Vol.2016-OS-138 No.4
2016/8/8

マルチコア環境における優先度逆転を抑制する *AnT* のスケジュール機構の評価

鴨生 悠冬[1]　山内 利宏[1]　谷口 秀夫[1]

概要：マルチコアプロセッサの普及により，サービスが複雑化し，優先度逆転が多発する．特に，マイクロカーネル OS では，AP プロセスだけでなく，OS サーバも優先度逆転の影響を受けるため，優先度逆転を抑制することが重要である．また，マルチコア環境において，コア毎に独立したスケジュール機能は低オーバヘッドな利点を有する．しかし，このスケジュール機能を利用する場合，優先度逆転の抑制にコア間通信を必要とし，オーバヘッドが大きい．そこで，マイクロカーネル OS である ***AnT*** において，マルチコア環境で優先度逆転を抑制し，かつコア間通信回数を削減するスケジュール機構を提案した．本稿では，このスケジュール機構の評価結果を報告する．

1. はじめに

サービスやシステムはマルチコアプロセッサを利用するためにますます複雑になっており，優先度逆転 [1] が多発する．特に，マイクロカーネル構造 OS[2][3][4] は OS 機能をプロセス（以降，OS サーバ）として実現しており，OS 機能を実現する処理にも優先度逆転が生じる．したがって，マイクロカーネル構造 OS では，AP プロセスだけでなく OS サーバも含めた優先度逆転の抑制が重要である．

マルチコア環境において，コア毎に独立したスケジュール機能は，スケジューラによる他コア上のプロセス情報の操作を不可とすることで排他制御を不要とするため，低オーバヘッドである．しかし，優先度逆転を抑制する場合，他コア上のプロセス情報を操作する必要があるため，コア間通信によるスケジューラ間の連携が必要となる．コア間通信のオーバヘッドは大きいため，優先度逆転の抑制におけるオーバヘッド削減には，コア間通信回数の削減が重要である．

そこで，我々は，マルチコア環境に対応したマイクロカーネル構造 OS である ***AnT*** オペレーティングシステム（An operating system with adaptability and toughness）（以降，***AnT***）[5] において，サーバプログラム間通信での優先度継承によって優先度逆転を抑制し，かつコア間通信回数を削減するスケジュール機構を提案した [6]．

本稿では，提案手法の評価を行い，優先度逆転の抑制効果とコア間通信回数の削減による効果を明らかにする．

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

2. マルチコア環境における優先度逆転抑制法

2.1 *AnT* オペレーティングシステム

AnT は，マイクロカーネル構造を有する OS であり，マルチコア環境で動作する．***AnT*** は，各コアの独立動作（方針 1）を実現するために，カーネルをコア毎に配置する．また，各カーネルの軽量化（方針 2）を実現するために，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，マイクロカーネルに必要な全機能を有する．一方，p-カーネルは，m-カーネルが制御するコア以外のコアを制御し，スケジューラ，サーバプログラム間通信機能，およびコア間通信制御機能を有する．さらに，処理の高速化（方針 3）を実現するために，高速なサーバプログラム間通信機構を実現している．

AnT の高速なサーバプログラム間通信 [7][8] の基本機構を**図 1** に示す．この機構は，制御用情報とデータ情報を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，プロセス間でデータ複写レスでの通信を実現している．ここで，コア間通信データ域の「コア」は，CPU コアではなく 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を（アンマップ）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

© 2016 Information Processing Society of Japan

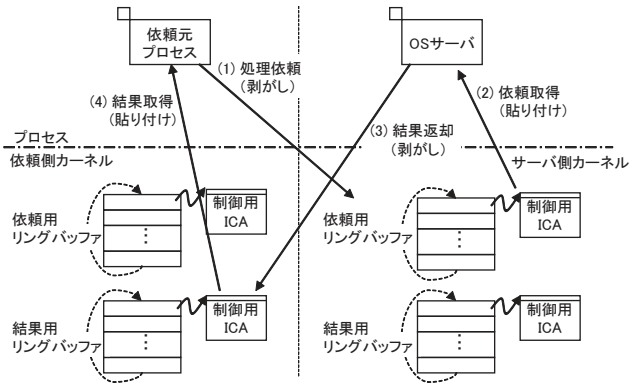

図 1 サーバプログラム間通信の基本機構

セスの仮想空間へ貼り付けることで行われる．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用リングバッファと結果用リングバッファを持つ．マルチコア環境では，これらのリングバッファをコア毎に用意することで排他制御オーバヘッドを削減している．さらに，同期型と非同期型の通信インタフェースを同様な形式で提供し，両インタフェースを選択して利用できる．基本的な通信の流れを以下に説明する．

（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）は依頼先 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，依頼先 OS サーバが WAIT 状態の場合，依頼先 OS サーバを起床させる．

（2）依頼先 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，依頼先 OS サーバに制御用 ICA を貼り付ける．依頼先 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）依頼先 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，依頼元プロセスが WAIT 状態の場合，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．

