NUMAマシンでのコマーシャルワークロード向けLinux最適化

平井 聡[†] 山本昌生[†] 佐藤 充[†]
成瀬 彰[†] 久門耕一[†]

本論文では，NUMA（Non Uniform Memory Access）マシンで Commercial Workload を実行する際の Linux カーネルの最適化実験について述べる．評価システムには，2 ノード 4 プロセッサの小規模な NUMA マシンの実機を使用し，性能の検証には WEB サーバとアプリケーションサーバをモデル化した 2 種類のベンチマークを用いた．また，カーネルプロファイラによる OS 動作の関数レベルでの実測，およびハードウェア・バストレーサによるメモリアクセスの実測を行い，動作分析を行った．実験の結果，これらのベンチマークプログラムでは NUMA マシンのオーバヘッドは主に OS 部に依存していることが分かり，メモリアクセス局所化により OS 実行時間を 18%〜20%削減し，NUMA オーバヘッドを 1/3〜2/3 に削減した．

Linux Optimization for Commercial Workload on NUMA Machine

Akira Hirai,[†] Masao Yamamoto,[†] Mitsuru Sato,[†]
Akira Naruse[†] and Kouichi Kumon[†]

In this paper, we describe the experimental optimization of Linux kernel on a NUMA machine for commercial workloads. For the evaluation, we used a small-scale NUMA machine which consists of two nodes total of four processors. Two kinds of benchmark programs were used for the measurements, each models WEB server execution or application server execution. We measured the OS function execution timings by a kernel profiler and also measured the memory access statistics by a hardware bus tracer. By using these data, we analyzed the execution characteristics of the programs. The experimental results showed the overhead of the NUMA machine is mainly in OS, and the OS execution time can be reduced 18%–20% by the memory access localization for these benchmark programs.

1. はじめに

共有メモリ型並列計算機の形態の 1 つである NUMA（Non Uniform Memory Access）はその拡張性の高さ，大規模システム構成時のコストの安さなどから将来の大規模並列システム向けとして期待されている構成方式である．また近年，IA プロセッサを用いた PC サーバにおいても大規模システム構築の需要が高まっており，コスト，拡張性，開発期間などの面から NUMA システムが注目されている．NUMA システムの用途は，科学技術計算だけでなく，Web サーバ，OLTP，DSS（ERP）などいわゆる商用アプリケーション処理に広がると考えられ，この分野における動作分析および性能向上手法が必要となっている．これらの一般的な商用アプリケーションには以下のような特徴がある．

- 科学技術計算分野のアプリケーションに比べて，OS 動作比率が高い．
- ソフトウェアがバイナリで提供される，あるいはユーザがコンパイルを意識せずにセットアップできるインストール・ツールが提供される．

商用 NUMA システムとしては，SGI の Origin[1)]，Sequent（IBM）の NUMA-Q[2)] などがあり，実験用システムとしては Stanford 大の DASH[3)]，FLASH[4)]，Sun の WildFire[5)] などが知られているが，Commercial Workload に関する分析や最適化を行った報告例は少ない．WildFire では，実機を使用し，OLTP（TPC ベンチマーク）を用いた最適化評価を行っているが，性能向上の中心は，ハードウェア・サポートによるメモリのレプリケーションやマイグレーションによる最適化である[5)]．

このような点をふまえ，我々は，特殊なハードウェア・サポートなしに，またアプリケーションの変更なしに，オペレーティング・システムの最適化のみでいかに性能が向上するかの実験を行った．本論文では，

† 株式会社富士通研究所
Fujitsu Laboratories, LTD.

表 1 サーバ機 — GRANPOWER5000（Model880）
Table 1 Server system.

CPU	Pentium-II Xeon 450 MHz × 4 （NUMA 構成では各筐体 2CPU）
Memory ,MB）	1 GB（NUMA 構成では各筐体 512
HDD	内蔵 SCSI 9 GB × 1
LAN	Intel 社 EtherExpress Pro/100 × 4 （NUMA 構成では各筐体 2 枚）

表 2 クライアント機 — PC-AT 互換機 28 台
Table 2 Client system.

