Vol.2018-OS-142 No.5
2018/2/27

Tender オペレーティングシステムの資源プール機能の評価

田村 大[1]　山内 利宏[1]　谷口 秀夫[1]

概要：***Tender*** オペレーティングシステムでは，オペレーティングシステムが制御し管理する対象を資源とし，資源の分離と独立化を実現している．このため，資源プール機能により，プロセスを構成する資源を保持しておき，プロセス生成処理時に利用することで処理を高速化できる．本稿では，評価により，資源プール機能について有効性を示す．また，***Tender*** と Linux の実メモリの割り当て方式と処理時間について比較する．

1. はじめに

オペレーティングシステム（以降，OS）は，プロセスを基本単位として応用プログラム（以降，AP）の実行を管理し制御する．このため，プロセス生成やプロセス走行時における OS オーバヘッドの増加は，AP によるサービス提供に悪影響を与える．Linux などの既存 OS では，デマンドページング機能 [1] やコピーオンライト機能 [2] を利用することにより，プロセス生成処理時のメモリ確保処理と複写処理を最小限にし，プロセス生成処理を高速化している．しかし，プロセス走行時のメモリ領域への初アクセスによりページ例外が発生し，オーバヘッドが生じる．一方，分散指向永続オペレーティングシステム ***Tender***[3]（The ENduring operating system for Distributed EnviRonment）は，プロセス生成処理時に AP の内容を全てメモリ上に読みこむ．このため，プロセス走行時に，ページ例外は発生しないものの，プロセス生成処理時におけるメモリ確保処理とプログラム内容の読み込み処理の OS オーバヘッドが大きい．

Tender は，OS が制御し管理する対象を資源として細分化し，資源の分離と独立化を行っている．したがって，***Tender*** におけるプロセスは，複数の資源から構成される．また，プロセスを構成する資源（以降，プロセス構成資源）は，プロセスの存在に関係なく生成や存在が可能である．これらの特徴により，プロセス削除時にプロセス構成資源を削除せずに保持することやプロセス構成資源が必要となる前に生成して保持することが可能であり，これらの資源を利用することでプロセス生成処理を高速化する機構を実現し，有効性を示した [3][4][5][6]．

本稿では，***Tender*** の資源プール機能について評価する．具体的には，Linux と ***Tender*** について，Apache Web サーバを用いた評価により，プロセス生成と走行時の処理時間の観点から比較評価する．

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

2. *Tender* オペレーティングシステム

2.1 資源の分離と独立化

Tender では，OS が制御し管理する対象を資源として細分化し，資源の分離と独立化を行っている．Linux などの既存 OS は，資源に相互の依存関係があるため，OS が扱う資源の粒度が大きく，かつ資源を構成する資源は独立して存在できない．一方，***Tender*** は，資源の分離と独立化により，個々の資源が独立して存在できる．資源の分離と独立化を行うことで，各資源を操作するためのプログラムを部品化できる．このため，機能の追加や削除が容易になる．また，各資源を操作するためのプログラムの呼び出しは，特定のプログラムを介する特徴がある．このプログラムが各資源を管理するプログラムの呼び出しを管理するため，OS の動作や内部状態の理解や把握が容易になる．

2.2 プロセス構成資源

Tender におけるプロセスを構成する資源を**図 1** に示す．「プロセス」は，AP の実行単位であり，プロセス識別子とプロセス管理表の情報を持つ．「プログラム」は，AP のテキスト部，データ部，BSS 部の大きさと先頭アドレス，および走行開始アドレスの情報を持つ．AP のプログラム実体などの内容は「プレート」が保持する．「プレート」は，永続的な記憶をメモリ上に提供する資源であり，既存 OS のファイルに相当する．また，「演算」は，プロセスへのプロセッサ割り当て単位を資源化したもので，「プロセス」とは独立して存在する．「プロセス」は，「演算」を確保することで，プロセッサ割り当てを受けて走行できる．

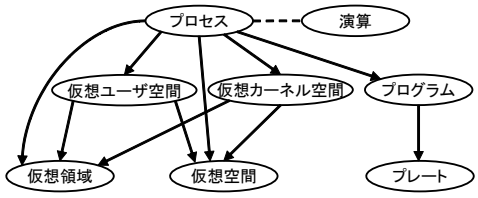

図 1　プロセスを構成する資源

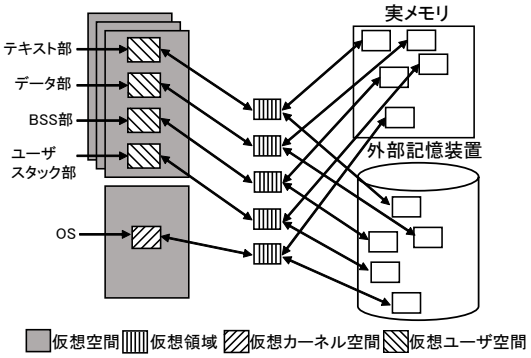

図 2　***Tender*** におけるメモリ関連資源

また，***Tender*** におけるメモリ関連資源の関係を**図 2** に示す．「仮想空間」は，仮想アドレスの空間であり，仮想アドレスを実アドレスに変換する変換表に相当する．「仮想領域」は，実メモリと外部記憶装置の領域を仮想化した領域であり，サイズはページサイズ（4KB）の整数倍である．この領域の実体は，実メモリか外部記憶装置の領域上に存在する．「仮想カーネル空間」と「仮想ユーザ空間」は，「仮想領域」を「仮想空間」の持つ仮想アドレスと対応付けたものである．「仮想カーネル空間」と「仮想ユーザ空間」の差異は，ユーザモードで走行するプログラムがアクセス可能か否かの違いである．「仮想カーネル空間」は，カーネルモードで走行するプログラムのみアクセス可能であるのに対し，「仮想ユーザ空間」は，ユーザモードで走行するプログラムもアクセス可能である．プロセスのテキスト部，データ部，BSS 部，およびユーザスタック部は，「仮想ユーザ空間」として「仮想空間」上に存在する．ここで，データ部と BSS 部はそれぞれ，初期値を持つ変数の集合と持たない変数の集合である．

