Webサーバを利用した*Tender*の資源「演算」の評価
–複数プロセスの実行性能調整機能–

田端 利宏† 野口 直樹†
中島 耕太† 谷口 秀夫‡

九州大学大学院システム情報科学府†
九州大学大学院システム情報科学研究院‡

計算機の性能向上により，1台の計算機で多くのサービスを提供することが可能になった．しかし，サービス毎に要求する実行性能が異なるため，各サービスに合わせた実行性能の保証が必要になっている．また，ひとつのサービスは，複数のプロセスで構成されることが多い．したがって，複数のプロセスをひとつの単位とし，その単位で実行性能を調整できる機能が望まれる．我々は，***Tender***オペレーティングシステムにおいて，プロセッサの割り当て単位である資源「演算」を利用した，複数プロセスの実行性能調整機能を提案している．本論文では，提案した機能をWebサーバを利用して評価した結果を報告する．具体的には，Apache Webサーバを***Tender***で実行するために実現したBSD/OS互換システムコールインタフェースについて述べる．また，BSD/OS互換システムコールインタフェース，およびApache Webサーバの性能を評価した結果を報告する．最後に，提案した機能の評価結果を報告する．

Evaluation of *Execution* Resource on ***Tender*** by Using Web server
–A Mechanism of Regulating Execution Performance for Multi Process–

Toshihiro TABATA, Naoki NOGUCHI,
Kohta NAKASHIMA and Hideo TANIGUCHI

Graduate School of Information Science and Electrical Engineering, Kyushu University

Many services can be provided on a computer by improvement of computer performance. Also, each service requests different processing performance. Besides, one service is composed of many processes in many cases. Thus these processes need to be a unit for process scheduling. We proposed a mechanism of regulating execution performance for multi process by execution resource. In this paper, we report a result of an evaluation of our proposed mechanism by using Web server. We describe BSD/OS compatible system-call interface, and a result using the interface and Apache Web server. Also, we show that our proposed mechanism is able to regulate execution performance of multi process.

1 はじめに

計算機性能の向上により，計算機を利用して様々なサービスが提供されている．特に，クライアント/サーバ方式のサービスでは，サーバ計算機が多くの要求を同時に処理する必要があるため，サービスを複数プロセスで構成することが多い．例えば，Webサーバでは，サーバ側計算機で複数の子プロセスをあらかじめ用意し，クライアントからの要求毎に，子プロセスが処理を行う．また，一方では，計算機性能が向上したことにより，一つの計算機上で同時に複数のサービスを提供することが可能となっている．このとき，提供するサービス毎に重要度が異なるこ

とが考えられるため，重要なサービスについては，他のサービスの実行に関係なく，サービス品質を保証する必要がある．しかし，多くのオペレーティングシステム（以降，OS と略す）では，サービス毎の実行優先度を決定するために，サービスを構成する個々のプロセスの優先度を変更する必要があり，サービス毎の実行優先度の変更は容易ではない．また，サービスを構成するプロセスを生成する度に，実行優先度を設定する必要がある．したがって，サービスを構成するプロセス群を単位とし，サービスに対し品質を保証できることが望ましい．とりわけ，計算機資源の中でも，プログラムの実行に必要なプロセッサ時間の割当が重要である．

我々は，文献[1]において，***Tender***オペレーティングシステム[2]の資源「演算」[3]を利用し，複数プロセスからなるサービスを単位として処理時間を保証する方式を提案した．提案方式では,「演算」を木構造で管理し，プロセス群に対しプロセッサ時間を保証する．また，一つのプロセス群の中に，複数のサブプロセス群を作ることを可能にし，サブプロセス群を単位として，プロセッサ時間の割当を調整することを可能にしている．

本論文では，文献[1]で提案した方式を Web サーバを用いて評価した結果を報告する．評価には，多くの Web サーバで利用されている Apache を利用することとした．Apache を ***Tender*** 上で実行するために，BSD/OS 互換システムコールインタフェースを実現し，BSD/OS と比較評価した．さらに，***Tender*** 上で Apache Web サーバを用いて複数プロセスの実行速度調整機能を評価した結果を報告する．

2 *Tender* オペレーティングシステム

本章では，複数プロセスの実行速度調整機能を実現した***Tender***オペレーティングシステム[2]について簡単に説明する．

Tenderでは，OS の操作する対象を資源として，分離し独立化している．資源には，資源名と資源識別子を付与し，資源操作のインタフェースを統一している．さらに，各資源を操作するプログラム部品を資源毎に分離し，共有プログラムを排除している．また，各資源の管理情報も資源毎に分離し，各資源の管理表の間の参照関係を禁止している．

