起動した通信制御サーバへの処理依頼を分散する方式が必要になる．

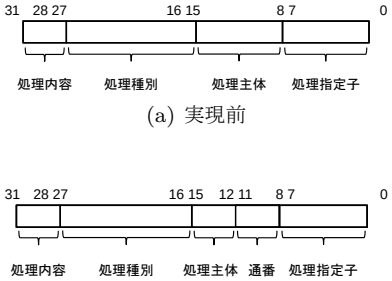


図 7 コア ID

4.2 課題と対処

4.1 節の考え方に沿って通信制御処理の分散を実現する．通信制御処理の分散を実現するための課題は 2 つある．2 つの課題を以下に示す．

（課題 1）通信制御サーバを別々の OS サーバとして複数起動させる方式の確立

（課題 2）複数の通信制御サーバへの処理依頼を分散する方式の確立

（課題 1）は，同等の OS 処理を実行する OS サーバを識別する必要があるためである．これは，同等の OS 処理を実行する OS サーバは同一のコア ID を有することに起因する．対処として，コア ID に同等の OS 処理を実行する OS サーバが複数起動した場合の通番を格納する領域を用意する．これにより，通信制御サーバを複数起動させても，それぞれの通信制御サーバを個別に識別可能になる．

（課題 2）は，通信制御サーバへの処理依頼において，同一のインタフェースを用いて複数起動した通信制御サーバへの処理依頼を分散する必要があるためである．これは，通信制御サーバへの処理依頼が発生するたびに処理依頼先の通信制御サーバを手動で設定するのにはコストがかかり，また，手動で設定する場合，効率的な負荷分散が実現できないことに起因する．対処として，同一のインタフェースを利用した際に通信制御サーバへの依頼回数を通信制御サーバごとに管理する．これにより，通信制御サーバへの処理依頼を平均化することができ，負荷分散が実現できる．

4.3 実現方式

4.3.1 通信制御サーバを複数起動させる方式

前述の（課題 1）について，通信制御サーバを複数起動させる方法をコア ID を改変することにより実現する．コア ID を**図 7** に示す．実現前のコア ID（**図 7(a)**）は，処理内容，処理種別，処理主体，および処理指定子によって構成

されている．処理内容は，read や write といった OS サーバが実行する処理の内容を表す．処理種別は，通信制御処理や NIC ドライバ処理といった OS サーバが実行する処理の種別を表す．処理指定子は，どのデバイスがどのような処理を行なうかといった処理内容の詳細を表す．処理主体は，制御するデバイスの種類を表す．具体的には，NIC である RealTek 8139 や IntelPRO1000MT といったデバイスの種類を区別するために本フィールドの上位 4bit を利用する．つまり，同一の NIC ドライバサーバを複数起動する場合，起動した NIC ドライバサーバは同一の処理種別を有し，同一の処理主体になり，別々の NIC ドライバサーバとして識別できない．このため，複数起動した NIC ドライバサーバを区別するために本フィールドの下位 4bit を利用する．これにより，NIC ドライバサーバに関しては複数起動しても各 NIC ドライバサーバを個別に識別可能である．

この処理主体の下位 4bit を NIC ドライバサーバを区別する目的にのみ利用するのではなく，複数起動したすべての OS サーバを区別する通番のフィールドとして利用する．この対処により，複数起動した通信制御サーバについて，通番を割り振ることで個別に識別可能になる．対処を実現した場合のコア ID を**図 7(b)** に示す．

4.3.2 複数起動した通信制御サーバへの処理依頼を分散する方式

前述の（課題 2）について，複数起動した通信制御サーバへの処理依頼の分散を通信制御サーバへの依頼回数を保存することで実現する．実現方式として以下の 2 案がある．