2.3　資源プール機能

2.3.1　基本機構

資源プール機構を**図 3** に示す．AP の実行により OS は，プロセスを生成し，実行し，削除する．プロセス削除の際，プロセス構成資源を削除せずに，資源プールに資源を保持する．これにより，プロセス生成の際，資源プールに保持された資源（以降，プール資源）を取得し，再利用することで，プロセス構成資源の生成処理を省略し，プロセス処理を高速化する．プール資源は，資源管理部が管理する．

また，資源プール機能は，一度利用した資源を再利用す

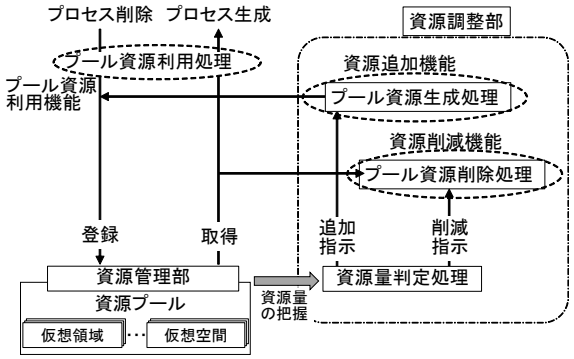

図 3　資源プール機構

る機能（以降，再利用機能）に加え，資源を生成して資源プールに追加する機能（以降，資源追加機能）と資源プールから資源を削除して資源プールの資源量を削減する機能（以降，資源削減機能）からなる．これらの機能は，資源調整部が提供する．資源追加機能は，利用できる条件が一致しやすい資源を生成し，資源プールに追加する機能である．本機能により，プロセス生成処理時に資源プールの資源を利用できる可能性を上げ，プロセス生成処理を高速化できる．これにより，プール資源の不足により，プロセス生成処理が高速化できない問題に対処している．資源削減機能は，利用されずに資源プールに保持され続ける資源を削除し，資源プールの資源量を削減する機能である．これにより，プール資源の増加により未使用メモリが枯渇する問題に対処している．資源を削除することで資源量を調整する具体的な方法については，後述する．

2.3.2　プール資源

プール資源を**表 1** に示す．プール資源は，大きく 2 つに分類できる．1 つは，プログラムの内容に依存する情報を保持しない資源（以降，プログラム非依存資源）である．もう 1 つは，プログラムの内容に依存する情報を保持する資源（以降，プログラム依存資源）である．

プログラム非依存資源として，以下の 3 種類がある．

(1) ワーク領域用仮想カーネル空間は，新規プロセスに渡す引数を格納するための一時的な領域であり，常に利用可能である．メモリ使用量は，ページサイズ（4KB）である．

(2)「仮想空間」は，仮想アドレスの空間であり，ページディレクトリとページテーブルの情報を持ち，常に利用可能である．メモリ使用量は，ページディレクトリのサイズ (S_{pd}=4KB) とページテーブルのサイズ (S_{pts}KB) の合計である．

(3)「仮想領域」は，実メモリと外部記憶装置の領域を仮想化した領域であり，ページサイズの整数倍で管理されている．このため，サイズが同じであれば，「仮想領域」を利用可能である．メモリ使用量は，「仮想領域」のサイズ (S_{vr}KB) 分である．

また，プログラム依存資源として，以下の 3 種類がある．

表 1　プール資源

通番	資源の種類	利用の観点	保持できる資源	メモリ使用量 (KB)
1	プログラム非依存資源	常に利用可能	ワーク領域用仮想カーネル空間	4
2			「仮想空間」	S_{pd}+S_{pts}
3		サイズが同じ場合	「仮想領域」	S_{vr}
4	プログラム依存資源	プログラム内容が同じ場合	「プログラム」	0
5			内容を利用できるテキスト部用仮想領域	S_{text}
6			各部の仮想領域が対応づけられた仮想空間	S_{text}+S_{data}+S_{BSS}+S_{us}+S_{pd}+S_{pts}

(4)「プログラム」は，各部の先頭アドレスとサイズ，および走行開始アドレスといった AP の情報を持つ．情報の格納場所は，***Tender*** 起動時に確保される領域を使用するため，生成時にメモリを確保しない．
(5) 内容を利用できるテキスト部用仮想領域は，プログラムのテキスト部のデータが読みこまれた「仮想領域」である．メモリ使用量は，プログラムのテキスト部のサイズ (S_{text}KB) 分である．
(6) 各部の仮想領域が対応づけられた仮想空間は，プログラムに対応するテキスト部，データ部，BSS 部，およびユーザスタック部用の「仮想領域」が対応付けられた「仮想空間」である．メモリ使用量は，プログラムのテキスト部，データ部，および BSS 部のサイズ (S_{text}KB，S_{data}KB，および S_{BSS}KB)，ユーザスタック部のサイズ (S_{us}=64KB)，および「仮想空間」のサイズ (S_{pd}+S_{pts}KB) 分である．