このように，資源の分離と独立化を行うことで，資源の事前生成や保留により，資源の作成や削除を伴う処理を高速化している．また，OS の動作や内部状態の理解や把握が容易になり，OS の理解を支援できる．さらに，プログラムを部品化できるため，機能の追加や変更が容易になっている．

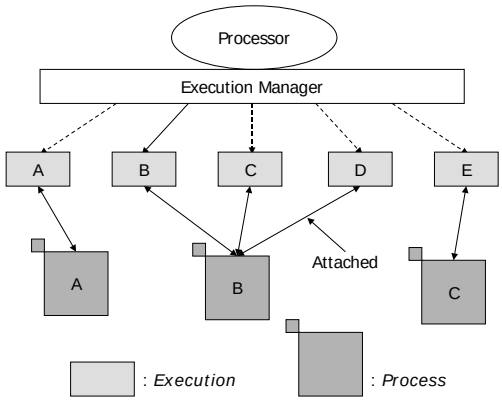

図 1 演算とプロセスの関係

3 資源「演算」を利用したプロセスグループの実行性能調整法

資源「演算」[3]とは，プロセッサの割当単位をプロセスから分離し資源化したものである．本章では，資源「演算」の基本機能と種類，およびプログラム実行速度調整法について簡単に説明する．

3.1 資源「演算」とは

3.1.1 基本機能

演算は，プロセッサを割り当てられる程度（以降，演算の程度と名付ける）を持つ．演算は，演算の程度により，割り当てられるプロセッサ時間が決定される．また，プロセスを走行させるには，演算とプロセスを関連付ける必要がある．演算とプロセスを関連付けることにより，演算に割り当てられたプロセッサ時間を利用して，当該演算に関連付けられたプロセスが走行する．

演算とプロセスの関係を図 1に示す．図 1に示したように，演算とプロセスの関連付けの関係は，$n:1$（n は任意の正の整数）である．つまり，複数の演算を一つのプロセスに関連付けることができる．また，異なる種類の演算を，同時に一つのプロセスに関連付けることができる．演算管理には，八つのインタフェースがある．このうち，演算とプロセスの関連付けに関するものを表 1に示す．

3.1.2 種類

演算には，性能調整の演算と優先度の演算がある．性能調整の演算が持つ演算の程度は，プログラムの実行速度を調整する性能（1〜100%，プロセッサそのものの性能を 100%とする）を示す．性能調整の演

表 1 演算管理インタフェース

通番	形式	機能
1	attach_execution(execid, pid)	演算 execid とプロセス pid を関連付ける.
2	detach_execution(execid, pid)	演算 execid とプロセス pid の関連付けを解除する.

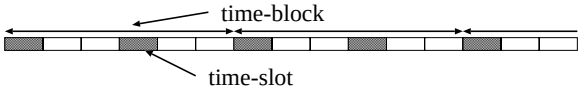

図 2 タイムスロットとタイムブロックの関係

算が持つ演算の程度の総計が 100%以下となる範囲で，性能調整の演算を生成できる．優先度の演算が持つ演算の程度は，スケジューリングの優先順位となる優先度を示す.

3.1.3 プログラム実行速度調整法

プログラム実行速度の調整[3]は，ある単位時間（これをタイムスロットと名付ける）で，プログラムの実行と停止を繰り返すことで実現できる．一定の連続したタイムスロットをタイムブロックと名付ける．タイムスロットとタイムブロックの関係を図 2に示す．性能調整の演算は，一つのタイムブロックの中から，演算の程度に見合う割合（%）のタイムスロットを割り当てられる．割り当てられたタイムスロットの分だけ，性能調整の演算に関連付けられたプロセスを走行させることで，実行速度を調整する．

3.2 複数プロセスの実行性能調整機能

文献[1]で提案した複数プロセスの実行性能調整機能について説明する．

3.2.1 実現方式

複数プロセスのプログラム実行性能調整機能の実現方式を以下に述べる．

演算を利用して，一つ以上のプロセスの集まりをプロセスグループとして管理する．具体的には，演算の操作しやすさを考慮し，演算の木構造（演算木と名付ける）でプロセスグループを管理することとした．この様子を図 3 に示す．演算木の根はプロセッサを表す．演算木の葉をリーフ演算と名付ける．上記二つ以外の演算をディレクトリ演算と名付ける．リーフ演算は，プロセスに関連付けられる．

