マルチコア向け *AnT* オペレーティングシステムのファイル操作における分散効果の評価

河上 裕太[1]　山内 利宏[1]　谷口 秀夫[1]

概要：マルチコアプロセッサ環境において，AP 処理だけでなく OS 処理も分散できれば，高い分散効果を期待できる．マイクロカーネル構造 OS は，OS 処理をカーネルとプロセスで分担して実現する．このため，OS 処理を提供するプロセス（OS サーバ）を各コアに分散することにより，OS 処理を分散できる．我々は，マイクロカーネル構造を有する ***AnT*** オペレーティングシステムを提案した．本稿では，***AnT*** オペレーティングシステムのファイル操作処理の性能について，ベンチマークを利用し，マルチコアプロセッサにおける AP 処理と OS 処理の分散効果について述べる．

1. はじめに

マルチコアプロセッサが普及し，マルチコアプロセッサに搭載されるコア数も増加している．マルチコアプロセッサを有効利用するには，処理を各コアに分散することが効果的である．特に，OS 処理の多いサービスの効率的な実行には，AP 処理だけでなく OS 処理の分散が必要である．

モノリシックカーネル構造 OS は，データの共有と排他制御により OS 処理の分散を実現する．排他制御は，大別して粗粒度ロックと細粒度ロックの 2 種類があり，並列性向上のために細粒度ロックが多く用いられている．しかし，コア数の多い計算機環境においては，細粒度ロックを用いた場合に性能が低下し易い [1], [2]．一方，マイクロカーネル構造 OS[3], [4], [5] では，OS 機能をカーネルとプロセスで分担して実現する．このため，OS 機能を提供するプロセス（OS サーバ）を各コアに分散することで，OS 処理を分散できる．

我々はマイクロカーネル構造を有する ***AnT*** オペレーティングシステム（***AnT***）を提案した [6], [7], [8]．

本稿では，OS 処理のみを分散した場合，および AP 処理と OS 処理の両方を分散した場合について，ファイル操作における分散効果を述べる．具体的には，ファイル操作処理のベンチマークとして Bonnie ベンチマーク [9] を利用し，その性能を報告する．

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

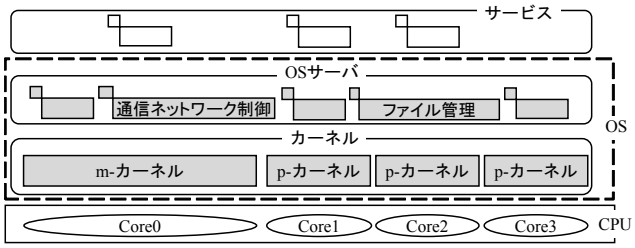

図 1　***AnT*** の基本構造

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 において，カーネルは，最小限の OS 機能を有する部分である．例えば，スケジュール処理といったプロセス実行制御機能を有する．OS サーバは，OS 機能をプロセス化した部分である．例えば，ファイル管理機能や通信ネットワーク制御機能を持つ．なお，サービスはサービスを提供するプログラム部分である．

AnT におけるカーネルは，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，電源投入時に最初に起動するコアを制御し，マイクロカーネルの全機能を有する．一方，p-カーネルは，m-カーネルが制御する以外のコアを制御し，プロセス実行制御機能，コア間通信制御機能，およびサーバ間通信機能を有する．したがって，m-カーネルが保有し p-カーネルが保有しない機能の例として，メモリ管理機能がある．