異なるコア上のプロセス間でサーバプログラム間通信を行う場合，プロセスを起床させる処理においてコア間通信を必要とする．コア間通信は，コア間で共有している領域（以降，共有領域）を利用し通信内容の授受を行う．また，通信内容の登録を通知するために IPI (Inter-Processor Interrupt) を利用する．共有領域は，N × N（N はコア数）の配列であり，発行側コア（i）と受理側コア（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通信内容を授受する．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処理制御を不要としている．

AnT は，1 つのコアの処理がネックにならないようにして処理全体を高速化するために各コアの独立動作（方針 1）を設計方針としており，各コアにスケジューラを配置している．スケジューラは，自コア上のプロセスのみを制御し，他コア上のプロセス情報を確認するだけで操作しない．これにより，スケジューラ間の排他制御を不要にしている．

2.2 マルチコア環境における優先度逆転抑制法

2.2.1 優先度逆転抑制

文献 [6] で提案したマルチコア環境でのサーバプログラム間通信における優先度逆転抑制法を説明する．提案手法では，異なるコア上のプロセス間のサーバプログラム間通信において生じる優先度逆転を抑制するために，以下の 2 つの対処を行う．

（対処 1）サーバプログラム間通信時に授受する制御用 ICA に依頼元 AP プロセスの優先度（以降，ICA 優先度）を保持させ，ICA 優先度が高い順に依頼を取得する．

（対処 2）OS サーバの依頼取得時または OS サーバへの依頼登録時に OS サーバの優先度を ICA 優先度に変更する．

上記 2 つの対処を行った優先度逆転を抑制する制御法として，可変優先度取得時変更法（VPGET）と可変優先度登録時変更法（VPSET）が考えられる．VPGET は，依頼取得を ICA 優先度順に行うとともに，依頼取得時に OS サーバの優先度を ICA 優先度に変更する制御法である．VPSET は，依頼取得を ICA 優先度順に行うとともに，依頼登録時と依頼取得時に OS サーバの優先度を ICA 優先度に変更する制御法である．ただし，依頼登録時の OS サーバの優先度変更は，ICA 優先度が OS サーバの優先度よりも高い場合に行う．これは，低優先度の AP プロセスの処理依頼によって，OS サーバの優先度が下がり優先度逆転が生じることを防ぐためである．また，依頼取得時の OS サーバの優先度変更は，ICA 優先度が OS サーバの優先度よりも低い場合に行う．これは，OS サーバの優先度を低下させる契機として OS サーバの優先度変更を行うためである．

2.2.2 コア間通信回数の削減

提案手法では，オーバヘッドを抑制するために，コア間通信回数を削減する．本項では，提案手法のコア間通信回数の削減方法について説明する．サーバプログラム間通信の処理依頼/結果返却において，依頼先/依頼元プロセスを起床させる，または優先度を継承する場合，他コア上のプ