ここで，以降では，演算木を構成するプロセッサと演算，および演算と演算を結びつけることを「連結する」と呼ぶ．また，連結されたプロセッサと演算，および演算と演算の結びつきを切ることを「切り離す」と呼ぶ．プロセッサの下には，性能調整の演算と優先度の演算を連結できる．もちろん，これらの演算は，ディレクトリ演算またはリーフ演算で

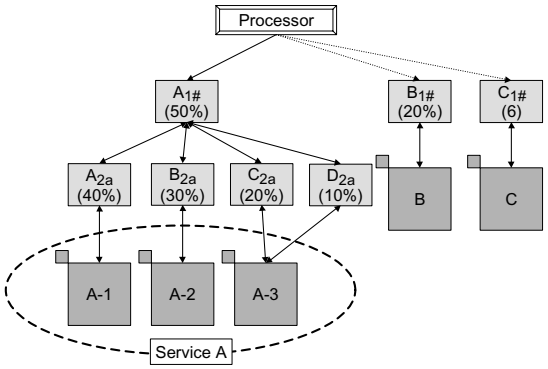

図 3 演算によるプロセスグループの表現

ある．プロセッサの下に連結された性能調整の演算が持つ演算の程度は，プロセッサそのものの性能に対する割合を示す．このため，プロセッサの下，およびディレクトリ演算に連結できる性能調整の演算が持つ演算の程度の合計は，100 %以下に制限される．また，優先度の演算は，性能調整の演算が必要としない残りのプロセッサ時間を利用するため，プロセッサの下に連結できる数に制限はない．例えば，図 3 では，ディレクトリ演算 $A_{1\#}$は，プロセッサそのものの性能の 50 %を割り当てられる．リーフ演算 A_{2a}には，ディレクトリ演算 $A_{1\#}$の 40%の性能が割り当てられる．したがって，リーフ演算 A_{2a}は，プロセッサそのものの性能のうち 20%(=50%× 40%)を割り当てられる．

次に，プロセスグループに対し複数の演算を同時に関連付けるために，一つのプロセスグループに対し，複数のディレクトリ演算を関連付けることとした．この様子を図 4 に示す．図 4 では，ディレクトリ演算 $A_{1\#}$と $B_{1\#}$をプロセスグループ（サービス A）に関連付けている．具体的には，ディレクトリ演算 A には，リーフ演算 A_{2a}，B_{2a}，C_{2a}が連結されており，ディレクトリ演算 B には，リーフ演算 D_{2b}，E_{2b}，F_{2b}が連結されている．また，プロセス A-1，A-2，A-3 には，ディレクトリ演算 $A_{1\#}$に連結されたリーフ演算と，ディレクトリ演算 $B_{1\#}$に連結されたリーフ演算が関連付けられている．

最後に，プロセスグループ内に複数のプロセスグループを構成するために，ディレクトリ演算の下に，

表 2 演算管理インタフェース（演算の階層化対応）

通番	形式	機能
1’	attach_execution (execid, rid)	rid はプロセス識別子または演算識別子を示す.（1）rid がプロセス識別子のとき，演算 execid をプロセス rid に関連付ける．ただし，演算 execid がディレクトリ演算のとき，エラーとする.（2） rid が演算識別子のとき，ディレクトリ演算 rid に，演算 execid を連結する．ただし，ディレクトリ演算内で性能 mips が確保できないとき，エラーとする．
2’	detach_execution (execid, rid)	rid はプロセス識別子または演算識別子を示す.（1）rid がプロセス識別子のとき，演算 execid とプロセス pid の関連付けを解除する.（2）rid が演算識別子のとき，ディレクトリ演算 execid と演算 rid を切り放し，演算 rid は演算木の根（プロセッサ）に連結される．ただし，連結後に性能 mips が確保できないとき，エラーとする．