（案 1）ソケット作成時に通信制御処理内で保存

（案 2）通信制御サーバへの処理依頼を実行するカーネル内で保存

（案 1）は，通信制御サーバへのソケット作成の処理依頼以外に影響を与えない．しかし，ソケット作成の処理依頼において，通信制御サーバ内で他の通信制御サーバへの処理依頼回数との同期をとる必要があるため，ソケット作成の処理時間は増加する．一方，（案 2）では，通信制御サーバへの処理依頼時にカーネル内で依頼回数を保存するため，通信制御サーバ間で同期をとる必要はない．しかし，通信制御サーバへのすべての処理依頼においてオーバヘッドが発生する．

通信制御サーバへの処理依頼を分散する場合の分散の粒度を考える．このとき，パケット送信を実行するソケット単位で処理依頼を分散することが考えられる．ソケット単位で処理依頼を分散する場合であるならば，パケット送信に利用するソケット作成時に通信制御サーバへの処理依頼先を分散すればよいため，（案 1）で示した通りソケット作成の処理依頼にのみオーバヘッドが発生する．しかし，ソケット単位で処理を分散したとしても，パケット送信処理は，複数のパケットを送信することが考えられる．このため，パケット送信の処理依頼を発行するたびに通信制御サーバへの処理依頼が発生し，（案 2）によるオーバヘッドの増加回数はパケット送信回数に比例して増加する．したがって，パケット送信処理の回数に応じてオーバヘッドが増加する（案 2）では，通信制御サーバへの処理依頼を分散しても効果が挙げられないと考えられる．このため，（案 1）を採用し，ソケット作成時に通信制御サーバへの処理依頼を分散する．

表 1 評価環境

	クライアント側計算機	サーバ側計算機（2 台）
CPU	Core i7-2600 (3.4GHz)	Core i7-2600 (3.4GHz)
NIC	IntelPRO1000MT（2 枚）	IntelPRO1000MT

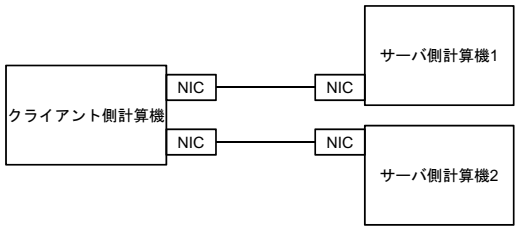


図 8 評価環境

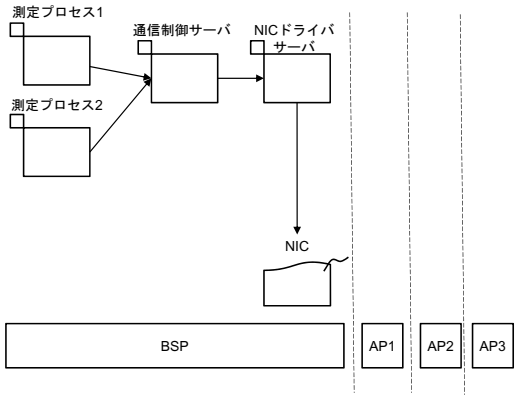
5. 評価

5.1 観点と評価環境

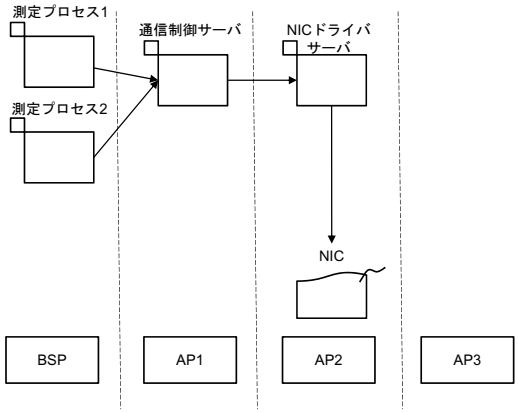
通信ネットワーク処理の高スループットを得るには通信制御処理の負荷分散が効果的である．ここでは，プロトコル制御を含む通信制御サーバに着目し，その負荷分散の効果について明らかにする．