© 2016 Information Processing Society of Japan

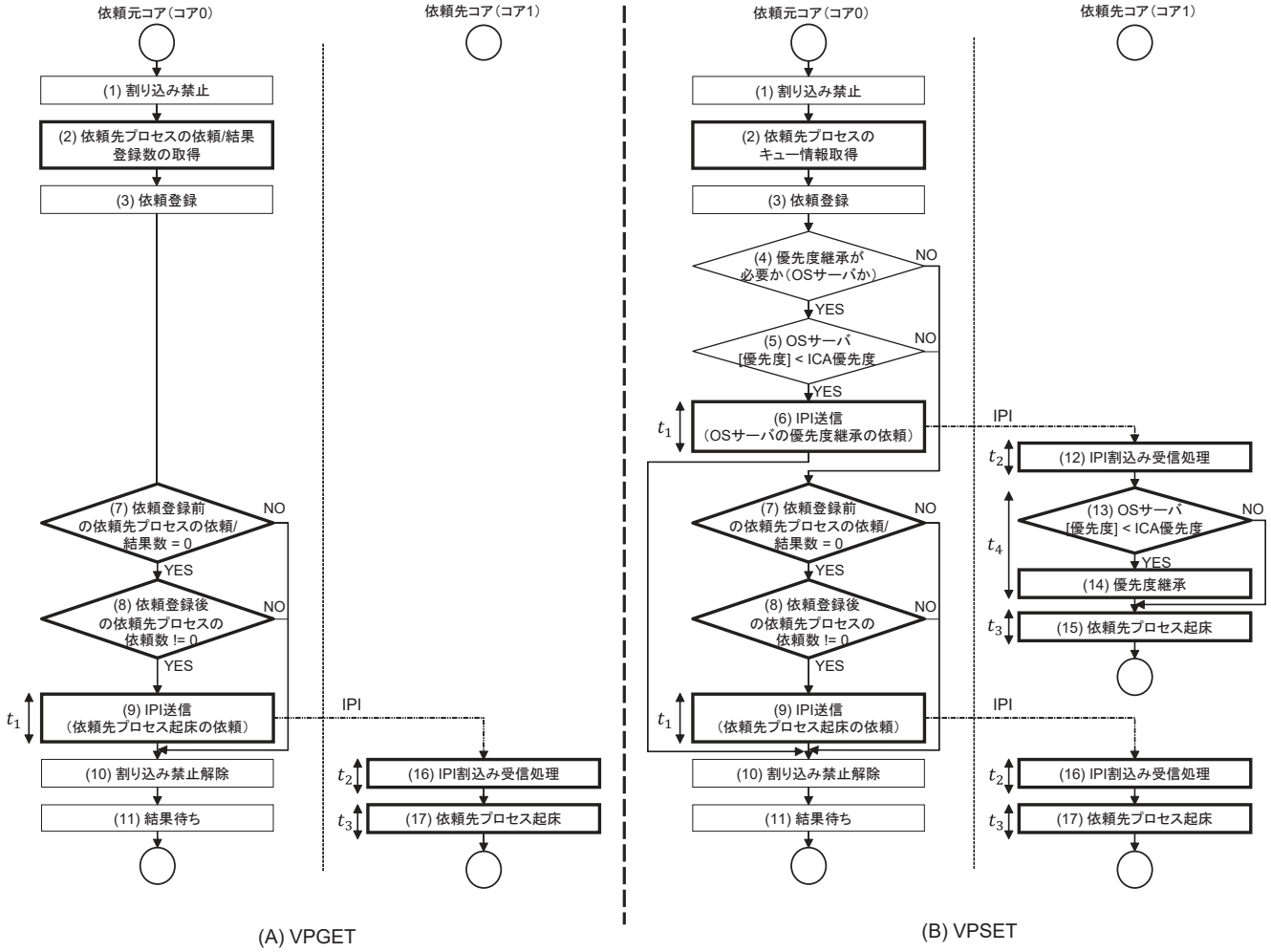

図 2 異なるコア上のプロセス間のサーバプログラム間通信における処理依頼の処理流れ

ロセス情報を変更する必要がある．コア毎に独立したスケジュール機能は，他コアのプロセス情報を変更できないため，コア間通信を行い，他コアのスケジューラへプロセス情報の変更を依頼する．以降では，依頼先プロセスを起床させるコア間通信（起床コア間通信）と優先度を継承するためのコア間通信（継承コア間通信）の回数を削減する方法について説明する．

提案手法の異なるコア上のプロセス間のサーバプログラム間通信における処理依頼の処理流れを**図 2** に示す．VPGET を例に起床コア間通信回数の削減について説明する．シングルコア環境では，依頼先プロセスの起床判定に依頼先プロセスの実行状態を利用できる．しかし，マルチコア環境では，異なるコア上の依頼先プロセスの起床判定に依頼先プロセスの実行状態を利用できない．これは，依頼元プロセスによる依頼先プロセスの実行状態の確認後，依頼先コアにより依頼先プロセスの実行状態が変更され，依頼先プロセスを起床させるべき際に起床させられない可能性があるためである．このため，コア間通信を行い，依頼先コアのスケジューラへ依頼先 OS サーバの起床を依頼する必要がある．しかし，処理依頼時に常に起床コア間通信を行うとオーバヘッドが大きい．

そこで，提案手法では，異なるコア上の依頼先プロセスの起床条件として，依頼先プロセスの依頼キュー/結果キューに登録されている依頼数/結果数を利用する (7)(8)．これらの情報を利用することで，依頼先コアの状態が変わっても依頼先プロセスの起床が不要な場合を判定できる．ここで，依頼登録前の依頼キューに繋がる依頼数と結果キューに繋がる結果数をそれぞれ Req_{prev}，Ret_{prev}，依頼登録後の依頼キューに繋がる依頼数を $Req_{currnent}$ とする．以下の全ての条件（以降，起床条件）を満たす場合，依頼先プロセスを起床させる．

（条件 1）$Req_{prev} = 0$
（条件 2）$Ret_{prev} = 0$
（条件 3）$Req_{currnent} \neq 0$

（条件 1）を満たさない場合，他のプロセスの依頼登録において（条件 1）が満たされるため，他のプロセスが依頼先プロセスを起床させている．（条件 2）を満たさない場合，他のプロセスの結果返却において結果返却時の（条件

© 2016 Information Processing Society of Japan

表 1 VPGET と VPSET における優先度逆転時間とオーバヘッドの関係

通番	起床	継承	AP_r と OS	AP_o と OS	VPGET		VPSET		t_{inv}+オーバヘッド	
					t_{inv}	オーバヘッド	t_{inv}	オーバヘッド	VPGET	VPSET
(1)	無	無	$AP_r \leq OS$		0	0	0	0	0	0
(2)		有	$AP_r > OS$	$AP_o < OS$	0			$t_1+t_2+t_3$	0	$t_1+t_2+t_3$
(3)				$AP_o > OS$	> 0			$+t_4$	t_{inv}	$+t_4$
(4)	有	無	$AP_r \leq OS$		0	$t_1+t_2+t_3$		$t_1+t_2+t_3$	$t_1+t_2+t_3$	$t_1+t_2+t_3$
(5)		有	$AP_r > OS$	$AP_o < OS$	0			$t_1+t_2+t_3$	$t_1+t_2+t_3$	$t_1+t_2+t_3$
(6)				$AP_o > OS$	> 0			$+t_4$	$t_1+t_2+t_3+t_{inv}$	$+t_4$