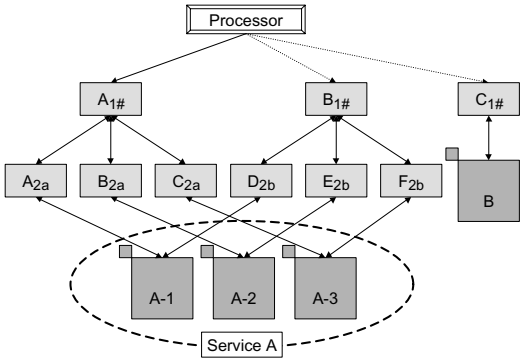

図 4 プロセスグループへの複数演算の関連付け

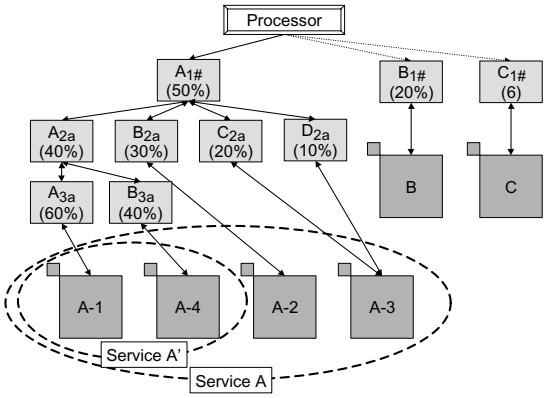

図 5 演算木の階層化によるサブプロセスグループの実現

ディレクトリ演算を連結することとした．つまり，深さが 2 以上の演算木を構築可能とした．この様子を図 5 に示す．図 5 では，ディレクトリ演算 $A_{1\#}$にディレクトリ演算 A_{2a}を連結し，サービス A のサブプロセスグループとなるサービス A’ を構築している．これに伴い，ディレクトリ演算の下にディレクトリ演算が存在する場合の演算の程度は，先に述べたリーフ演算が存在する場合を拡張する考え方で実現した．例えば，図 5 では，ディレクトリ演算 A_{2a}は，ディレクトリ演算 $A_{1\#}$の 40%の性能を割り当てられるため，プロセッサそのものの性能の 20%(=50%× 40%) を割り当てられる．したがって，リーフ演算 A_{3a}は，ディレクトリ演算 A_{2a}の 60%の性能を割り当てられるため，プロセッサそのものの性能の 12%(=50%× 40%× 60%) を割り当てられる．

3.2.2 提供インタフェース

演算木の導入に伴い，生成された演算は，最初は演算木のプロセッサ（根）に連結されることとした．また，プロセスグループの実行性能調整を可能にする演算木を構築するために，演算をプロセッサから切り離し，ディレクトリ演算に連結する機能，および演算をディレクトリ演算から切り離す機能を用意した．これらの機能を提供するため，表 1 に示したインタフェース attach_execution と detach_execution の機能を拡張した．拡張したインタフェースを表 2に示す．

3.2.3 期待される効果

プロセスグループの実行性能調整法を実現することで，以下の三つの効果が期待できる．

(1) プロセスグループの実行速度の調整
性能調整の演算をプロセスグループに関連付け，演算の程度を変更することにより，プロセスグループを構成するすべてのプロセスの実行速度を一度に調整できる．

(2) プロセスグループの処理時間の保証
性能調整の演算をプロセスグループに関連付けることにより，プロセスグループに対しプロセッサ性能を保証できる．さらに，優先度の演算を同時に関連付けることにより，プロセッサ

表 3 システムコールの分類

分類	システムコール名
機能を完全に実現するシステムコール	accept, bind, close, connect, dup2, exit, fork, getdirentries, getdtablesize, getpid, getsockname, listen, lseek, read, recvfrom, sendto, setsockopt, shutdown, sigaction, sigprocmask, sigreturn, socket, unlink, write, writev（計 25 個）
機能の一部を実現するシステムコール	__sysctl, break, fcntl, fstat, gettimeofday, kill, lstat, open, select, setitimer, stat, wait4（計 12 個）
成功の値のみを返すシステムコール	access, chdir, fstatfs, geteuid, setgid, setgroups, setsid, setuid, umask（計 9 個）

の空き時間をプロセスグループが利用できる．

(3) サービスの処理内容に合わせた自由なプロセッサ割当の実現
演算木の導入により，複数プロセスをプロセスグループとしてグループ化できる．さらに，一つのプロセスグループの中に，複数のサブプロセスグループを構築することを可能にしている．

4 Web サーバによる評価

4.1 *Tender*への異種 OS 共存手法

4.1.1 BSD/OS 互換システムコールインタフェースの実現

提案手法を既存 OS でも利用されている Web サーバで評価するために，***Tender***に BSD/OS のシステムコールインタフェースを実装し，***Tender***上で Apache Web サーバを実行できるようにした．本項では，***Tender***への BSD/OS 互換システムコールインタフェースの実現方式について述べる．