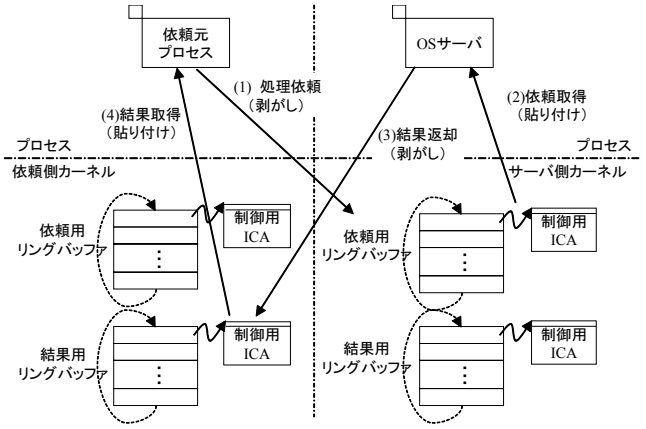

図 2 サーバ間通信機構の処理流れ

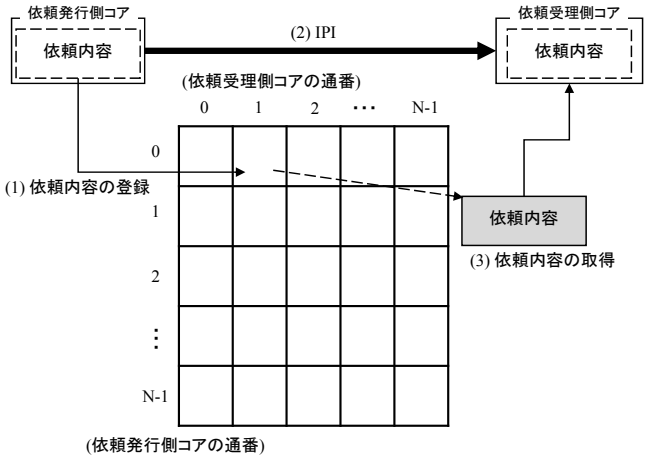

図 3 コア間通信の処理流れ

2.2 サーバ間通信機構

2.2.1 概要

AnT は，マルチコア環境においても高速なサーバ間通信機構 [6], [7], [8] を有している．この機構は，制御用情報とデータ情報をそれぞれコア間通信データ域（ICA：Inter-core Communication Area）と呼ばれるプロセス間での共用領域に格納し，OS サーバ間でのデータ授受を複写レスで実現している．ここで，コア間通信データ域の「コア」は，CPU コアでなく OS サーバを意味する．仮想空間のマッピング表への書き込み（マップ）を貼り付け，マッピング表からの削除（アンマップ）を剥がしと呼んでいる．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 をデータ送信プロセスの仮想空間から剥がし，データ受信プロセスの仮想空間へ貼り付けることで行われる．制御用情報を格納する ICA を制御用 ICA，データ情報を格納する ICA をデータ用 ICA と名付けている．

また，AP プロセスからの処理依頼に対し，OS サーバを多段に経由して処理依頼が発行される場合，1 つの制御用 ICA を繰り返し利用し，多段依頼を実現している．この多段依頼と直接返却と呼ばれる機能により，OS サーバを多段に経由する際の ICA の確保回数とプロセス間通信回数を削減している．これに対し，MINIX は，マイクロカーネル構造 OS であり，***AnT*** と同様に OS 機能を複数プロセスに分割して実現している．しかし，多段依頼や直接返却に相当する機能は実現されていない．

さらに，***AnT*** は，コアごとに用意したリングバッファを用いた通信制御構造により，排他制御オーバヘッドを削減している．

サーバ間通信機構の処理流れを**図 2** に示し，以下に説明する．

(1) 依頼元プロセスが処理依頼を行うと，依頼元プロセスの動作するコア上のカーネル（依頼側カーネル）は OS サーバの依頼用リングバッファに依頼情報を格納した制御用 ICA を登録し，依頼元プロセスから制御用 ICA を剥がす．また，WAIT 状態の OS サーバに依頼情報を格納した制御用 ICA を登録した場合は OS サーバを起床させる．

(2) OS サーバの動作するコア上のカーネル（サーバ側カーネル）は，OS サーバに制御用 ICA を貼り付ける．OS サーバは，依頼用リングバッファ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) OS サーバが結果返却を行うと，サーバ側カーネルは依頼元プロセスの結果用リングバッファに結果情報を格納した制御用 ICA を登録し，自身から結果情報を格納した制御用 ICA を剥がす．また，WAIT 状態の依頼元プロセスに結果情報を格納した制御用 ICA を登録した場合，依頼元プロセスを起床させる．

(4) 依頼側カーネルは，依頼元プロセスに制御用 ICA を貼り付ける．依頼元プロセスは，結果用リングバッファ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．

2.2.2 コア間とコア内のサーバ間通信の差異

コア間とコア内のサーバ間通信の差異は，プロセスを起床させる処理において，コア間通信を伴うか否かである．各コアのカーネルは，個別にスケジュールキューを持ちプロセス実行を制御している．このため，他コアのプロセス起床は， コア間通信を用いて当該コアに依頼する．

2.2.3 コア間通信

コア間通信は，コア間で共有している領域（以降，共有領域）を利用し依頼内容の授受を行うことで実現している．また，依頼内容の登録を通知するために IPI（Inter-Processor Interrupt）を利用している．コア間通信の処理流れを**図 3** に示し，以下に説明する．