CPU	Pentium-III 800 MHz × 4 Pentium-III 783 MHz × 8 Celeron 500 MHz × 8 Celeron 333 MHz × 8
Memory	128 MB
OS	WindowsNT Workstation4.0（SP5）

2 章で評価システムの説明および UMA システムと NUMA システムの性能差について述べる．3 章では，Linux カーネルの NUMA 向け最適化手法について述べる．4 章では性能検証の結果を分析し，5 章でまとめる．

2. UMA と NUMA の性能比較

NUMA システムは UMA システムに対してどの程度のオーバヘッドがあるのか，またどのような特徴があるのかを調べるため，NUMA 機とその構成要素である UMA 機の双方で測定を行うことにより，Commercial Workload での性能比較を行った．

2.1 評価環境

評価システムのサーバ機を**表 1** に，クライアント機を**表 2** に，環境構成図を**図 1**（UMA）および**図 2**（NUMA）に示す．サーバ機の GRANPOWER5000 model880[6] は，2 台の 4way（UMA）構成サーバを 1.6 GB/s のインターコネクト技術「Synfinity-1」[7] で結合することにより 8way（NUMA）構成システムを構成する．本評価では，UMA 構成時には片側の筐体のみを 4way で使用し，NUMA 構成時には 1 筐体につき 2CPU 構成の 4way で使用し測定を行った．

I/O 資源に関しては，本評価で使用するアプリケーション（ベンチマークテスト）でサーバ CPU が飽和する負荷を考慮して構成した．LAN 構成に関しては，100 Mbit Ethernet Card を 4 枚使用し（NUMA 構成では各 Node に 2 枚ずつ），Disk 構成に関しては，1 台のみ（NUMA 構成では片側 Node に 1 台のみ）接続した．

本サーバ機は NUMA 構成における I/O 資源に関

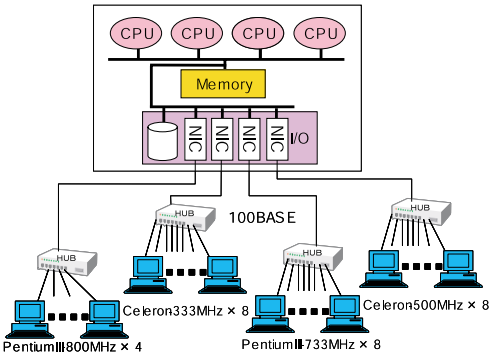


図 1 環境構成図（UMA）
Fig. 1 System organization for evaluation (UMA).

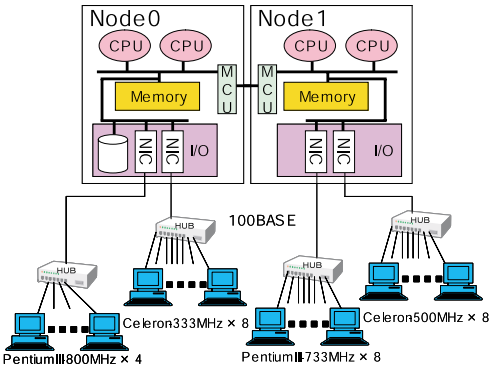


図 2 環境構成図（NUMA）
Fig. 2 System organization for evaluation (NUMA).

しても双方の Node の機器を透過的にアクセス可能である．また本評価においては，各 I/O 装置，バスともにボトルネックとなるような負荷には至っておらず，UMA 構成時と NUMA 構成時のバスのトータル・スループットの差（NUMA 構成時は倍）による性能への影響はない．なお，NUMA 構成時のローカルとリモートのメモリアクセス・レイテンシ比は，およそ 1 : 3 であり NUMA 構成時のオーバヘッドは基本的にこの部分に起因する．

2.2 サーバ機 OS

サーバ機には GNU-Linux システム（RedHat Linux 6.1—US 版）を使用した．カーネルに関しては，開発版カーネル「2.4.0-test1」をベースとし，UMA システムではコード無修正で，NUMA システムでは以下の修正を行った☆．

(1) NUMA システム固有仕様への対応
プロセッサ管理，Node 間割込み，Node 間 I/O 処理など．

☆ ソースコード変更量：23 ファイル（850 行）．

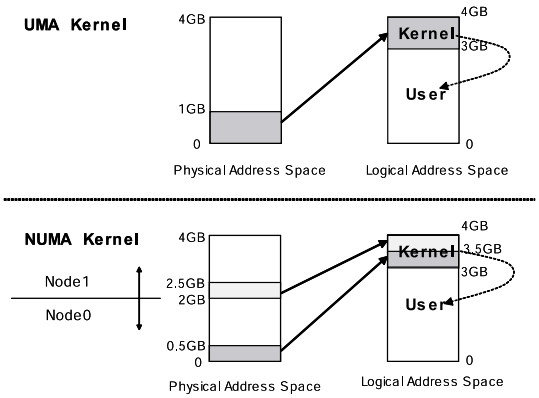


図 3 カーネル・メモリ・マップ
Fig. 3 Kernel memory map.