2.4　資源量の調整方法

2.4.1　追加する資源量の決定法

資源追加機能により追加する資源量は，資源管理部から取得するプール資源の量の情報と設定情報により決定する．設定情報とは，資源プールに保持しておく資源の種類と量の情報である．追加する資源量は，資源プールにあるプール資源量が設定量となるように，資源プールに資源を追加する．

設定情報は，計算機管理者が設定する．このため，計算機管理者は，提供するサービスが必要とするプロセスを把握し，適切に設定する必要がある．

2.4.2　削減する資源量の決定法

資源削減機能により削減する資源量は，表 1 に示した各資源の利用の観点基づき，削除する資源量を決定する．以下に削除する資源量の決定法を示す．
(1) 常に利用可能：この資源は，削除しない．これは，新たなプロセス生成処理時に常に利用できるためである．
(2) サイズが同じ場合：この資源は，利用される確率の高低で分類し，削除する量を決める．サイズが 64KB 以下の場合は利用される確率が高い [4] ため削除しない．サイズが 64KB より大きい場合，一定時間以上利用されなければ，削除する．これは，一定時間利用されない場合，利用される確率は低いと考えられるためである．
(3) プログラム内容が同じ場合：この資源は，同じプログラム内容のものを 1 個残す．これにより，当該プログラムを利用する最初のプロセス生成処理時間を短縮できる．なお，2 個目以降のプロセス生成は，テキスト部の共有機能が有効に働き，高速化される．また，一定時間利用されない場合，残した 1 個も削除する．これは，利用される確率は低いと考えられるためである．なお，表 1 の資源において，(5) 内容を利用できるテキスト部用仮想領域と (6) 各部の仮想領域が対応づけられた仮想空間は，他の複数の資源から構成される．具体的には，(5) 内容を利用できるテキスト部用仮想領域は，(3)「仮想領域」と (4)「プログラム」から構成される．また，(6) 各部の仮想領域が対応付けられた仮想空間は，(2)「仮想空間」，(3)「仮想領域」，および (5) 内容を利用できるテキスト部用仮想領域から構成される．このように複数の資源から構成される資源を削除する際，当該資源は削除するものの，構成する資源は削除せずにプール資源として残す．このプール資源として残した資源は，当該資源の資源量調整方法に基づき，その量が調整される．

3.　評価

3.1　評価目的と方法

資源プール機能の再利用機能，資源追加機能，および資源削減機能の効果を明らかにするために，Apache Web サーバを用いて評価する．また，既存 OS として，Linux と比較評価し，***Tender*** の資源プール機能の有効性を示す．ただし，***Tender*** と Linux は，プロセス生成時の実メモリ割り当て方式が異なるため，実メモリ割り当て方式の違いによる処理時間への影響を明らかにしたうえで，資源プール機能の有効性について考察する．***Tender*** は，プロセス生成時にプロセスのメモリ領域に実メモリをすべて割り当てる．これにより，プロセス生成時の処理時間は長くなるものの，実メモリがページアウトされなければ，プロセス実行時にページ例外と実メモリ割り当て処理は発生しない．一方，Linux は，プロセス生成時に実メモリを割り当てず，メモリ領域の各ページへの最初のアクセス時に，ページ例外が発生し，実メモリ割り当て処理を行う．このため，実行時にページ例外処理によるオーバヘッドが発生する．

クライアント計算機上で ApacheBench を実行し，Web

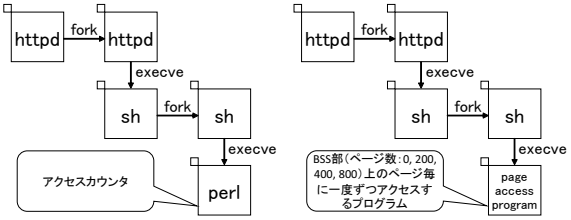

図 4　CGI プログラムの処理流れ

表 2　プログラムサイズ

プログラム名	テキスト部	データ部	BSS 部
httpd	656 KB	36 KB	72 KB
sh	328 KB	16 KB	16 KB
perl	816 KB	40 KB	28 KB
page access program	4 KB	4 KB	0 KB（　0 ページ）
			800 KB（200 ページ）
			1,600 KB（400 ページ）
			3,200 KB（800 ページ）

サーバに対し 100 回の要求を行い，合計応答時間を測定した．この際，並列要求無の場合と 10 並列要求の場合でそれぞれ測定した．Web サーバへの要求は，静的なページへの要求に加え，Web サーバがプロセス生成を行う事例として，CGI プログラムの実行を伴うページを Web サーバに要求する場合も評価した．

CGI プログラム実行時の処理流れを**図 4** に示す．CGI プログラムとして，perl で記述したプログラムと BSS 部のページ全てに 1 度ずつアクセスするプログラム（以降，page access program）を使用した．

(A) perl で記述した CGI プログラム：ファイルから値を読み出し，その値を表示する処理を実行する．静的なページと CGI プログラムとして perl の実行を伴うページへの要求に対する応答時間の観点より，***Tender*** の資源プール機能の有効性を示す．また，Linux と比較する．