(1) 依頼発行側コアは依頼内容を共有領域に登録する．

(2) 依頼発行側コアは依頼受理側コアに IPI を送信し，依頼受理側コアに依頼内容の登録を通知する．

(3) 依頼受理側コアは，IPI の受信を契機に，共有領域から依頼内容を取得し依頼内容の処理を実行する．

また，共有領域について以下に説明する．***AnT*** は，共有領域を $N \times N$（N はコア数）の配列として実現してお

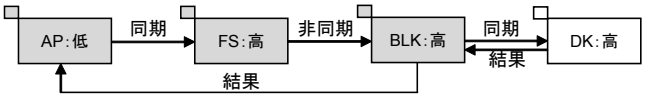

図 4 ファイル操作処理の様子

り，依頼発行側コア（i）と依頼受理側コア（j）のコア間通信は，共有領域の（i, j）番目のエントリを利用して依頼内容を授受する．これにより，複数のコアが 1 つのコアにコア間通信を要求する際の排他制御処理を不要としている．

3. 評価

3.1 観点

AP 処理と OS 処理の分散効果を明らかにするため，Bonnie ベンチマークを用いて評価した．Bonnie は，複数の AP プロセス（ベンチマークプロセス）が単一ファイルのランダムな位置を参照し，指定した確率で参照したデータを更新するベンチマークプログラムである．そこで，ファイル操作処理に関連する OS サーバ群と複数のベンチマークプロセスについて，各コアへの配置の形態（分散形態）を変化させて評価した．

また，Linux 3.16 と ***AnT*** を比較した．なお，Linux 3.16 と ***AnT*** の両方において，ファイルシステムの違いやディスク I/O のボトルネックを避けるため，ファイルデータが全てブロックキャッシュに存在する場合，つまりディスク I/O の生じない場合について評価した．

評価は，Intel Xeon E5-2665(8Core, 2.4GHz) の CPU ソケットを 2 個搭載する計算機を利用した．

3.2 *AnT* のファイル操作処理

AnT のファイル操作処理の様子を**図 4** に示し，以下に説明する [10]． ファイル操作処理に関連する OS サーバは，3 種類である．ファイル管理サーバ（FS）は，i ノードの管理を行う．ブロック管理サーバ（BLK）は，ファイルキャッシュの管理を行う．ディスクドライバサーバ（DK）は，外部記憶装置の管理を行う．

AP プロセスがファイル参照を行う場合を例に，ファイル操作処理の処理流れを説明する．ファイル操作を行う AP プロセスは，FS に対してファイルの読み込みを同期依頼する．FS は，BLK に対して当該ファイルに対応するデータブロックの読み込みを非同期依頼する．BLK は，当該データブロックをファイルキャッシュから探索し，AP プロセスに対して当該データブロックを返却する．ここで，同期依頼と非同期依頼は，それぞれ依頼の結果を受け取るまで依頼元プロセスの実行がブロックされる依頼方式とブロックされない依頼方式である．なお，ファイルキャッシュに当該データブロックが存在しない場合，BLK は，DK に対して当該データブロックの読み込みを同期依頼する．

評価では，ファイルデータが全てブロックキャッシュに存在する場合，つまり DK への処理依頼の生じない場合ついて評価したため，以降では，FS と BLK に着目する．

表 1 ファイル参照のインタフェース

機能	形式
データの参照（ブロック単位）	readblock(fd, size, blkoff); fd: ファイル識別子 size: データのサイズ（ブロック単位） blkoff: 参照開始位置（ブロック単位）
データの参照（バイト単位）	readbyte(fd, size, offset); fd: ファイル識別子 size: データのサイズ（バイト単位） offset: 参照開始位置（バイト単位）

3.3 Bonnie

Bonnie のパラメータとして，プロセス数を 1〜16，ファイルサイズを 128KB ，参照回数を 4,000 回，更新確率を 10%として評価した．

各パラメータについて，プロセス数は，最大数を測定計算機のコア数としている．ファイルサイズは，ファイルキャッシュに必ずヒットするサイズとした．参照回数と更新確率は，Bonnie の初期値を利用した．

また，***AnT*** は，ファイル参照のインタフェース（**表 1**）として，readblock（システムコール）と readbyte（ライブラリコール）を有する [10]．そこで，ファイルの参照時に，readblock を用いる様に変更した Bonnie と readbyte を用いる様に変更した Bonnie について評価した．ここで，readbyte は，データブロックのバッファリングを行うインタフェースであり，ファイルの各データブロックについて，初参照時にのみ，OS サーバに処理依頼を行う．一方，readblock は，データブロックのバッファリングを行わず，ファイルの各データブロックについて，参照時に毎回，OS サーバに処理依頼を行う．

3.4 評価項目

以下の分散形態について，Bonnie による性能評価を行った．

(1) OS 処理のみ分散

OS サーバ（FS と BLK）の分散による分散効果を明らかにするため，AP プロセスをコア 0 に固定し，OS サーバのみを分散する分散形態である．

(2) AP 処理と OS 処理を分散

OS 処理だけでなく AP 処理も分散する場合の分散効果について明らかにするため，OS サーバと AP プロセスの両方を分散する分散形態である．

また，Linux との比較として，OS 構造の違いによる分散効果の差について明らかにするため，モノリシックカーネル構造 OS である Linux 3.16 と比較した．