(2) NUMA システム固有メモリマップへの対応
Linux カーネルは，**図 3** のように物理メモリ 0〜最大 1 GB をカーネル領域として論理アドレス 3 GB 以降のアドレスにマップしている．この領域は，すべてのプロセスに共通であり，さらにこの領域の中から必要に応じてプロセスごとにユーザ領域としてマップする．
GRANPOWER5000（Model880）では，Node 0 が物理アドレス空間 0〜2 GB，Node1 が 2〜4 GB に割り当てられるため，そのままでは 1 GB 以下のカーネル領域はすべて Node 0 のメモリとなってしまう．このため，カーネルのメモリ管理を修正し，物理アドレス 0〜0.5 GB および 2〜2.5 GB を連続の論理アドレス空間としてマップするよう修正した．

(3) ページ単位メモリ管理の Node 対応
ページ単位のメモリ確保要求が行われた場合に，要求元 CPU の Node に合わせて物理ページの確保を行うようにメモリ管理の Node 対応を行った．

2.3 ベンチマークテスト

Commercial Workload に基づいたベンチマークテストとして，Ziff Davis 社が提供している 2 種類のベンチマークプログラム「WebBench 3.0」[8] および「ServerBench 4.1」[9] を用いた．

WebBench は Web サーバアプリケーションの性能評価を行うベンチマークであり，ServerBench は一般的なアプリケーションサーバを模したベンチマークである．特徴的な違いは，**図 4** のように USER 部の実行比率と OS 部の実行比率が対照的なことである．

2.3.1 WebBench

サーバ側は一般的な Web サーバアプリケーションのみを使用し，クライアント側で動作するベンチマーク・プログラムの要求に従い，Web ページ（画像ファイルなど）を送信する．本測定では，Web サーバとして「Apache 1.3.9」を使用した．

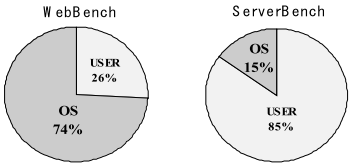


図 4 USER/OS 実行時間比率（UMA 4CPU）
Fig. 4 USER/OS runtime ratio.

クライアント側は，1 クライアントプロセスあたり 2 スレッドでサーバへのリクエストを行う設定とし，最大でクライアント 28 台（32 クライアントプロセス）から 64 リクエストを同時発行する設定とした．また測定は “static get” のみを行った（Mindcraft benchmark Report [10] に準拠した測定．ただし，Disk 容量の都合上アクセス・ログファイルの採取は行っていない）．

特徴

- サーバ側プログラムの Apache は，クライアント（スレッド）からのコネクト要求に対応したプロセスを 1 対 1 で生成する．このため，ベンチマーク測定中は，最大で 64 プロセスが同時動作する．
- サーバメモリ 1 GB での測定で，ベンチマーク値測定時にファイルはすべてファイル・キャッシュに載る．この結果，ディスクアクセスはほとんどなく，ネットワーク処理が主である．
- OS 実行時間比率が高い．

2.3.2 ServerBench 4.1

サーバ側はベンチマーク専用プログラムを動作させる．クライアント側のベンチマーク・プログラムと協調し，一般的アプリケーションサーバ動作をシミュレートしており，以下の種類のテストを組み合わせている．

(1) Processor test（ファイルおよびネットワークアクセスをともなわない）
(2) Disk test（Sequential R/W，Random R/W）
(3) Network test（Server→Client，Client→Server）

クライアント側は 16 台のみ使用し，16 クライアントプロセス（1 クライアントプロセスあたり 1 スレッド）を動作させた．測定テストセットは標準添付の “standard system test suite” を使用した．

特徴

- サーバ側プログラムは，テスト開始時にクライアント（スレッド）数に応じたプロセスを 1 対 1 で生成する．このため，ベンチマーク測定中は，常時 16 プロセスが同時動作する．

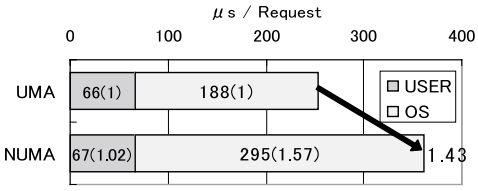


図 5 WebBench 性能
（カッコ内は UMA を 1 としたときの比率）
Fig. 5 Performance of WebBench.