1）が満たされるため，他のプロセスが依頼先プロセスを起床させている．（条件 3）を満たさない場合，依頼元プロセスの依頼登録から登録後の依頼数の確認までの間に，依頼先プロセスは依頼を取得している．すなわち，依頼先プロセスは依頼処理を実行中である．

同様に結果返却時の依頼元プロセスの起床条件を以下に示す．なお，結果返却後の結果キューに繋がる結果数を $Ret_{currnent}$ とする．以下の全ての条件を満たす場合，依頼元プロセスを起床させる．

（条件 1）$Ret_{prev} = 0$
（条件 2）$Req_{prev} = 0$
（条件 3）$Ret_{currnent} \neq 0$

VPSET におけるコア間通信回数の削減について説明する．VPSET は，依頼登録時に優先度継承を行い，依頼先プロセスの優先度を変更する．コア毎に独立したスケジュール機能は他コア上のプロセス情報を変更できない．したがって，サーバプログラム間通信の処理依頼において，依頼元コアのスケジューラは，コア間通信によって，依頼先コアのスケジューラへ優先度継承を依頼する (6)．

ここで，依頼登録時に依頼先プロセスの起床依頼を行うことに着目する．依頼先プロセスの起床依頼は，優先度継承と同じくコア間通信を必要とする (9)．そこで，優先度継承が必要な場合，優先度継承とプロセス起床の 2 つの処理を 1 つのコア間通信によって依頼する (13)(14)(15)．優先度継承が不要な場合（(4) または (5) が偽），依頼先プロセスが起床条件を満たすか確認し (7)，起床条件を満たす場合，処理 (9) を行う．これにより，(6)(9) いずれか 1 回のコア間通信となり，コア間通信回数を最大 1 回に抑制できる．

VPSET は，依頼登録時に OS サーバの優先度が ICA 優先度より低いこと（以降，継承条件）を確認する（図 2(B)(5)）．OS サーバの優先度が ICA 優先度より低い場合，コア間通信によって，依頼先コアのスケジューラに優先度継承を依頼する．このとき，依頼先コアでも再度，OS サーバの優先度が ICA 優先度より高いことを確認する（図 2(B)(13)）．これは，複数のコアが同時にコア間通信による優先度継承を行った場合，優先度逆転が生じる可能性があるためである．

2.2.3 優先度逆転とオーバヘッド

VPGET と VPSET における優先度逆転時間とオーバヘッドの関係を**表 1** に示す．以降，依頼元プロセスの優先度を AP_r，依頼先 OS サーバの優先度を OS，依頼先コア上の RUN/READY 状態の AP プロセスの優先度を AP_o，優先度逆転時間を t_{inv} とする．

優先度逆転について，VPGET では発生するが，VPSET では発生しないことが分かる．優先度逆転の発生条件は，$AP_r > OS$ かつ $AP_o > OS$（以降, 逆転条件）である．オーバヘッドについて, 継承条件を満たさない場合（表 1(1)(4)），VPGET と VPSET の差はない．一方, 継承条件を満たす場合（表 1(2)(3)(5)(6)），VPSET のオーバヘッドは，VPGET より以下の値だけ大きくなる．

（起床無）$t_1+t_2+t_3+t_4$
（起床有）t_4

優先度逆転とオーバヘッドによる OS サーバの実行開始について, 継承条件を満たさない場合（表 1(1)(4)），VPGET と VPSET の差はない．一方，継承条件を満たす場合（表 1(2)(3)(5)(6)），以下に示す差が生じる．

（1）起床無，かつ $AP_o < OS$（表 1(2)）
VPSET は，VPGET より $t_1+t_2+t_3+t_4$ だけ遅れる
（2）起床無，かつ $AP_o > OS$（表 1(3)）
VPGET は，VPSET より $t_{inv}-(t_1+t_2+t_3+t_4)$ だけ遅れる
（3）起床有，かつ $AP_o < OS$（表 1(5)）
VPSET は，VPGET より t_4 だけ遅れる
（4）起床有，かつ $AP_o > OS$（表 1(6)）
VPGET は，VPSET より $t_{inv}-t_4$ だけ遅れる

3. 評価

3.1 基本評価

3.1.1 観点と評価内容

実環境における VPGET と VPSET の高優先度 AP プロセスの処理依頼への影響を明らかにするため，表 1 における t_{inv} とオーバヘッドの実測値を求める．また，コア間通信回数の削減による効果を明らかにするため，コア間通信処理における依頼処理（依頼先 OS サーバの起床や優先度継承）以外の処理時間を求める．測定は，VPGET と VPSET を ***AnT*** に実現し，Intel Xeon E5-2630 v3（8Core,