(B) BSS 部にアクセスする CGI プログラム：page access program の BSS 部のページ数は，0，200，400，および 800 の 4 通りで測定しており，ページ数が 0 の場合，プログラムは実行後すぐに終了する．これにより，Linux におけるページ例外発生回数と応答時間の関係を明らかにし，***Tender*** と比較する．

Web サーバプログラムとして，Apache ver.1.3.33 を使用する．また，perl は，perl ver.5.005_03 を使用する．httpd，sh，perl，および page access program のプログラムサイズを**表 2** に示す．

評価環境を**表 3** に示す．クライアント計算機とサーバ計算機を 100Base-TX の Ethernet で接続して評価する．

サーバ計算機で ***Tender*** を使用する場合，以下の 3 つの場合で測定した．

(1) 資源プール機能を利用しない場合（以降，基本）

(2) 測定区間の開始時においてプール資源が存在しないで資源を再利用する機能を使用する場合（以降，再利用）

(3) 測定区間の前に資源追加機能により「各部の領域が対応付けられた仮想空間」を生成して再利用機能を使用する場合（以降，(再利用+追加)）

また，CGI プログラムの実行を伴うページに 10 並列要求する場合において，上記の 3 つの場合に加え，

(4) 測定区間の前に資源追加機能により「各部の領域が対応付けられた仮想空間」を生成して再利用機能と資源削減機能を使用する場合（以降，(再利用+追加+削減)）

も測定した．これは，10 並列要求の場合，サーバ計算機上で，複数プロセスが同時に走行するため，再利用機能により，資源プールの資源量が増加するためである．そこで，10 並列要求の場合，プール資源のメモリ使用量を測定した．なお，資源削減機能を使用する場合，プール資源が一定時間利用されない場合に削除される時間と資源削減の実行間隔を共に 50ms とした．

サーバ計算機で Linux を使用する場合，ページ例外発生回数と処理時間の差を明らかにするために，要求時に発生するページ例外発生回数を測定した．

また，ディスク I/O は発生しない場合とした．なお，資源プール機能は，マルチコア対応していないため，使用 1 コアで評価し，Linux も同様に使用 1 コアで評価した．

表 3　評価環境

	client machine	server machine
OS	FreeBSD 11.0-RELEASE	***Tender***, Linux 2.6.32-573.el6.i686 (CentOS 6.9)
CPU	Intel Core i7-4770 (3.40 GHz) (4 cores)	Intel Core i3-2100 (3.10 GHz) (use 1 core)
RAM	4,096 MB	4,096 MB

3.2　評価結果

3.2.1　資源プール機能の効果

Web サーバを使用し，***Tender*** の資源プール機能の有効性を示す．また，Linux と比較評価する．

静的なページ（html）と perl の実行を伴うページ（html+CGI）に対し，並列要求無と 10 並列要求した場合の合計応答時間をそれぞれ**図 5** と**図 6** に示す．また，10 並列要求時における，***Tender*** のプール資源のメモリ使用量を**図 7** に示し，***Tender*** の資源プール機能について考察する．

まず，並列要求無の場合（図 5）において，資源プール機能について考察する．

(1) 静的なページの要求に対する応答時間は，プール資源の有無にかかわらず，ほぼ一定である．これは，並列要求無のため，Apache Web サーバの子プロセス生成処理が行われないためである．

(2) CGI プログラムの実行を伴うページの要求に対する応答時間は，再利用の場合，基本の場合と比較し，応答時間を 94.16ms（=545.88−451.72）（17.2%=94.16/545.88）短