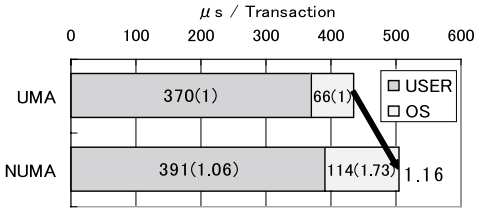


図 6 ServerBench 性能
（カッコ内は UMA を 1 としたときの比率）
Fig. 6 Performance of ServerBench.

- 1 台のクライアントにつき 16 MB のファイルを専有してアクセス．
- USER 比率が高い．OS 実行時間としては，ファイルアクセス処理が多い．

なお，各々のベンチマークのクライアント台数，プロセス数は，CPU 負荷率が 100%（Idle が 0%）で性能が最も高い台数に設定した．

2.4 性　　能

各々のベンチマークにおける性能（1 処理単位あたりの実行時間：WebBench は 1 リクエストあたり，ServerBench は 1 トランザクションあたり）を**図 5**，**図 6** に示す．

UMA と NUMA の性能を比較すると，WebBench で 43%，ServerBench で 16%実行時間が増加している．しかし，実行時間の USER/OS 内訳を分析してみると，双方のベンチマークともに USER 部の実行時間の増加に比べて，OS 部は非常に高い割合で実行時間が増加している．このことから，これらのプログラムにおける UMA と NUMA の性能差（これを NUMA オーバヘッドと定義する）は OS 内部の動作に起因しており，OS チューニングにより NUMA オーバヘッドを減少させることが可能であることが分かる．

3. Linux カーネルの最適化

NUMA システムの OS 部のオーバヘッドを削減するために，Linux カーネルの最適化を行った．

今回実施した最適化の基本方針は，メモリアクセスの局所化である．プロセッサ内のキャッシュ効率を高めること，また同一 Node 内でのメモリアクセス頻度を高めることで，他 Node へのメモリアクセスを減らすことを主眼とした．また，最適化手法において，コード領域やデータ領域のメモリレプリケーションといった NUMA システム固有の最適化ではなく，UMA と NUMA の双方に適用可能で，かつ相互に性能劣化を起こさない手法を選択した．これは，以下の理由による．

- 本評価でのベンチマーク測定において，NUMA 構成時のインストラクション・キャッシュ・ミス率（L2 キャッシュ・インストラクション・フェッチ・ミス数/実行命令数）が双方ののベンチマークともに 0.01%以下であり，コード領域に関してはメモリレプリケーションの効果が見込めない．
- UMA にも適用可能な一般的な最適化手法で NUMA システムのオーバヘッドをどこまで削減でき，またどの部分に固有の最適化が必要かを検証する．
- UMA にも適用可能な一般的手法を用いることで，Linux コミュニティへの成果のフィードバックが行いやすく，Linux カーネル本体への取り込みが期待できる．

本最適化では大別して，短期的なメモリアクセスの局所化である「メモリ管理の最適化」と，長期的なメモリアクセスの局所化である「スケジューラの最適化」の 2 つを行った．以下にそれぞれの手法について述べる．

3.1 メモリ管理の最適化

一般的なスケジューラにおいて，プロセスはタスクスイッチにより CPU 間，Node 間を遷移する．このため，長期的に割り当てられているユーザ領域のメモリやファイル・キャッシュなどは 2Node システムでは 50%の Local Node メモリアクセスしか見込めない．しかし，OS サービスルーチンや割込み処理など短期間で頻繁にメモリの割当て/開放が行われる処理に関しては，CPU と同一 Node のメモリを割り当て，また再利用させることでメモリアクセスの局所化が可能である．これを実現するため，Linux のメモリ管理部に機能追加を行った．

Linux のメモリ管理は，**図 7** のような構造をとっている．

「1. Page Allocator」は，物理メモリをページ単位で管理する機構であり，下位構造のページング・システムやカーネル・メモリ・アロケータの要求に応じてページ単位でメモリを切り出す．

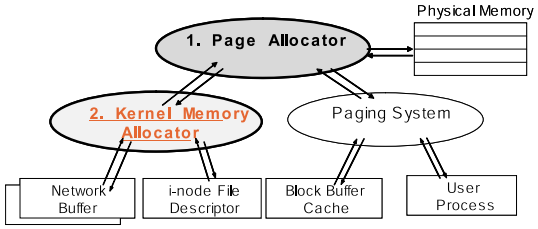


図 7 メモリアロケータ概念図
Fig. 7 Abstract view of Linux Memory Allocator.