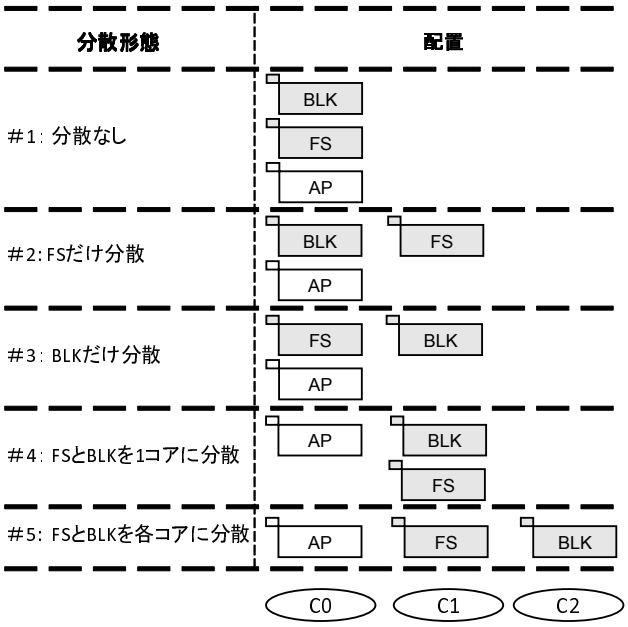

図 5 分散形態（OS 処理のみ分散）

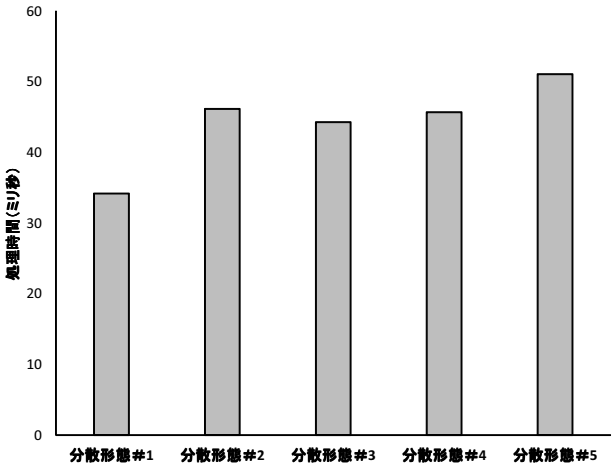

図 6 OS 処理のみ分散 (readblock)（ベンチマークプロセス数 1）

3.5 OS 処理のみ分散

OS 処理のみを分散する場合の分散効果について，評価した分散形態を**図 5** に示す．なお，図 5 において，C0，C1，および C2 は CPU の各コアを表している．評価した分散形態は，AP プロセスをコア 0 に固定した場合に考えられる全ての分散形態である．ただし，DK への処理依頼の生じない場合についての評価であるため，DK の分散については考慮していない．

測定結果について，データブロックのバッファリングを行わないインタフェース（readblock）を用いる場合の Bonnie の測定結果を**図 6** と**図 7** に示し，データブロックのバッファリングを行うインタフェース（readbyte）を用いる場合の Bonnie の測定結果を**図 8** と**図 9** に示す．また，測定結果に対する考察を以下に示す．

(1) ベンチマークプロセス数が 1 の場合，OS 処理の分散

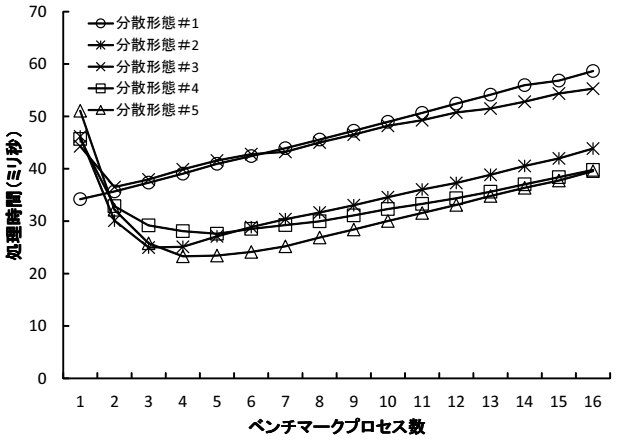

図 7 OS 処理のみ分散 (readblock)