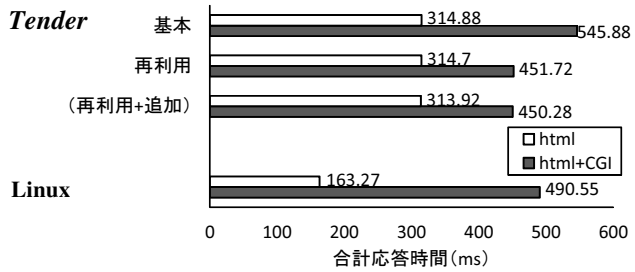

図 5　静的なページと perl の実行を伴うページ要求に対する合計応答時間（並列要求無）

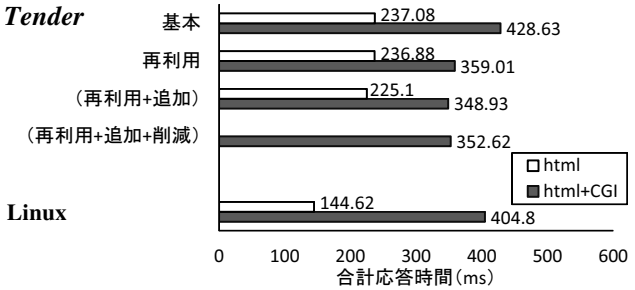

図 6　静的なページと perl の実行を伴うページ要求に対する合計応答時間（10 並列要求）

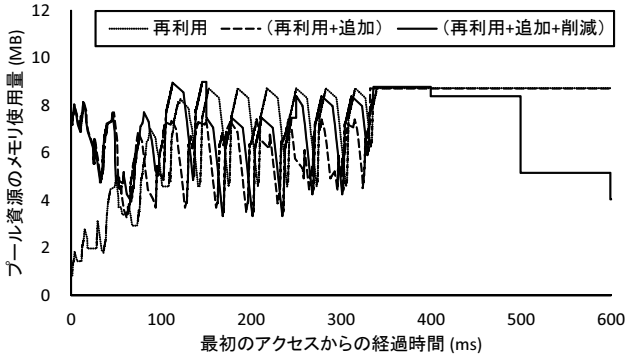

図 7　perl の実行を伴うページ要求時のプール資源のメモリ使用量（10 並列要求）

くできる．これは，CGI プログラムの実行時に再利用機能によるプール資源を利用し，プロセス生成処理を高速化できたためである．また，（再利用+追加）の場合，応答時間を更に 1.44ms（=451.72−450.28）短くできる．したがって，基本の場合と比較し，合計応答時間を 95.6ms（=545.88−450.28）（17.5%=95.6/545.88）短くできる．これは，1 回目の要求時における，sh と perl のプロセス生成処理時のプール資源利用のためである．

また，10 並列要求の場合（図 6）において，資源プール機能について考察する．

(1) 静的なページの要求に対する応答時間は，並列要求無の場合と異なり，（再利用+追加）の場合の応答時間は，再利用の場合と比較し，11.78ms（236.88−225.1）（5.0%=11.78/236.88）短い．これは，Apache Web サーバの子プロセス生成処理において，資源追加機能によるプール資源の利用のためである．

表 4　ページ例外発生回数（perl）

	並列要求無	10 並列要求
html	466	473
html+CGI	72,594	72,887

(2) CGI プログラムの実行を伴うページの要求に対する応答時間は，プール資源の利用により，応答時間を最大 79.7ms（=428.63−348.93）（18.6%=79.7/428.63）短くできる．

(3)（再利用+追加+削減）の場合は，（再利用+追加）の場合と比較し，資源削減の実行により，要求終了後におけるメモリ使用量を 4.68MB（=8.72−4.04）削減できる．しかし，合計応答時間は 3.69ms（=352.62−348.93）長くなる．一方，プール資源のメモリ使用量について，ApacheBench による要求中は，資源削減機能の有無によるプール資源のメモリ使用量の違いは見られない．これは，ApacheBench の要求によるプロセス生成処理によりプール資源が利用されるため，プール資源が資源削減機能による削除の対象とならないためである．このため，ApacheBench の要求終了時において，プール資源は，メモリを 8.72MB 使用している．また，資源削減の契機によっては，応答時間に影響を与えず，プール資源のメモリ使用量を削減可能であると考えられる．

以上より，資源プール機能の有効性を示した．具体的には，***Tender*** では，資源の分離と独立化による操作オーバヘッドがあるものの，再利用機能と資源追加機能により，静的なページの要求と CGI プログラムの実行を伴うページの応答時間をそれぞれ最大で 5.0%と 18.6%短くできることを示した．また，資源削減機能により，メモリ使用量を最大で 4.68MB 削減できることを示した．

次に，***Tender*** と Linux を比較評価する．

(1) 静的なページを要求した場合において，***Tender*** の合計応答時間は，Linux と比較し，並列要求無（図 5）と 10 並列要求（図 6）の場合でそれぞれ 151.61ms（=313.92−163.27）と 80.48ms（=225.1−144.62）以上長い．静的なページを要求する場合，Apache Web サーバは，並列要求数に応じて子プロセスを生成するものの，測定区間中において子プロセスは削除されない．このため，Linux 上のページ例外発生回数は増加しない．ページ例外発生回数を**表 4** に示す．ページ例外発生回数は，並列要求無と 10 並列要求の場合でそれぞれ 466 回，473 回と少なく，オーバヘッドは小さい．

(2) CGI プログラムの実行を伴うページを要求した場合，Linux の合計応答時間は，***Tender*** の基本の場合と比較し，並列要求無（図 5）と 10 並列要求（図 6）の場合でそれぞれ最大 55.33ms（=545.88−490.55）と 23.83ms（=428.63−404.8）短い．これに対し，***Tender*** が（再利用+追加）の場合の合計応答時間は，Linux と