OS インタフェース共存に対する要求条件として，AP のソースコードを修正せず，実行ファイルをそのまま利用できることがある．これは，既存 OS と同等の AP で評価するために必要となる．この要求を満たすために，以下の対処を行う．

(1) 実行ファイルの形式をサポートする．
(2) BSD/OS 互換システムコールインタフェース（以降，I/F と略す）を実現する．
(3) 対象とする AP で必要となる処理のみを実現する．

BSD/OS 互換システムコール I/F を***Tender***に実装した様子を図 6に示す．***Tender***では，BSD/OS 3.1 の実行形式（a.out）をサポートしているため，BSD/OS プログラムから，プロセスを生成し実行することができる．また，BSD/OS システムコール処理関数を新たに***Tender***に実装した．これにより，カーネルは BSD/OS AP の発行するシステムコール

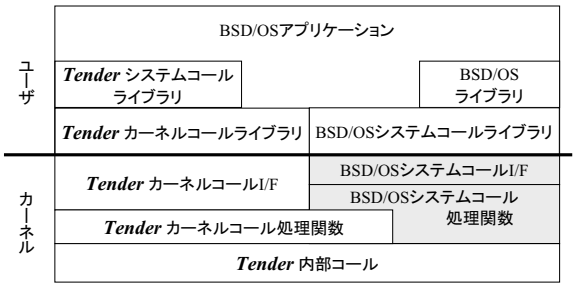

図 6 ***Tender*** と BSD/OS AP の関係

番号と引数を取得することができる．また，***Tender***カーネルコール I/F は，これとは独立して存在する．

BSD/OS システムコール処理関数では，発行された BSD/OS のシステムコールに相当する処理を実行し，戻り値を返す．この処理は，***Tender***の内部コールやカーネルコール処理関数を利用して実現している．

4.1.2 Apache で利用するシステムコール処理関数の実現

実装するシステムコールを決定するために，Web サーバの動作に必要なシステムコールを調査した．具体的には，BSD/OS 上で Web サーバを動作させ，クライアントからの要求を受け，ファイルをクライアントに送信する処理のログを取った．この結果から，システムコール名と利用している機能を明らかにした．

調査の結果，Web サーバが呼び出したシステムコールは 46 個であった．なお，BSD/OS の全システムコール数は 152 個である．つまり，全体の約三分の一のシステムコールを実装すれば良い．

実装するシステムコール 46 個を，システムコールの機能や返却情報を全て利用するもの，システムコールの機能や返却情報の一部を利用しているもの，およびシステムコールの成功かどうかの返り値しか利用してないものに分類した．分類した結果を表 3 に示す．この分類により，完全な実現が必要なシステムコールは約半分の 25 個となり，実装にかかる工

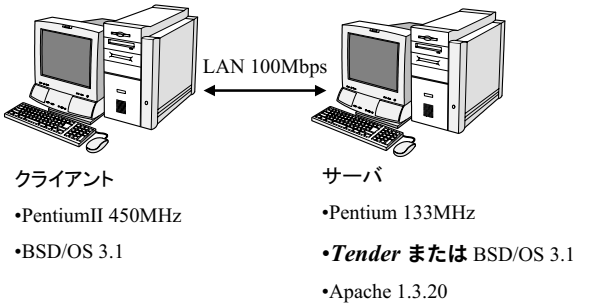

図 7 Web サーバの性能測定環境

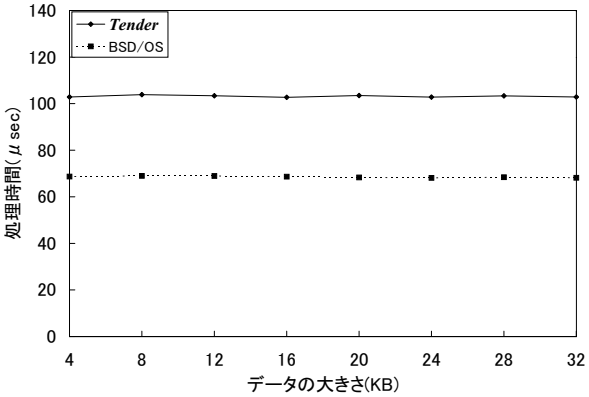

図 8 open システムコールの処理時間

数を大幅に削減できる．

実装するシステムコールには，**Tender**の機能を利用して実現するものと，BSD/OS の実装を参考にして新規に実装するものがある．前者として，プロセスの生成，ファイルに関するシステムコールである fork，exit，open，close，read，write などがある．後者として，connect，accept，dup2 などがある．