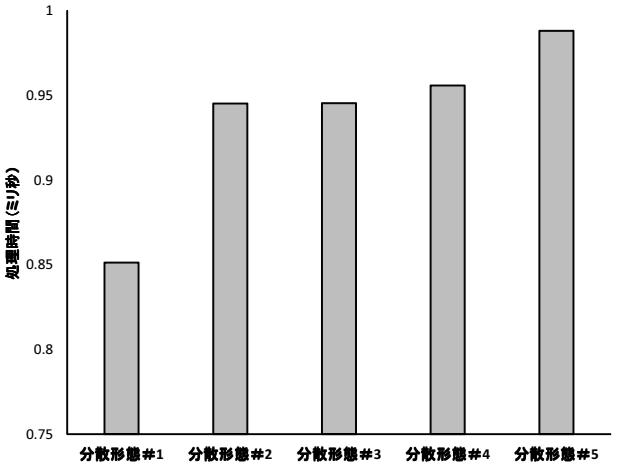

図 8 OS 処理のみ分散 (readbyte)（ベンチマークプロセス数 1）

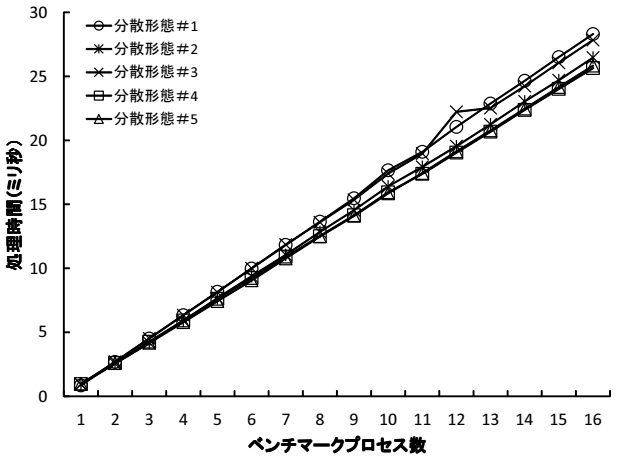

図 9 OS 処理のみ分散 (readbyte)

で性能が低下している．図 6 と図 8 について，分散形態＃ 1 が最も処理時間が短い．次に処理時間短いのは，分散形態＃ 2，分散形態＃ 3，および分散形態＃ 4 であり，分散形態＃ 5 は最も処理時間が長い．これは，並列実行可能な処理が存在しないため OS 処理の分散には効果がなく，

OS 処理を分散したことにより生じるコア間通信によって処理時間が長大化しているためである．具体的には，ベンチマークプロセスの処理依頼 1 回につき，分散形態＃ 1 の場合 1 回，分散形態＃ 2，分散形態＃ 3，および分散形態＃ 4 の場合 2 回，分散形態＃ 5 の場合 3 回のコア間通信が発生する．

(2) ベンチマークプロセス数が 2 以上の場合，OS 処理の分散で性能が向上している．図 7 と図 9 について，ベンチマークプロセス数が 2 以上の場合，分散形態＃ 2，分散形態＃ 3，分散形態＃ 4，および分散形態＃ 5 の処理時間は，分散形態＃ 1 に比べ短い．これは，分散形態＃ 2，分散形態＃ 3，分散形態＃ 4，および分散形態＃ 5 の場合，複数のベンチマークプロセスの処理を並列化できるためである．一方，分散形態＃ 1 の場合，各ベンチマークプロセスの処理は，他のベンチマークプロセスの処理依頼にはじまる一連の処理の完了を待つ必要がある．

(3) 分散形態＃ 3 は分散効果が小さい．図 7 と図 9 について，分散形態＃ 3 の処理時間は，分散形態＃ 1 と同程度である．つまり，BLK だけを分散することの分散効果は小さい．これは，BLK の処理時間が FS の処理時間に対して十分に短いため，BLK に依頼が複数溜まらないことに起因する．また，BLK に依頼が複数溜まらない場合，FS から BLK への処理依頼の際に BLK は WAIT 状態であり，処理依頼に BLK を起床させるためのコア間通信を伴う．ここで，ベンチマークプロセスの処理と BLK の処理を並列に実行できることによる処理時間の短縮とコア間通信オーバヘッドの差分が分散形態＃ 1 との処理時間の差になる．

(4) 図 7 と図 9 から，分散形態＃ 2，分散形態＃ 4，および分散形態＃ 5 の処理時間は，分散形態＃ 1 に比べ短い．これは，FS の処理時間がベンチマークプロセスと BLK の処理時間に対して，長いことに起因する．分散する OS サーバの処理時間が長い場合，並列化できる処理は多い．また，FS の処理時間に対してベンチマークプロセスの処理時間が十分に短い場合，FS に依頼が複数溜まるからである．