評価環境を**表 1** と**図 8** に示す．***AnT*** を物理コアが 4 コアである Core i7-2600（3.4GHz）を搭載したクライアント側計算機で動作させ評価した．***AnT*** では，通信ネットワーク処理を通信制御サーバと NIC ドライバサーバにより実現している．この ***AnT*** の通信ネットワーク処理を利用して，2 つの測定プログラムのプロセス（以降，測定プロセスと呼ぶ）が同時にパケット送信処理を実行した場合の性能を測定する．なお，測定プロセスは，ソケットの作成，通信先計算機とのコネクションの確立，1024Bytes パケットの複数回送受信，およびソケットの削除を行なう．

今回の評価では，従来の通信ネットワーク処理の構成（従来：**図 9**），***AnT*** 上に実装した Bonding 機能の構成（Bonding：**図 10**），および NIC ドライバと同数だけ通信制御サーバを起動した構成（提案手法：**図 11**）の 3 通りで評価した．このうち、***AnT*** 上に実装した Bonding 機能とは，NIC ドライバサーバを複数起動し，NIC ドライバサーバへの処理依頼を通信制御サーバが動的に変更して処理依頼する機能である．このため，Bonding 機能を利用してパケット送信する場合，依頼元プロセスが発行したパケット送信処理依頼が毎回同一の NIC ドライバサーバに対して