4.2 異種 OS インタフェースの基本評価

4.2.1 測定環境

システムコールの性能測定は，Pentium（133MHz）の計算機を用いて行った．Web サーバの性能測定は，図 7に示す環境で行った．どちらの測定も，比較のため，**Tender**と BSD/OS 3.1 両方で測定を行った．なお，測定は，**Tender**，BSD/OS 共に，シングルユーザモードで行った．測定には，ハードウェアクロックカウンタを用いた．

4.2.2 システムコールの性能評価

システムコールを 10000 回繰り返して呼び出し，その平均の処理時間を測定した．**Tender**での実装において，BSD/OS の実現方式を参考に実現したシステムコールでは，BSD/OS との性能の差があまりないと考えられるため，**Tender**の機能を利用して実現したシステムコールの性能を測定した．具体的には，open，close，read，および write システムコールを利用して評価した．これらのシステムコールの実現には，既存の OS でのファイルシステムに相当する**Tender**の資源「プレート」[4]を利用している．

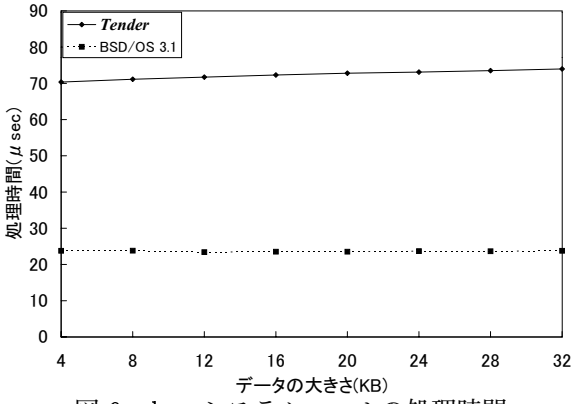

図 9 close システムコールの処理時間

open システムコールの測定は，オープンするデータの大きさを変えて行った．open システムコールの測定結果を図 8に示す．open システムコールについては，**Tender**と BSD/OS 共にデータの大きさに関係なく処理時間に変化はないことがわかる．これは，open の処理内容が**Tender**と BSD/OS が共に，データへの参照数のインクリメントのみを行っているためである．open システムコールについては，約 35 マイクロ秒**Tender**の処理時間の方が長い．これは，オープンするデータを検索する際，BSD/OS ではキャッシュを用い同じ名前のデータが指定された場合の検索を高速化しているためである．

close システムコールの測定は，データの大きさを変えて行った．close の測定結果を図 9に示す．close システムコールについては，BSD/OS では処理時間は一定であるが，**Tender**ではデータの大きさに比例して処理時間が増加する．これは，**Tender**では close に相当する処理を行う際，ディスクに書き出す必要があるかをチェックするためである．

read システムコールと write システムコールについて，読み書きするデータの大きさを変えて測定を行った．

read システムコールの測定結果を図 10に，write システムコールの測定結果を図 11に示す．read システムコールについては約 33〜53 マイクロ秒，**Tender**の処理時間の方が短い．write システムコールについては約 38〜65 マイクロ秒**Tender**の処理時間の方が短い．これは，BSD/OS では，ファイルシステムのブロックを意識して読み込みと書き出しを行っているのに対し，**Tender**では，メモリの読み込みと