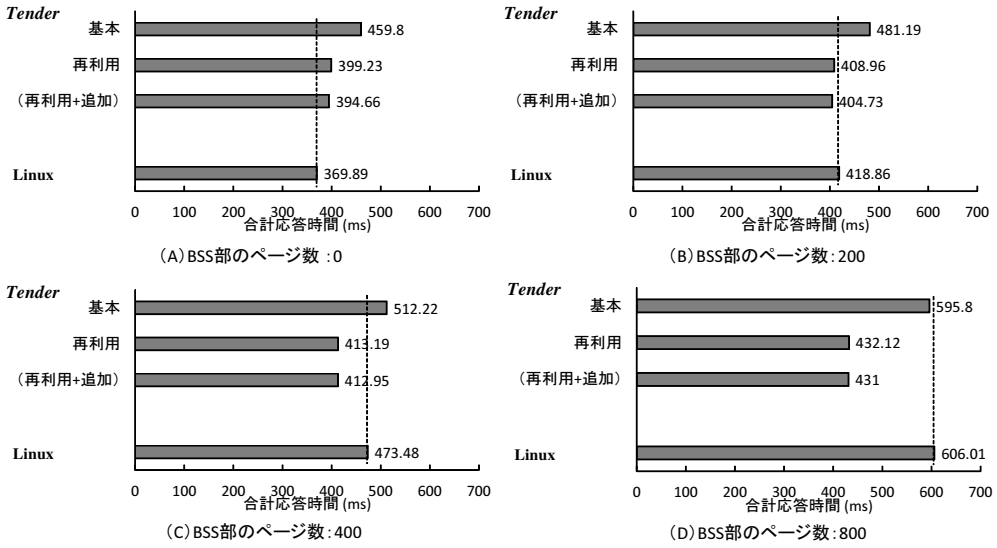

図 8 page access program の実行を伴うページ要求に対する合計応答時間（並列要求無）

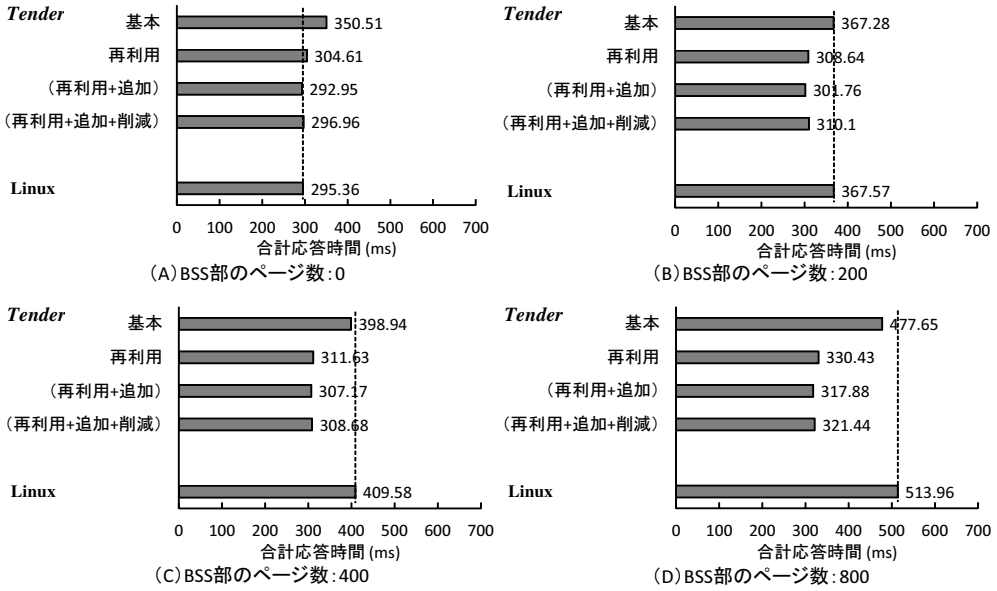

図 9 page access program の実行を伴うページ要求に対する合計応答時間（10 並列要求）

比較し，並列要求無と 10 並列要求の場合でそれぞれ最大 40.27ms（=490.55−450.28）(8.2%=40.27/490.55）と 55.87ms（=404.8−348.93）(13.8%=55.87/404.8）短くなる．つまり，並列要求無の場合，1 要求あたりの応答時間を平均で約 0.40ms，Linux より高速化したことがわかる．CGI プログラムの実行により，Linux では，並列要求無の場合と 10 並列要求の場合でそれぞれページ例外が 72,594 回と 72,887 回発生し，プロセス走行時のオーバヘッドが大きい．

以上より，CGI プログラム実行のように，プロセスが生成と削除を繰り返すような場合，Linux では，ページ例外発生回数が増加するため，プール資源を利用する ***Tender*** の方が性能で有利になる場合があることを示した．

3.2.2 ページ例外発生回数と処理時間の比較

CGI プログラムとして page acccess program を使用して，Linux 上で発生するページ例外発生回数と処理時間の関係から，***Tender*** と比較評価する．

表 5 ページ例外発生回数（page access program）

BSS 部のページ数	並列要求無	10 並列要求
0	49,054	49,834
200	69,911	69,696
400	89,947	89,651
800	129,685	129,852

並列要求無と 10 並列要求した場合の合計応答時間をそれぞれ**図 8** と**図 9** に示す．また，10 並列要求時における，***Tender*** のプール資源のメモリ使用量を**図 10** に示し，要求時における Linux のページ例外発生回数を**表 5** に示す．

まず，並列要求無（図 8）の場合を考察する．

(1) page access program の BSS 部のページ数が 0 の場合，(再利用+追加) の場合の ***Tender*** の合計応答時間は，Linux