© 2016 Information Processing Society of Japan

表 2 優先度逆転時間とオーバヘッドの測定結果

通番	起床	継承	AP_r と OS	AP_o と OS	VPGET		VPSET		t_{inv}+オーバヘッド	
					t_{inv}	オーバヘッド	t_{inv}	オーバヘッド	VPGET	VPSET
(1)	無	無	$AP_r \leq OS$		0 μs	0 μs	0 μs	0 μs	0 μs	0 μs
(2)		有	$AP_r > OS$	$AP_o < OS$	0 μs	0 μs	0 μs	0.87 μs	0 μs	0.87 μs
(3)				$AP_o > OS$	$>$ 0(実測値:100.41 μs)	0 μs	0 μs	0.90 μs	100.41 μs	0.90 μs
(4)	有	無	$AP_r \leq OS$		0 μs	0.87 μs	0 μs	0.87 μs	0.87 μs	0.87 μs
(5)		有	$AP_r > OS$	$AP_o < OS$	0 μs	0.89 μs	0 μs	1.09 μs	0.89 μs	1.09 μs
(6)				$AP_o > OS$	$>$ 0(実測値:100.42 μs)	0.95 μs	0 μs	1.00 μs	101.37 μs	1.00 μs

2.4GHz）の計算機で行った．

基本測定の様子を**図 3** に示す．基本測定は，異なるコア上に存在する依頼元プロセス (APr) と OS サーバでサーバプログラム間通信を行い，OS サーバにおいて生じる優先度逆転時間，コア間通信オーバヘッド，同期処理依頼から結果返却までの時間（応答時間），および図 2(B)(6)〜(15) のコア間通信処理時間を測定する．OS サーバと同一コア上の AP プロセス (APo) は，PU 処理を 100 μs 行う．OS サーバは依頼を取得した後，依頼処理として PU 処理を 100 μs 行い，結果を返却する．

APr の優先度（AP_r），OS サーバの優先度（OS），および APo の優先度（AP_o），ならびに起床条件を満たすか否かの組み合わせは，表 1 に従う．起床条件を満たす場合と満たさない場合において，OS サーバの依頼数 (n) をそれぞれ 1 と 0 として測定した．測定結果は 5 回試行した場合の平均時間である．

3.1.2 測定結果と考察

優先度逆転時間とコア間通信オーバヘッドの測定結果を**表 2** に示す．測定結果より，以下のことが分かる．

（1）逆転条件を満たす場合（表 2(3)(6)），VPGET で生じる優先度逆転時間（t_{inv}）は，APo の PU 処理時間と同じである．この結果から，t_{inv} は，APo の PU 処理時間に依存すると考えられる．したがって，逆転条件を満たし，APo の PU 処理時間が VPGET のオーバヘッドに対する VPSET のオーバヘッドの増分より長い場合，VPSET は VPGET より応答時間が短いと考えられる．すなわち，表 2(3) の場合，APo の PU 処理時間が 0.90 μs（0.90 μs - 0 μs）より長ければ，VPSET は VPGET より応答時間が短いと考えられる．表 2(6) の場合，APo の PU 処理時間が 0.05 μs（1.00 μs - 0.95 μs）より長ければ，VPSET は VPGET より応答時間が短いと考えられる．

（2）全ての場合において，VPSET のオーバヘッドは，1.1 μs より短い．したがって，OS サーバの依頼処理時間が 150 μs 程度であっても，VPSET のオーバヘッドは 1%以下である．

（3）起床条件を満たさず継承条件を満たす場合（表 2(2)(3)），VPGET と VPSET の t_{inv}+オーバヘッドの大小関係は，AP_o と OS の大小関係によって異なる．$AP_o < OS$ の場合（表 2(2)），VPSET の t_{inv}+オーバヘッドは，VPGET より 0.87 μs（0.87 μs - 0 μs）長い．一方，$AP_o > OS$ の場合（表 2(3)），VPSET の t_{inv}+オーバヘッドは，VPGET より 99.51 μs（100.41 μs - 0.90 μs）短い．したがって，OS サーバへの処理依頼時に，$AP_o > OS$ であり，APo が RUN/READY 状態である確率が 1%以上であれば，VPSET の t_{inv}+オーバヘッドの平均は VPGET より短いと考えられる．

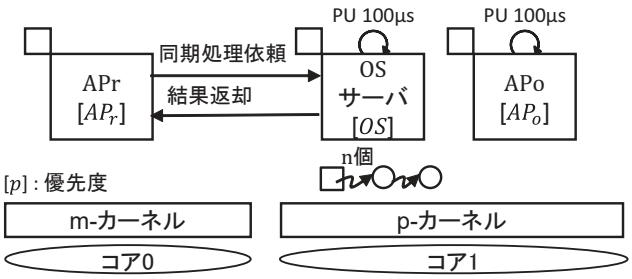

図 3 基本測定の様子

（4）起床条件を満たさず継承条件を満たす場合（表 2(2)(3)），VPGET のオーバヘッドは VPSET より約 0.9 μs 短い．すなわち，依頼先 OS サーバへの依頼頻度が低いまたは依頼先 OS サーバの処理時間が短い場合，VPGET を利用すると VPSET よりオーバヘッドによる影響を小さくできる．