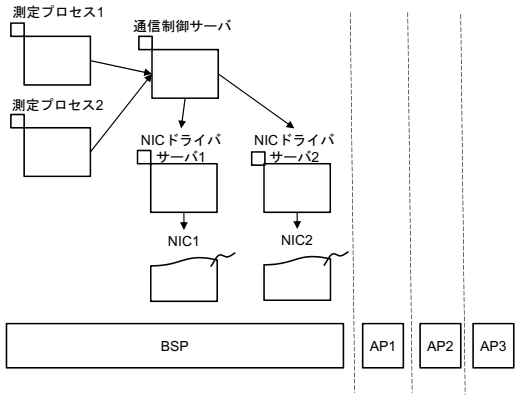


(a) 分散無

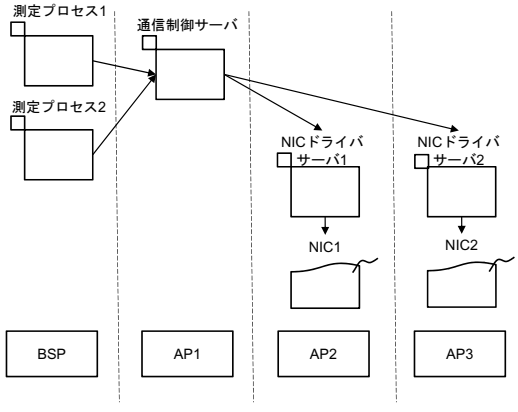


(b) 分散有

図 9 従来の通信ネットワーク処理での動作状況



(a) 分散無



(b) 分散有

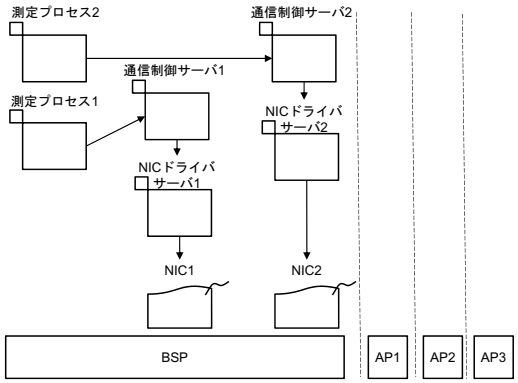
図 10 ***AnT*** に実装した Bonding 機能での動作状況

通知されるとは限らない．

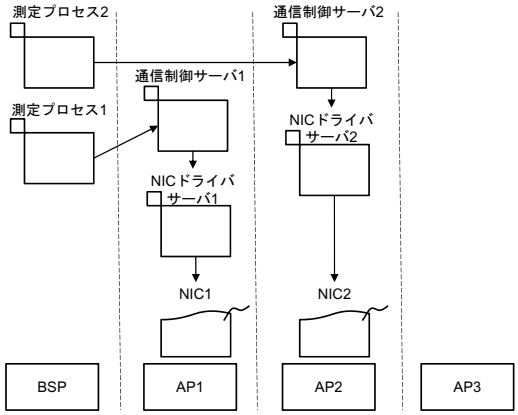
最初に，従来における分散無と分散有の場合での各プロセスの動作状況を図 9 に示す．図 9 では，2 つの測定プロセスが通信制御サーバへ処理依頼する．通信制御サーバは，測定プロセスから受け取った依頼内容のプロトコル制御を実行し，NIC ドライバサーバへ処理依頼する．NIC ドライバサーバは，通信制御サーバから受け取った依頼内容をもとに NIC にデータを書き込み，測定プロセスへ結果を返却する．分散無である**図 9(a)** では，測定プロセス，通信制御サーバ，および NIC ドライバサーバの全プロセスが BSP 上で動作する．つまり，シングルコアプロセッサとして動作している環境といえる．一方，分散有である**図 9(b)** では，測定プロセスは BSP 上，通信制御サーバは AP1 上，および NIC ドライバサーバは AP2 上で動作する．これにより，測定プログラム，通信制御サーバ，および NIC ドライバサーバは並列動作可能になる．

次に，***AnT*** に実装した Bonding 機能における分散無と分散有の場合での各プロセスの動作状況を図 10 に示す．図 10 では，2 つの測定プロセスが通信制御サーバへ処理依頼する．通信制御サーバは，測定プロセスから受け取った依頼内容のプロトコル制御を実行し，動作状況を考慮して NIC ドライバサーバ 1，もしくは NIC ドライバサーバ 2 へ処理依頼する．NIC ドライバサーバ 1 と NIC ドライバサーバ 2 は，通信制御サーバから受け取った依頼内容をもとにそれぞれの NIC ドライバサーバが制御する NIC にデータを書き込み，送信プロセス 1，もしくは送信プロセス 2 へ結果を返却する．分散無である**図 10(a)** では，従来の通信ネットワーク処理の場合と同様に全プロセスが BSP 上で動作する．一方，分散有である**図 10(b)** では，測定プロセスは BSP 上，通信制御サーバは AP1 上，NIC ドライバサーバ 1 は AP2 上，および NIC ドライバサーバ 2 は AP3 上で動作する．これにより，通信制御サーバと 2 つの NIC ドライバサーバは並列動作可能になっている．

最後に，提案手法における分散無と分散有の場合での各プロセスの動作状況を図 11 に示す．図 11 では，測定プロセス 1 と測定プロセス 2 は，それぞれ通信制御サーバ 1，もしくは通信制御サーバ 2 へ別々に処理依頼する．通信制御サーバ 1 と通信制御サーバ 2 は，それぞれ測定プロセス 1，もしくは測定プロセス 2 から受け取った依頼内容のプロトコル制御を実行し，自身と組になる NIC ドライバサーバ 1，もしくは NIC ドライバサーバ 2 へ処理依頼する．NIC ドライバサーバ 1 と NIC ドライバサーバ 2 は，自身と組になる通信制御サーバ 1，もしくは通信制御サーバ 2 から受け取った依頼内容をもとにそれぞれの NIC ドライバサーバ



(a) 分散無



(b) 分散有

図 11 提案手法での動作状況

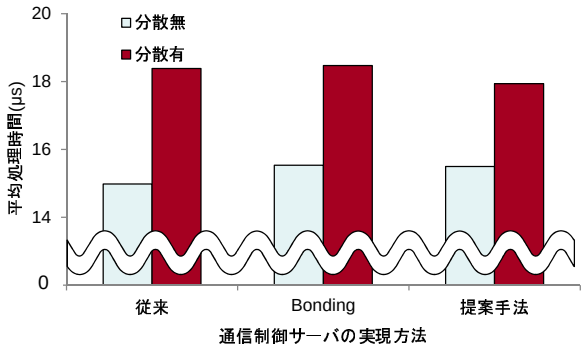


図 12 基本評価 (send())