(5) 図 7 と図 9 から，ベンチマークプロセス数が 16 以上の場合，分散形態＃ 4 ≥ 分散形態＃ 5 ≥ 分散形態＃ 2 の順で性能が高い．これは，ベンチマークプロセスによる FS への処理依頼の頻度とコア間通信回数に起因する．分散形態＃ 4 は FS から BLK への処理依頼の際に，コア間通信を必要としない．一方，分散形態＃ 5 は毎回コア間通信を必要とする．したがって，コア間通信回数の差により，分散形態＃ 4 の処理時間が短くなる．次に，分散形態＃ 5 と分散形態＃ 2 について，ベンチマークプロセス数が多い場合，FS への処理依頼の頻度は，分散形態＃ 5 の方が高くなる．これは，分散形態＃ 2 では，ベンチマークプロセスと BLK が同一コア上で走行し，BLK の方が優先度が高いため，BLK の処理中は，ベンチマークプロセスが処理依頼を行えないためである．したがって，分散形態＃ 2 より分散形態＃ 5 の処理時間が短くなる．

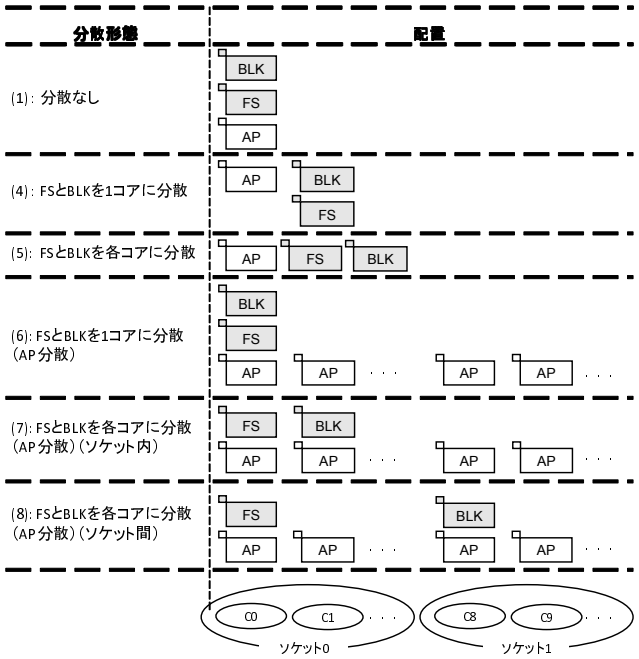

図 10 分散形態（AP 処理と OS 処理を分散）

(6) readblock を用いる場合に比べ，readbyte を用いる場合は OS 処理の分散効果は小さい．これは，readbyte を用いる場合では，既にバッファリングされているデータブロックの参照時に FS への処理依頼を必要とせず，OS サーバへの処理依頼の回数が少なくなるためである．図 7 と図 9 について，readblock を用いる場合において最も分散効果の高い分散形態＃ 5 では，分散形態＃ 1 に比べ処理時間を最大約 19.6 ミリ秒できる（約 35%の処理時間を短縮）．一方で，readbyte を用いる場合において最も分散効果の高い分散形態＃ 4 では，分散形態＃ 1 に比べ処理時間を最大約 2.7 ミリ秒短縮できる（約 5%の処理時間を短縮）．

3.6 AP 処理と OS 処理を分散

AP 処理と OS 処理を分散する場合について，評価した分散形態を**図 10** に示す．なお，図 10 において，C0，C1，C8，および C9 は CPU の各コアを表している．評価した分散形態について，AP プロセスの分散時には，各コアの AP プロセス数が全コアで均等になるように配置し，AP プロセス数と OS サーバでの利用コア数の和がコア数より少ない場合には，OS サーバの配置コアに AP を配置しないように分散した．なお，図 10 において，分散形態＃ 2 と分散形態＃ 3 に相当する分散形態がない理由は，AP を分散する場合，これらは，分散形態＃ 5 と同じ分散形態であるためである．また，分散形態＃ 5 に相当する分散形態について，FS と BLK の両方を 1 つの CPU ソケット内に配置する分散形態（分散形態＃ 7）と FS を 1 つの CPU ソケット内に配置し，BLK をもう一方の CPU ソケットに配置した分散形態（分散形態＃ 8）を評価した．