（5）起床条件と継承条件を満たす場合（表 2(5)(6)），VPSET のオーバヘッド（$t_1 + t_2 + t_3 + t_4$）は，VPGET のオーバヘッド（$t_1 + t_2 + t_3$）より約 0.05〜0.2 μs 長い．すなわち，VPSET における優先度変更処理のオーバヘッドは，VPSET のオーバヘッドの約 5〜20%を占めると考えられる．

応答時間の測定結果を**表 3** に示す．制御を行わない，すなわち，コア間通信を行わない場合（表 3(1)）の VPSET の応答時間は 106.93 μs であるのに対し，制御を行う，すなわち，コア間通信を伴う場合（表 3(2)〜(6)）の VPSET の応答時間の最大値は 108.50 μs である．この結果から，VPSET の制御によるオーバヘッドは 1.57 μs（108.50 μs - 106.93 μs）であることが分かる．したがって，VPSET の制御によって増加する処理時間は，約 1.5%である．一方，制御を行わない，かつ逆転条件を満たさない場合（表

© 2016 Information Processing Society of Japan

表 3 応答時間の測定結果

通番	起床	継承	AP_r と OS	AP_o と OS	応答時間 (μs)	
					VPGET	VPSET
(1)	無	無	$AP_r \leq OS$		107.00	106.93
(2)		有	$AP_r > OS$	$AP_o < OS$	106.91	108.40
(3)				$AP_o > OS$	205.65	108.49
(4)	有	無	$AP_r \leq OS$		108.50	108.50
(5)		有	$AP_r > OS$	$AP_o < OS$	108.06	108.16
(6)				$AP_o > OS$	206.86	108.28

表 4 コア間通信処理時間とコア間通信個別処理時間

コア間通信処理時間	t_1	t_2	t_3	t_4
1.44 μs	0.22 μs	0.35 μs	0.36 μs	0.15 μs

3(1)(2)）の VPGET の応答時間の最小値 106.91 μs であるのに対し，制御を行い，かつ逆転条件を満たす場合の VPGET の応答時間は，206.86 μs である．この結果から，VPGET の制御によるオーバヘッドと優先度逆転時間の和は，99.95 μs であることが分かる．したがって，VPSET の制御と優先度逆転によって増加する処理時間は，約 94%である．

図 2(B)(6)〜(15) のコア間通信処理時間とコア間通信個別処理時間（t_1，t_2，t_3，および t_4）の測定結果を表 4 に示す．測定結果より，コア間通信処理における依頼処理（依頼先 OS サーバの起床や優先度変更）以外の処理時間は，0.93 μs（1.44 μs - (0.36 μs + 0.15 μs)）であることが分かる．すなわち，コア間通信回数を 1 回削減する毎に 0.93 μs のオーバヘッドを削減することができる．コア間通信を伴わないサーバプログラム間通信の処理時間が約 7 μs であることから，コア間通信回数の増加を 1 回抑制する毎に約 13%のオーバヘッドの増加を抑制できると考えられる．

3.2 サービス評価

3.2.1 観点と評価内容

高優先度サービスの処理を優先して実行できるかを評価するために，高優先度サービスによるファイル参照時間を測定する．低優先度サービスによる影響を明らかにするため，バックグラウンドで低優先度サービスによるファイルバックアップ処理を行う場合と行わない場合を測定した．

実サービス測定の様子を**図 4** に示す．ファイル参照処理を行う高優先度の AP プロセス（参照 AP) は，4KB のファイルを 50 回読み込む処理を行う．ファイル複写処理を行う低優先度の AP プロセス（複写 AP）は，4KB のファイルを読み込み，異なるディレクトリ以下に書き出す処理を 100 ファイルに対して行う．参照 AP のファイル参照処理において，ブロックキャッシュにヒットする場合とヒットしない場合を測定するために，1 ファイルを 50 回読み込む場合と 50 ファイルを各 1 回ずつ読み込む場合を測定した．また，PU 処理の負荷を与えるため，参照 AP と複写 AP のそれぞれにおいて，ファイル参照処理またはファイル複写処理 1 回ごとに PU 処理（0 μs，50 μs）を行う場合を測定した．複写 AP と参照 AP をコア 0 上に配置し，ファイル操作処理に関連するファイル管理サーバ（FS)，ブロック管理サーバ（BLK)，ディスクドライバサーバ（DK）をコア 1 上に配置する．測定結果は 5 回試行した場合の平均時間である．