表 2 send() でのコア間通信の発生回数

	従来	Bonding	提案手法
分散無	0	0	0
分散有	3	3	2

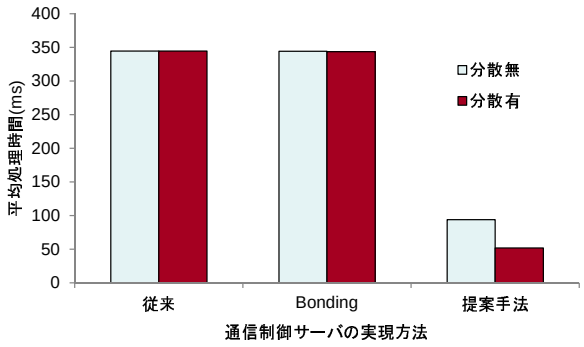


図 13 基本評価 (send() & recv())

が制御する NIC にデータを書き込み，NIC ドライバサーバ 1 は送信プロセス 1 へ，NIC ドライバサーバ 2 は送信プロセス 2 へ結果を返却する．分散無である**図 11(a)** では，従来の通信ネットワーク処理と ***AnT*** に実装した Bonding 機能の場合と同様に全プロセスが BSP 上で動作する．一方，分散有である**図 11(b)** では，測定プロセスは BSP 上，通信制御サーバ 1 と NIC ドライバサーバ 1 は AP1 上，および通信制御サーバ 2 と NIC ドライバサーバ 2 は AP2 上で動作する。これにより，通信制御サーバと NIC ドライバサーバの各組は並列動作可能になる．

5.2 結果と考察

測定プロセスで send() によるパケット送信を 10 回繰り返し，1 回あたりの平均処理時間を測定した．当測定結果を**図 12** に示す．図 12 より，以下のことが分かる．

(1) 通信制御サーバと NIC ドライバサーバを分散する場合，提案手法は従来の通信ネットワーク処理と ***AnT*** 上に実装した Bonding 機能に比べ，わずかに高速である．これは，分散した OS サーバの配置が各手法によって異なるためである．send() を実行した場合のコア間通信の発生回数を**表 2** に示す．提案手法では，測定プロセスから通信制御サーバへの処理依頼時と NIC ドライバサーバから送信プロセスへの結果返却時の計 2 回のコア間通信オーバヘッドが発生する．一方，従来の通信ネットワーク処理と ***AnT*** 上に実装した Bonding 機能では，送信プロセスから通信制御サーバへの処理依頼時，通信制御サーバから NIC ドライバサーバへの処理依頼時，および NIC ドライバサーバから送信プロセスへの結果返却時の計 3 回のコア間通信オーバヘッドが発生する．

次に，パケット送受信が発生する場合での性能を明らかにするため，send() によってサーバ側計算機にパケットを送信し，recv() によってサーバ側計算機からパケットを受信する処理を 10 回繰り返すのに要した処理時間を 3 回測定した．測定結果の平均処理時間を**図 13** に示す．図 13 より，以下のことが分かる．

(2) 提案手法は，従来の通信ネットワーク処理と Bonding 機能の構成に比べ，処理時間が短い．これは，通信制御サーバの処理負荷が減少するためである．2 つの測定プロセスから send() によるパケット送信要求と recv() によるパケット受信要求が頻繁に発生するため，通信制御サーバにはパケット送信の処理依頼とパケット受信の処理依頼

が多発する．提案手法では，通信制御サーバが 2 つ起動しているため，例えば，測定プロセス 1 から通信制御サーバ 1 へパケット送信の処理依頼をした場合でも，測定プロセス 2 から通信制御サーバ 2 への処理依頼と競合することはない．このため，通信制御サーバの処理負荷が軽減され，提案手法では他の方法に比べ平均処理時間が約 15% から 27%に減少した．この結果より，提案手法ではパケット送受信処理においてスループットが向上すると考えられる．