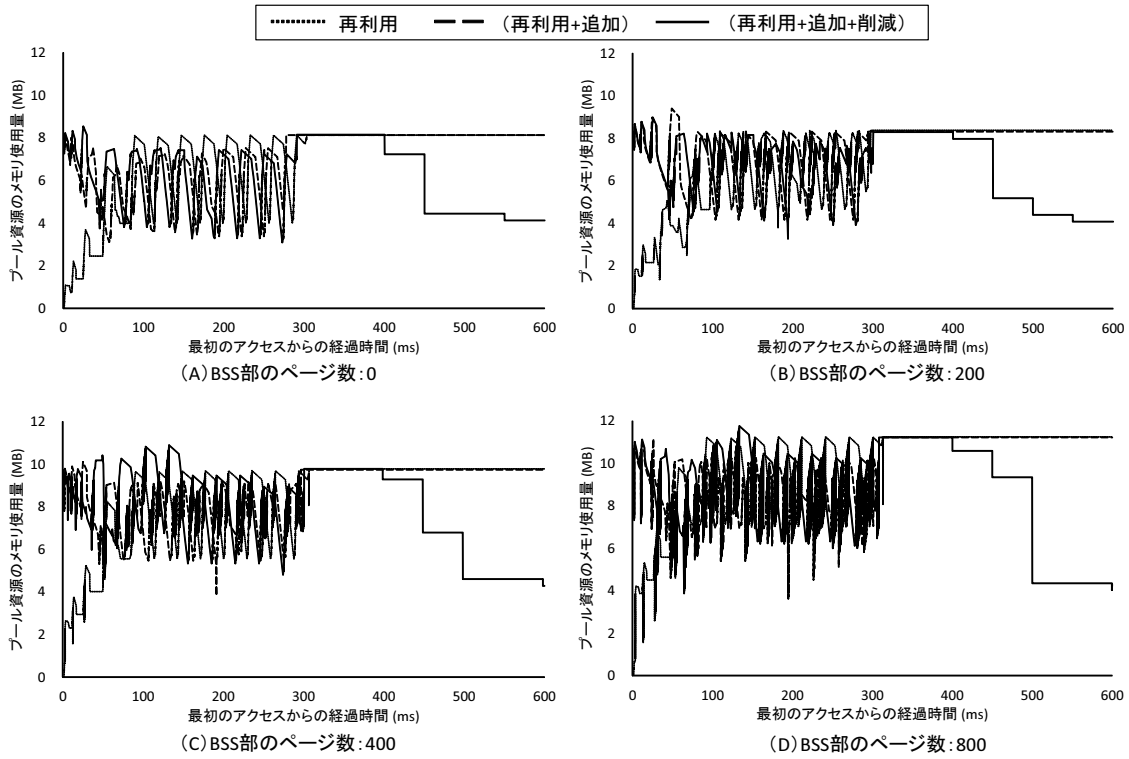

図 10 page access program の実行を伴うページ要求時のプール資源のメモリ使用量（10 並列要求）

と比較すると，24.77ms（=394.66−369.89）以上長い．

(2) page access program の BSS 部のページ数が 200 と 400 の場合，（再利用+追加）の場合の ***Tender*** の合計応答時間は，Linux と比較し，それぞれ最大，14.13ms（=418.86−404.73）（3.4%=14.13/418.86）と 60.53ms（=473.48−412.95）（12.8%=60.53/473.48）短い．したがって，1 要求あたりの応答時間は，平均でそれぞれ約 0.14ms と約 0.61ms 短くなる．これは，Linux において，BSS 部へのアクセス時に発生するページ例外により，オーバヘッドが大きくなるためである．具体的には，測定区間におけるページ例外発生回数は，BSS 部のページ数が 0 の場合，49,054 回であるのに対し，BSS 部のページ数が 200 と 400 の場合，それぞれ 69,911 回と 89,947 回であり，発生回数が多い．

(3) page access program の BSS 部のページ数が 800 の場合，プール資源利用の有無にかかわらず，***Tender*** の合計応答時間は，Linux より短い．***Tender*** がプール資源を利用した場合，Linux と比較すると，最大 175.01ms（=606.01−431.0）（28.9%=175.01/606.01）合計応答時間が短い．したがって，1 要求あたりの応答時間は，平均で約 1.75ms 短くなる．これは，Linux において，測定区間におけるページ例外発生回数が 129,685 回と更に多くなるためである．

次に，10 並列要求（図 9）の場合を考察する．

(1) page access program の BSS 部のページ数が 0 の時も，（再利用+追加）の場合の ***Tender*** の合計応答時間は，Linux と比較し，2.41ms（=295.36−292.95）短くなる．また，page access program の BSS 部のページ数が 200，400，および 800 の場合，プール資源の有無にかかわらず，***Tender*** は，Linux より，合計応答時間が短い．これは，ApacheBench の並列要求により，Apache Web サーバが子プロセスを生成したためである．具体的には，子プロセスの生成処理について，Linux ではプロセス生成処理の処理時間が長くなるのに対し，***Tender*** では，資源プール機能によりプロセス生成処理のオーバヘッドが小さくなったためである．

(2)（再利用+追加+削減）は，（再利用+追加）と比較すると，要求終了後における資源削減の実行により，メモリ使用量を最大 6.97MB（=11.25−4.28）削減できる．一方，CGI プログラムとして perl を使用した場合と同様，資源削減機能を利用することにより，合計応答時間は 3.56ms(=321.44−317.88) 長くなる．

以上より，Linux においてプロセス走行時のページ例外の発生回数が多くなると，プロセス生成時に AP の内容を読み込む方式でも，AP の処理性能が有利になる場合（図 8(C)，図 9(B)(C)(D)）があることを示した．また，***Tender*** の資源プール機能を利用することで，プログラム内容の読み込みなどの処理による，プロセス生成時の処理時間の長大化を抑制し，AP の処理性能が有利になる場合（図 8(B)(C)，図 9(A)）があることを示した．

4. 関連研究

プロセス生成処理を高速化する手法として，UNIX の vfork システムコール [7] や Linux の clone システムコール [8] を使用し，親プロセスのページテーブルの複写を省略する方法がある．vfork は，発行した関数内で，exit システムコールまたは，execve システムコールの発行を必要し，clone は，複写する範囲を引数で指定する．文献 [9] では，組み込みシステムにおけるプロセス生成処理の高速化として，親プロセスが AP を動的リンクライブラリとして事前に読み込むことで，子プロセス生成処理時に AP バイナリの配置を省略し，処理を高速化している．この手法は，プロセス生成処理時に fork と execve を使用する代わりに，fork と dlopen を使用する．また，文献 [10] では，同一 AP からの複数プロセス生成時に一括で AP の内容を読み込み，プロセス生成処理を高速化している．これらの手法は，使用するカーネルコールの変更により実現されるため，AP の変更を必要とする点が提案手法と異なる．