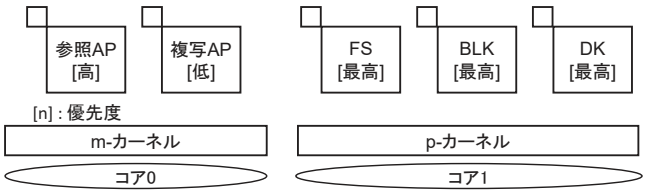

図 4 実サービス測定の様子

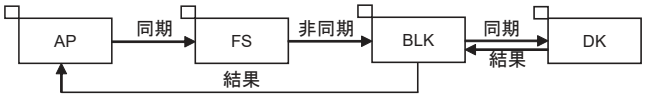

図 5 ***AnT*** のファイル操作処理の様子

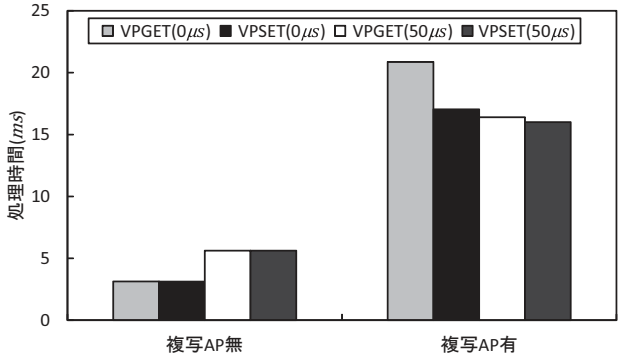

図 6 参照 AP 処理時間（1 ファイル× 50 回）

3.2.2 *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 5** に示し，以下に説明する．ファイル操作処理に関連する OS サーバは 3 種類である．FS は，i ノードの管理を行う．BLK は，ブロックキャッシュの管理を行う．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際を例にファイル操作の処理流れを説明する．AP プロセスは，FS に対してファイル読み込みを同期処理依頼する．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処理依頼する．BLK は，当該データブロックをブロックキャッシュから探索し，AP プロセスに対して返却する．ブロック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処理依頼する．

3.2.3 測定結果と考察

1 ファイルを 50 回読み込む場合の参照 AP の処理時間を

© 2016 Information Processing Society of Japan

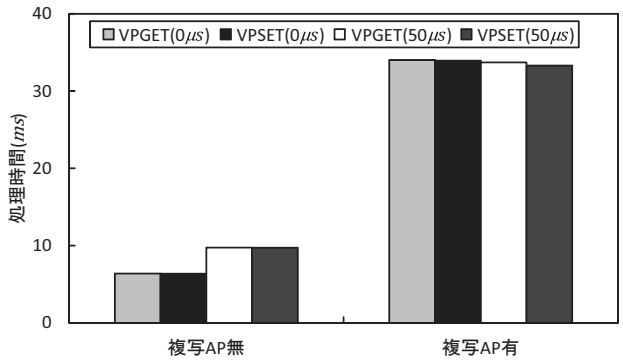

図 7 参照 AP 処理時間（50 ファイル× 1 回）

図 6 に示す．この測定結果より，以下のことが分かる．

（1）PU 処理が 0 μs で複写 AP が共存する場合の VPSET の処理時間は，VPGET より約 3.8 ms 短い．一方，複写 AP が共存しない場合の VPGET と VPSET の処理時間の差はない．これらの結果から，VPSET の優先度逆転の抑制効果は，VPGET より高いと考えられる．

（2）PU 処理が 50 μs で複写 AP が共存する場合の VPSET の処理時間は，VPGET より約 0.4 ms 短い．一方，PU 処理が 0 μs で複写 AP が共存する場合の VPSET の処理時間は，VPGET より約 3.8 ms 短い．これらの結果から，PU 処理を行うと VPSET と VPGET の処理時間の差が小さくなると考えられる．したがって，PU 処理時間の増加に伴い優先度逆転の抑制効果が減少すると考えられる．

50 ファイルを各 1 回ずつ読み込む場合の参照 AP の処理時間を**図 7** に示す．PU 処理が 0 μs と 50 μs のどちらの場合でも，VPGET と VPSET の処理時間の差はほとんどない．50 ファイルを各 1 回ずつ読み込む場合，I/O ネックとなる．したがって，I/O ネックの場合，優先度逆転の抑制効果は小さいと考えられる．

4. 関連研究

本稿のスケジュール機構は，マルチコア環境において優先度逆転を抑制し，かつコア間通信回数を削減することでオーバヘッドを削減する．

文献 [9] では，Mach[4] の実時間性を向上させるために，実時間処理のためのプロセス間通信ポートを用意している．このポートには，メッセージのキューイング方法や優先度継承するかどうかを設定することができ，優先度逆転抑制法を実現できる．しかし，Mach は，マルチコア環境を想定していない．このため，Mach において提案手法を実現する場合，プロセス間通信をマルチコア環境に対応させ，コア毎に独立したスケジューラを配置させ，コア間通信を実現する必要がある．

文献 [10] では，共有資源の操作において発生するデッドロックの対処を参考に，代替資源を用意することで優先度逆転に対処できることを示している．例えば，OS サーバにおける優先度逆転は，優先度の異なる OS サーバを複数用意するか，OS サーバの処理実行中に他の依頼を実行可能にすることで対処できる．優先度の異なる OS サーバを用意する場合，資源が無駄になる可能性がある．また，OS サーバの処理実行中に他の依頼を実行可能にすると制御が複雑化する．

文献 [11] では，シングルコア環境の L4[2] において，スレッドのコンテキストを 2 つに分割することでプロセス間通信のオーバヘッドを最小限に抑制し，かつ優先度継承と優先度上限プロトコルを実現している．この手法は，シングルコア環境のプロセス間通信においてコンテキストの利用を工夫しており，マルチコア環境において発生するコア間のプロセス間通信に適用することができない．このため，本稿にて提案するスケジュール機構と組み合わせるのは難しい．