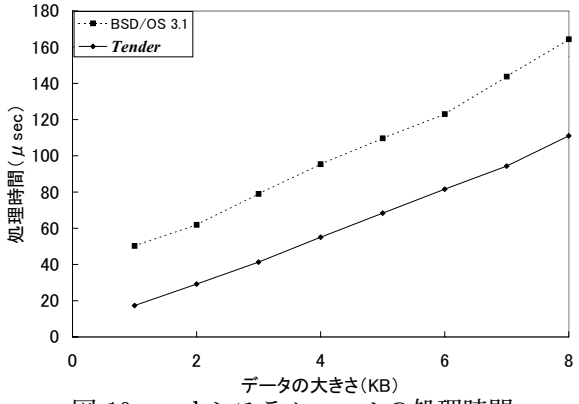

図 10　read システムコールの処理時間

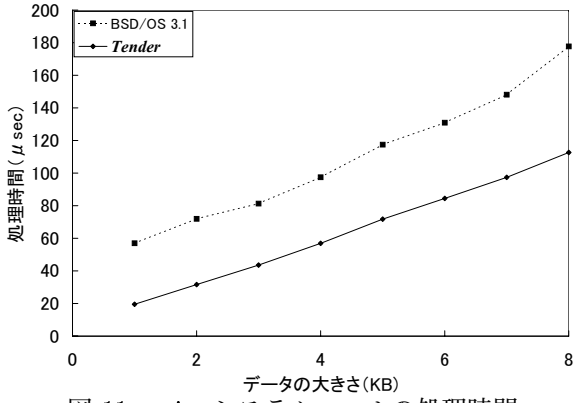

図 11　write システムコールの処理時間

書き出しのみ行えば良いためである．

評価結果より，1 度の open と close の間に，1 回か 2 回しか read や write を呼び出さないような AP の場合は BSD/OS に性能が劣るが，read や write を頻繁に呼び出す AP では ***Tender*** の方が性能が良いと考えられる．

4.3　Web サーバの性能評価

測定は，クライアントの計算機が，サーバの計算機の Web サーバに，データの取得の要求を出し，データの取得が完了するまでの時間を，100 回ずつ 5 回，計 500 回測定しその平均時間を求めた．取得するデータのサイズを変え測定を行った．

測定結果を図 12に示す．図 12から，***Tender*** と BSD/OS での測定結果にほとんど差のないことがわかる．このことから，本論文で述べた異種 OS インタフェースの実装によるオーバヘッドはかなり小さいといえる．

4.4　プロセスグループの実行性能調整法の評価

文献[1]で提案した方式の有効性を明らかにするため，Web サーバを用いて評価した．図 7に示す環境で測定した．サーバ側では，***Tender*** 上で Web サーバ（Apache 1.3.23）を起動し，クライアント側では，

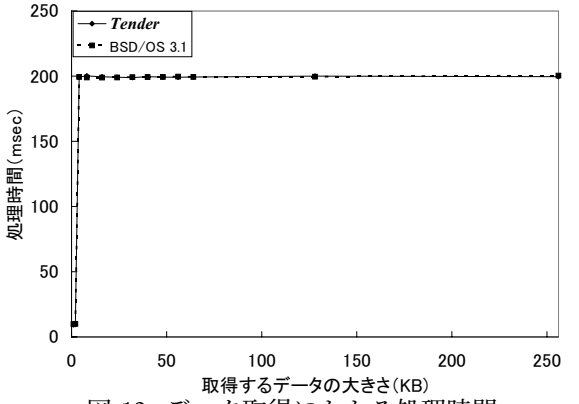

図 12　データ取得にかかる処理時間

BSD/OS 3.1 上で，サーバ側計算機に用意した九州大学の Web ページ（2001 年 2 月 19 日のもの）に一度だけアクセスするプログラムを 20 個同時に実行した．Apache に性能調整の演算を 1 個関連付け，その演算が持つ演算の程度を変更し，クライアント側計算機で，Web サーバからの応答時間の平均を求めた．なお，Apache の各プロセスには，同一優先度を持つ優先度の演算を関連付けた．サーバ側計算機では，Web サーバ以外に優先度の演算を関連付けられたプロセスを 1 個走行させた．このプロセスはプロセッサ処理のみを行う．測定結果を図 13に示す．

図 13から，以下のことがわかる．

(1) 演算の程度が 100%から 50%までは，Web サーバからの応答時間の変化が小さいことがわかる．これは，サーバ側計算機でのプロセッサ使用率が低いので，プロセッサ時間の割当を調整した結果が，応答時間に反映されにくいためである．

(2) 演算の程度が 50%以下の場合，指定された演算の程度の割合にしたがって，処理時間が変化していることがわかる．

この結果から，ある一定割合のプロセッサ時間を Web サーバに保証することで，プロセッサのボトルネックによるサービス品質の低下を抑止することができると考えられる．そこで，10%の演算の程度を持つ性能調整の演算 1 個と優先度 6 を持つ優先度演算 1 個を Apache に関連付けした場合について，サービス品質を保証できるのかを評価した．評価では，他プロセスの数を変化させ，先の測定と同様に 20 個のクライアントプログラムを実行し，Web サーバからの応答時間の平均を求めた．他プロセスには，Apache プロセスと同一優先度を持つ優先度の演算 1 個を関連付けた．他プロセスが，プロセッサ処理のみを行う場合と，プロセッサ処理 1 秒と WAIT 状態 1 秒を