（3）提案手法において，通信制御サーバと NIC ドライバサーバの組を測定プロセスと別のコアに分散することは有効である．これは，通信制御サーバが別コアで動作するため，複数の測定プロセスからの処理依頼を同時に実行できることに起因する．具体的には，分散有である場合の処理時間を分散無である場合の処理時間の約 55% に短縮した．この結果より，提案手法において，通信制御サーバと NIC ドライバサーバの組を測定プロセスと別コアに分散するとスループットが向上すると考えられる．

6. まとめ

通信ネットワーク処理において，通信制御サーバと NIC ドライバサーバを複数起動し，これらの OS サーバを各コアに分散することで高スループットを実現する手法について実現方式とその効果を述べた．

実現方式では，同等の機能を有する OS サーバを識別するために，OS サーバごとに与えらるコア ID に通番を用意することで，すべての OS サーバについて同等の機能を有する OS サーバを複数起動させても識別できるように対処した．また，複数起動した通信制御サーバへの処理依頼を分散するために，分散の粒度をソケット単位に制限することで，ソケット作成時に通信制御サーバへの処理依頼を分散できるように対処した．

評価では，パケット送受信を行なう環境において，提案手法は従来の通信ネットワーク処理と ***AnT*** に実装した Bonding 機能に比べ，平均処理時間を約 15% から 27% に減少することを明らかにした．さらに，提案手法において，通信制御サーバと NIC ドライバサーバの組を別々のコアに分散した場合，分散しない場合に比べ平均処理時間を約 55% に短縮できることを明らかにした．この結果より，提案手法ではスループットが向上すると考えられる．

残された課題として，***AnT*** におけるスループットの評価と Linux の Bonding 機能との比較評価がある．

謝辞 本研究の一部は，科学研究費補助金基盤研究（B）（課題番号：24300008）による．

参考文献

[1] Kamezawa, H., Nakamura. M., Inaba. M., and Hiraki. K.: Coordination between parallel TCP streams on Long Fat Pipe Network, 1st International Workshop on Data Processing and Storage Networking: Towards Grid Computing (2004).

[2] Thomas, D. : Linux Ethernet Bonding Driver HOWTO, available from <https://www.kernel.org/doc/Documentation/networking/bonding.txt>.

[3] Rotem, E., Naveh, A., Rajwan, D., Ananthakrishnan, A., and Weissmann, E.: Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge, IEEE Micro, Vol.32, No.2, pp.20-27 (2012).

[4] Liedtke, J.: Toward real microkernels, Communications of the ACM, Vol.39, No.9, pp.70-77 (1996).

[5] 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).

[6] Black, D.L., Golub, D.B., Julin, D.P., Rashid, R.F., Draves, R.P., Dean, R.W., Forin, A., Barrera, J., Tokuda, H., Malan, G., and Bohman, D.: Microkernel operating system architecture and mach, Journal of Information Processing, Vol.14, No.4, pp.442-453 (1992).

[7] 島崎泰, 山内利宏, 乃村能成, 谷口秀夫: ***AnT*** オペレーティングシステムにおける Linux の LKM 形式ドライバのプロセス化手法，電子情報通信学会論文誌 (D)，Vol.J93-D, No.10，pp.1990-2000 (2010).

[8] 谷口秀夫，乃村能成，田端利宏，安達俊光，野村裕佑，梅本昌典，仁科匡人：適応性と堅牢性をあわせもつ ***AnT*** オペレーティングシステム，情報処理学会研究報告，vol.2006-OS-103，pp.71-78 (2006.07).

[9] Matarneh, R.: Multi Microkernel Operating System for MUlti-Core Processor, Journal of Computer Science, Vol.5, No.7, pp.493-500 (2009).

[10] 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，情報処理学会研究報告，Vol.2012-DPS-150，No.37，pp.1-8 (2012).

[11] 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).

[12] 佐古田健志, 枡田圭祐, 井上喜弘, 谷口秀夫：マルチコア ***AnT*** における処理分散機能，情報処理学会研究報告，Vol.2012-OS-122，No.5，pp.1-8 (2012).