文献 [12] では，分割スケジューリングにおいて優先度継承が無力であることを示し，ロック保持タスクにロック解放待ちタスクと同じコアで動作できる資格を付与する移譲優先度継承を提案している．この手法は，モノリシックカーネルである Linux に実現しており，優先度継承を排他制御によって実現している．ロックに適用している移譲優先度継承を OS サーバに対して実現するべきか否かは，OS サーバの移譲のコストと優先度逆転による影響によって異なる．

文献 [13] では，L4[2] を拡張した優先度継承機構を持つ Fiasco をマルチコア環境に対応させる際の設計方針とマルチコアでの優先度継承方式（local-helping）を提案している．この手法は，本稿のスケジュール機構と異なり，依頼元プロセスのコア上に依頼先プロセスを移譲し，クリティカルセクションを実行させる．この手法は，処理が非常に短いクリティカルセクションの場合，有効である．しかし，ファイル管理サーバのような OS サーバの処理は，依頼処理時間が長い．また，プロセスを移譲するコストは大きい．したがって，OS サーバの優先度継承では，本稿のスケジュール機構の方が小さいオーバヘッドで優先度逆転を抑制できると考えられる．

5. おわりに

本稿では，コア毎に独立したスケジューラを持つ ***AnT*** について，プロセス間の優先度逆転を抑制し，かつコア間通信回数を削減するスケジュール機構を評価し結果を述べた．

基本評価として，抑制法による高優先度 AP プロセスへの影響とコア間通信回数の削減による効果を明らかにした．具体的には，Intel Xeon E5-2630 v3（8Core，2.4GHz）の計算機において，異なるコア上の AP プロセスと OS サーバ間で通信を行った際の処理時間を測定し，制御を行わな

© 2016 Information Processing Society of Japan

い VPSET の処理時間と制御を行う VPSET の処理時間の最大値の差が 1.57 μs であることから，VPSET の制御によるオーバヘッドが 1.57 μs であり，小さいことを示した．また，上記の環境で OS サーバと同一コア上で PU 処理を 100 μs 行う AP プロセスを走行させた場合の AP プロセスと OS サーバ間の通信の処理時間を測定し，制御を行わず優先度逆転が生じない VPGET の処理時間と制御を行い優先度逆転が生じる場合の VPGET の処理時間の差が 99.95 μs であることから，VPGET の制御によるオーバヘッドと優先度逆転時間の和が 99.95 μs であり，優先度逆転の影響が大きいことを示した．さらに，本計算機におけるコア間通信の処理時間が 0.93 μs であり，コア間通信を伴わないサーバプログラム間通信の処理時間が約 7 μs であることから，コア間通信を 1 回削減するごとにサーバプログラム間通信の処理時間に対して，約 13%のオーバヘッドを削減できることを示した．

実サービス評価として，高優先度サービスによるファイル参照がバックグラウンドで動作する低優先度ファイルバックアップサービスによって受ける影響を明らかにし，抑制法の優先度逆転の抑制効果を確認した．結果から，VPSET の方が VPGET より優先度逆転の抑制効果が高いことを示した．また，I/O ネックの環境では優先度逆転の抑制効果が小さいことを示した．

参考文献

[1] Sha, L., Rajkumar, R., Lehoczky, J.P.: Priority inheritance protocols: an approach to real-time synchronization, IEEE Transactions, Vol.39, No.9, pp.1175–1185 (1990).
[2] Liedtke, J.: Toward real microkernels，Communications of the ACM, Vol.39, No.9, pp.70–77 (1996).
[3] 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).
[4] 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).
[5] 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).
[6] 鴨生 悠冬, 山内 利宏, 谷口 秀夫: マルチコア環境における優先度逆転を抑制する ***AnT*** オペレーティングシステムのスケジュール機構, 情報処理学会研究報告, Vol.2016-OS-136, No.17, pp.1-8 (2016).
[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.1977–1987 (2010).
[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] Kitayama, T., Nakajima, T., and Tokuda, H.: RT-IPC:An IPC Extension for Real-Time Mach, USENIX Microkernels and Other Kernel Architectures Symposium, pp.91–104 (1993).
[10] Levine, G.: Priority Inversion with Fungible Resources, ACM SIGAda Ada Letters, Vol.31, No.2, pp.9–14 (2011).
[11] Steinberg, U., Wolter, J., Hartig, H.: Fast component interaction for real-time systems, In Euromicro Conference on Real-Time Systems, pp.89–97 (2005).
[12] Björn B. Brandenburg, Andrea Bastoni: The Case for Migratory Priority Inheritance in Linux: Bounded Priority Inversions on Multiprocessors, Proceeding of the 14th Real-Time Linux Workshop (RTLWS2012), pp.67–86 (2012).
[13] Michael, H. and Michael, P.: Helping in a Multiprocessor Environment, In Proceedings of the Second Workshop on Common Microkernel System Platforms, pp.1–5, (2001).

© 2016 Information Processing Society of Japan