プロセス生成処理を投機的に実行することで処理を高速化する事例がある．例えば，Apache Web サーバでは，子プロセスを事前に生成し，要求時に発生するプロセス生成処理を省略している [11]．また，文献 [12] では，OS の機能を利用した AP 実行の投機的実行により，プロセス走行開始までの処理時間を短縮している．しかし，これらの手法は，特定の AP に特化した手法であり，AP により実現される点が提案手法と異なる．

資源削減機能と類似した機能として，ガベージコレクション [13]（以降，GC）がある．GC は，資源削減機能と同様，使用されないメモリ領域を解放し，メモリ使用量を削減する．しかし，GC は，AP が管理するメモリ領域を AP により解放してメモリ使用量を抑制するのに対し，資源削減機能では，AP の変更なしにメモリ使用量を抑制する点で異なる．

5. おわりに

Tender の資源プール機能について，Apache Web サーバを使用し，有効性を示した．また，Linux と比較評価した．

Apache Web サーバを使用し，応答時間を評価した．***Tender*** の資源プール機能により，応答時間を短縮できる．また，Linux と比較し，最大約 0.40ms（8.2%）短くなる．BSS 部のページにアクセスするプログラムの BSS 部のサイズを変更し，CGI プログラムとして使用した場合，ページのアクセス数，つまりページ例外の発生回数が多いと，***Tender*** の応答時間は Linux より短くなる．例えば，BSS 部のページ数が 800 ページである CGI プログラム実行を伴うページの要求に対する応答時間は，最大約 1.75ms（28.9%）短い．しかし，静的なページやページ例外発生回数が少ない CGI プログラム実行を伴うページの要求の場合，Linux の方が応答時間が短い．以上より，資源の分離と独立化によるオーバヘッドがある ***Tender*** でも，資源プール機能により処理を高速化できることを示した．また，***Tender*** の資源プール機能を利用することで，プロセス生成の処理時間の長大化を抑制し，Linux より AP の処理性能が有利になる場合があることを示した．

残された課題として，追加する資源の種類と量を決定する機能の実現がある．

参考文献

[1] C. J. Kuehner, B. Randell: Demand Paging in Perspective, Proc. December 9-11, 1968, Fall Joint Computer Conference, Part II (AFIPS '68 (Fall, part II)), pp.1011–1018 (1968).
[2] J. M. Smith, and G. Q. Maguire Jr.: Effects of copy-on-write memory management on the response time of UNIX fork operations, COMPUTING SYSTEMS, Vol.1, No.3, pp.255–278 (1988).
[3] 谷口 秀夫，青木 義則，後藤 真孝，村上 大介，田端 利宏：資源の独立化機構による ***Tender*** オペレーティングシステム，情処学論，Vol.41，No.12，pp.3363–3374（2000）．
[4] 田端 利宏，谷口 秀夫：プロセス構成資源の効率的な再利用を目指した資源管理方法の提案，情処学論：コンピューティングシステム，Vol.44，No.SIG10(ACS2)，pp.48–61（2003）．
[5] 佐伯 顕治，田端 利宏，谷口 秀夫：***Tender*** の資源再利用機能を利用した高速 fork & exec 処理の実現と評価，信学論（D），Vol.J91-D，No.12，pp.2892–2903（2008）．
[6] 田村 大，佐藤 将也，山内 利宏，谷口 秀夫：***Tender*** におけるプロセス構成資源の事前生成による高速プロセス生成機能の評価，コンピュータシステム・シンポジウム (ComSys2016) 論文集, vol.2016, pp.94–101（2016）．
[7] D. Eckhardt, M. Haardt, I. Jackson, M. Kerrisk: vfork(2) - Linux Programmer's Manual, http://man7.org/linux/man-pages/man2/vfork.2.html, accessed Jan. 9, 2018.
[8] D. Eckhardt, M. Haardt, I. Jackson, M. Kerrisk: clone(2) - Linux Programmer's Manual, http://man7.org/linux/man-pages/man2/clone.2.html, accessed Jan. 9, 2018.
[9] C. Jung, D. Woo, K. Kim, S. Lim: Performance Characterization of Prelinking and Preloadingfor Embedded Systems, Proc. 7th ACM & IEEE International Conference on Embedded Software (EMSOFT'07), pp.213–220 (2007).
[10] A. Kulkarni, L. Ionkov, M. Lang, A. Lumsdaine: Optimizing process creation and execution on multi-core architectures, International Journal of High Performance Computing Applications, Vol 27, Issue 2, pp.147-161 (2013).
[11] The Apache Software Foundation: Apache MPM prefork, https://httpd.apache.org/docs/2.4/mod/prefork.html, accessed Jan. 9, 2018.
[12] B. Wester, P. M. Chen, J. Flinn: Operating System Support for Application-specific Speculation, Proc. 6th Conference on Computer Systems (EuroSys'11), pp.229–242 (2011).
[13] J. McCarthy: Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I," Commun. ACM, Vol.3, Issue 4, pp.491–502 (1960).