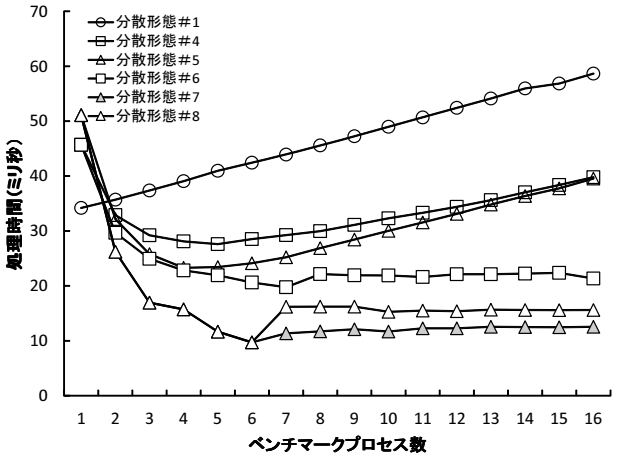

図 11 AP 処理と OS 処理を分散 (readblock)

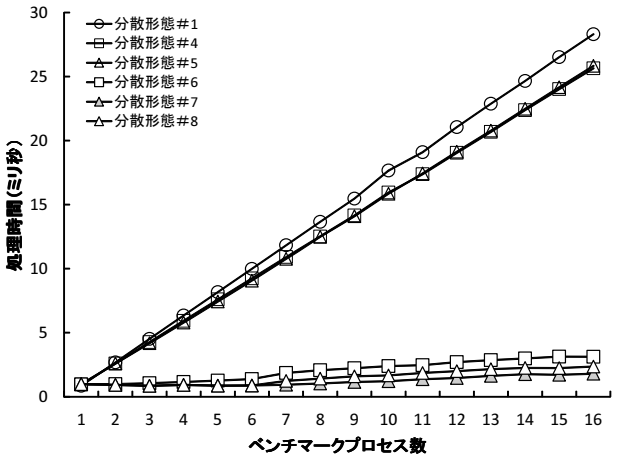

図 12 AP 処理と OS 処理を分散 (readbyte)

測定結果について，readblock を用いる場合の Bonnie の測定結果を**図 11** に示し，readbyte を用いる場合の Bonnie 測定結果を**図 12** に示す．また，測定結果に対する考察を以下に示す．

(1) 図 11 と図 12 から，ベンチマークプロセス数が 2 以上の場合，分散形態＃ 7 ≥ 分散形態＃ 8 ≥ 分散形態＃ 6 の順で性能が高い．分散形態＃ 6 は，AP 処理と OS 処理を分散する分散形態の中では最も処理時間が長いものの，OS 処理のみを分散する分散形態よりも処理時間が短い．これは，分散形態＃ 6 では，FS と BLK が同一のコア上で同一の優先度で走行するため，各ベンチマークプロセスによる処理依頼の全てを FS が完了するまで BLK が走行できず，ベンチマークプロセスへの結果返却が遅くなる．この結果，各ベンチマークプロセスが次の処理依頼を行うまでの時間が長くなる．一方で，分散形態＃ 7 と分散形態＃ 8 では，FS と BLK が異なるコア上で走行する．このため，各ベンチマークプロセスに対する FS の処理が完了すると直ちに，BLK が走行する．この結果，各ベンチマークプロセスが次の処理依頼を行うまでの時間は長大化せず，分散形態＃ 6 に比べ処理依頼の頻度が高い．また，分散形態＃ 7 と分散形態＃ 8 について，分散形態＃ 7 の処理時間が短い．これは，FS と BLK は密に連携することと，CPU ソケット間での通信オーバヘッドが大きい [11] ことに起因する．

(2) readblock を用いる場合に比べ，readbyte を用いる場合は AP 処理の分散効果が大きい．これは，readbyte を用いる場合，データブロックのバッファリングが行われるため，readblock を用いる場合に比べ AP 処理が多く OS 処理が少ないためである．図 11 と図 12 について，readblock を用いる場合において最も分散効果の大きい分散形態＃ 7 では，分散形態＃ 1 に比べ処理時間を最大約 46.1 ミリ秒短縮できる（約 79%の処理時間を短縮）．一方で，readbyte を用いるにおいて最も分散効果の大きい分散形態＃ 7 では，分散形態＃ 1 に比べ処理時間を最大約 26.1 ミリ秒短縮できる（約 94%の処理時間を短縮）．

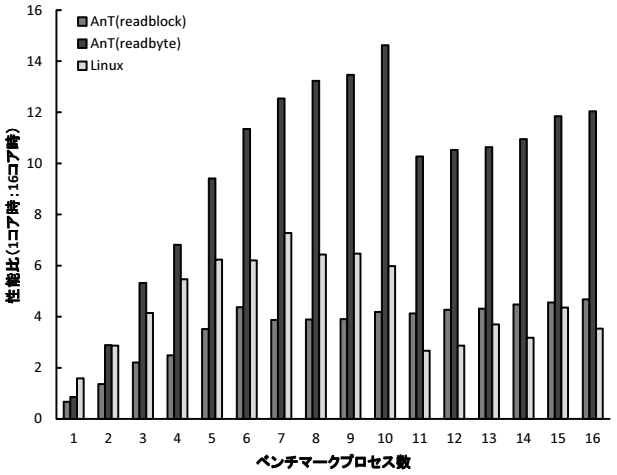

図 13 Linux 3.16 と ***AnT*** の性能向上比の比較

3.7 Linux との比較

比較の尺度は，1 コア時の処理時間に対する 16 コア時の処理時間の比とした．具体的には，性能比 ＝ 1 コア時の処理時間 ÷ 16 コア時の処理時間 である．なお，***AnT*** の分散形態は，最も分散効果の高い分散形態＃ 7 とした．