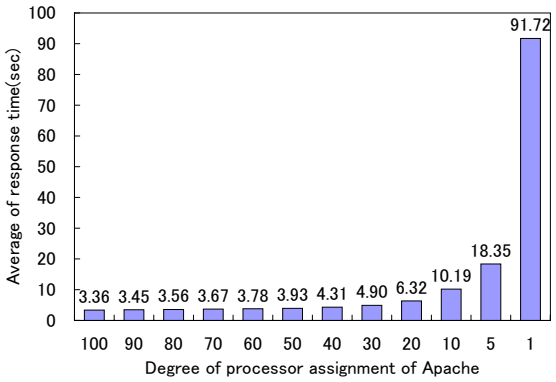

図 13　Apache に関連付けたディレクトリ演算が持つ演算の程度とクライアントプログラムの平均応答時間の関係

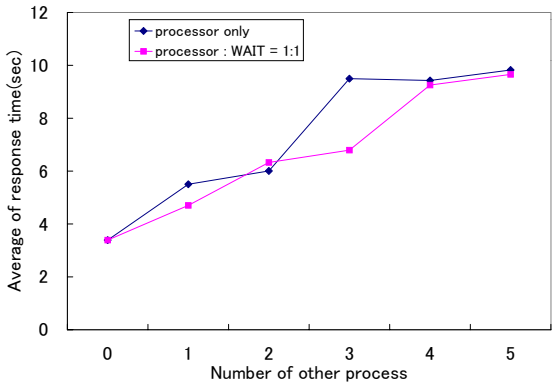

図 14　他プロセス数とクライアントプログラムの平均応答時間の関係

繰り返す場合について測定した．測定結果を図 14に示す．

図 14から，以下のことがわかる．

(1) 他プロセスが存在しない場合，応答時間の平均（他プロセス数 0 個，3.39 秒）は，図 13で 100%の性能を持つ演算を関連付けた場合（3.36 秒）とほぼ等しい．

(2) 他プロセスのプロセッサ使用率が低いほど，応答時間は短い傾向がある．

(3) 他プロセスのプロセッサ使用率が増加すると，応答時間は 10%の性能を持つ演算のみを関連付けた場合の応答時間（10.19 秒）に近づく．ただし，10%のプロセッサ時間を保証されているため，応答時間が 10.19 秒を越えていない．

以上のことから，性能調整の演算をサービスに関連付けることで，サービスの応答時間を調整できることを示した．また，性能調整の演算と優先度の演算を同時にサービスに関連付けることで，他プロセスがプロセッサを利用しない場合には，100%のプロセッサを使用でき，他プロセスがプロセッサを利用する場合でも，性能調整の演算が持つ性能の割合分のプロセッサ時間を保証でき，Web サーバの応答時間を保証できることを示した．

5　おわりに

本論文では，Web サーバを利用して，***Tender***におけるプロセッサの割当単位である資源「演算」を利用した，複数プロセスの実行性能調整機能を評価した．複数プロセスの実行性能調整機能は，一つのプロセスグループ内に複数のサブプロセスグループが存在させることができ，そのサブプロセスグループ単位で実行性能を調整できる．なお，演算には，プログラムの実行速度を調整できる性能調整の演算と，実行の優先順位を示す優先度の演算がある．このため，これらの演算をうまく込み合わせることで，ハードウェア性能に余裕があるときは多量の処理を可能にし，余裕が無いときは処理量の最低保証を行うことができる．また，Web サーバを***Tender***で実行するために実現した BSD/OS 互換システムコールインタフェースの実現方式について述べた．Web サーバを用いた評価では，Apache Web サーバが BSD/OS と同等の応答時間で要求を処理できることを示した．さらに，提案手法が，Web サーバプロセスの実行性能を調整し，Web サーバの応答時間を保証できることを示した．

残された課題として，調整された性能の調整精度の評価，多様なサービスでの評価がある．

参考文献

[1] 田端利宏, 谷口秀夫: ***Tender***における資源「演算」を利用したプロセスグループの実行速度調整法, 情報処理学会研究報告, Vol.2001, No.78, pp.3–10 (2001).

[2] 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, 情報処理学会論文誌, Vol.41, No.12, pp.3363–3374 (2000).

[3] 田端利宏, 谷口秀夫: ***Tender***オペレーティングシステムにおける資源「演算」を用いたサービス処理時間の保証, 情報処理学会論文誌, Vol.41, No.6, pp.1745–1754 (2000).

[4] 稲本慎司, 谷口秀夫: ***Tender***においてメモリ上のデータを永続化する資源「プレート」の復元機構, 情報処理学会第 63 回全国大会講演論文集（分冊 1）, pp.85–86 (2001).