Linux 3.16 と ***AnT*** について，1 コア時と 16 コア時の性能比の比較を**図 13** に示す．測定結果から以下のことが分かる．

AnT(readbyte) の性能向上比は高い．図 13 について，ベンチマークプロセス数が 1 以外の場合について，***AnT***(readbyte) の性能向上比が最も高い．これは，readbyte がデータブロックのバッファリングを行うインタフェースであることに起因する．readbyte の場合，各ベンチマークプロセスは，それぞれデータブロックのバッファを持ち，バッファにヒットしない場合のみ FS へ処理依頼を行う．ここで，逐次化される処理（FS の処理と BLK の

処理）の実行回数が少なくなるため，並列性が向上し，コア数を増加させた際の性能向上が大きくなる．

4. おわりに

AnT オペレーティングシステムについて，Bonnie ベンチマークを利用し，ファイル操作処理における AP 処理と OS 処理の分散効果を報告した．

OS 処理のみを分散する場合，ベンチマークプロセス数が 2 以上の時，処理時間を短縮できる．具体的には，ブロック単位のデータ参照を行うインタフェース（readblock）を用いる場合，ファイル管理サーバ（FS）とブロック管理サーバ（BLK）を異なるコアに配置する条件下での分散形態（分散形態＃ 5）では，分散しない場合に比べ処理時間を最大約 19.6 ミリ秒（約 35%）を短縮できる．バイト単位のデータ参照を行うインタフェース（readbyte）を用いる場合，FS と BLK を同じコアに配置する条件下での分散形態（分散形態＃ 4）は，分散しない場合に比べ処理時間を最大約 2.7 ミリ秒（約 5%）短縮できる．また，ベンチマークプロセス数 16 以上の場合には，分散形態＃ 4 の性能が最も高い．

AP 処理と OS 処理を分散する場合，OS 処理のみを分散する場合よりも処理時間を短縮できる．具体的には，readblock を利用する場合，FS と BLK を同じ CPU ソケット内の異なるコアに配置する条件下での分散形態（分散形態＃ 7）は，分散しない場合に比べ処理時間を最大約 46.1 ミリ秒（約 79%）短縮できる．readbyte を利用する場合，分散形態＃ 7 では，分散しない場合に比べ処理時間を最大約 26.5 ミリ秒（約 94%）短縮できる．また，分散形態＃ 7 の性能が最も高い．

残された課題として，他の OS との分散効果の詳細な比較がある．

参考文献

[1] Anderson, T.E.: The performance of spin lock alternatives for shared-memory multiprocessors, *IEEE Trans. Parallel Distrib. Syst.*, Vol.1, pp.6-16 (1990).

[2] Silas Boyd-Wickizer, M. Frans Kaashoek, Robert Morris and Nickolai Zeldovich: Non-scalable locks are dangerous, *In Proceedings of the Linux Symposium* (2012).

[3] Liedtke, J. : Toward Real Microkernels, *Communications of The ACM*, Vol.39, pp.70-77 (1996).

[4] 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).

[5] Liedtke, J. : Improving IPC by kernel design, *Proceedings of the fourteenth ACM symposium on Operating systems principles*, Vol.27, pp.175-188 (1993).

[6] 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，Vol.J93-D，No.10，pp.1977-1989 (2010).

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

[8] 河上 裕太, 山内 利宏, 谷口 秀夫 : ***AnT*** オペレーティングシステムにおける効率的なサーバ間通信機構, 情報処理学会研究報告, Vol.2015-OS-132, No.12, pp.1-7 (2015).

[9] Bray, T.: *The Bonnie home page* (online), available from ⟨http://www.textuality.com/bonnie⟩ (accessed 2015-8-24).

[10] 枡田圭祐, 谷口秀夫: プロセスとファイルキャッシュを共有するオンメモリファイル機能の提案，コンピュータシステム・シンポジウム論文集，Vol.2012，pp.25-32 (2012).

[11] Tudor David, Rachid Guerraoui and Vasileios Trigonakis: Everything you always wanted to know about synchronization but were afraid to ask, *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pp.33-48 (2013).