「2. Kernel Memory Allocator」は OS 内部で使用されるメモリを管理する機構であり，ページ・レベル・アロケータから切り出したメモリをデータ構造に合わせて細分化，あるいは結合してカーネル・ルーチンに供給する．

2 章「UMA と NUMA の性能比較」で使用した NUMA カーネルは，「1. Page Allocator」に Node 制御を追加し，要求元 CPU の Node に合わせて物理ページの確保を行うようにしたものである☆．

本最適化では，これに加え，「2. Kernel Memory Allocator」に以下の変更を行った☆☆．

- カーネル・メモリ・アロケータで管理するデータ構造に対して，それぞれ Node 0 用と Node 1 用の 2 つのバッファを用意し，メモリ確保要求を行う CPU の Node に合わせたバッファ側からメモリを供給する．
- カーネル・メモリ・アロケータで管理しているバッファに対して，CPU ごとにキャッシュするバッファ管理機能を追加．なお，他 Node のメモリであればキャッシュせずに元のバッファに返却する仕様としている．

後者の CPU ごとキャッシュに関しては UMA カーネルにも実装を行い検証した．

3.2 スケジューラの最適化

Linux のスケジューリング・ポリシーは，タスク切換え時に，以前走行した CPU が空いていればそこにスケジュールし，そうでなければプライオリティ計算を行いディスパッチするという方針である．プロセッサ・グループの概念はなく，プロセス・バインドの機能も持っていない（Linux-2.4.0-test1）．

本最適化では，プロセス・バインドを実装し，長期的なメモリのローカリティを高めている．具体的には，プロセス生成時に round robin でプロセッサへのバインドを行い，他の CPU へのマイグレーションはいっさい行わないようスケジューラの変更を行った☆☆☆．

一般的にはこのようなプロセス・バインドを行うと負荷分散がうまくいかず，システム全体のパフォーマンスが低下する可能性があるが，今回のターゲットアプリケーションである Apache（WebBench），ServeBench ともに，動的な負荷分散が必要ないアプリケーションであるため，実験的にバインド効果の最も高い CPU の固定化を行い性能測定した．

なお，プロセス・バインドに関しても UMA カーネルに同様の実装を行い検証した．

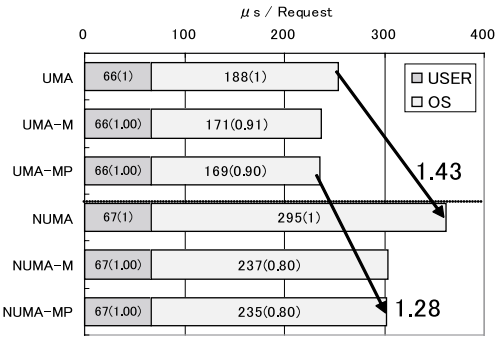


図 8 WebBench 性能
（カッコ内は base カーネルを 1 としたときの比率）
Fig. 8 Performance of WebBench.

4. 評　　価

評価は，2 章「UMA と NUMA の性能比較」と同一環境下で次のカーネルを使用して測定した．

- UMA（base）
- UMA-M（メモリ管理最適化）
- UMA-MP（メモリ管理最適化+プロセス・バインド）
- NUMA（base）
- NUMA-M（メモリ管理最適化）
- NUMA-MP（メモリ管理最適化+プロセス・バインド）

以下にまず性能結果を述べ，その後にカーネル・プロファイラによる OS 動作の分析結果，およびハードウェア・バストレーサによるメモリアクセスの分析結果を述べる．

4.1 性 能 結 果

WebBench

WebBench の性能を**図 8** に示す．

本結果により以下のことが分かる．

☆ ページ単位のメモリ管理 Node 対応は性能にほとんど影響せず，機能の有無による性能差は 3%以内であった．

☆☆ ソースコード変更量：9 ファイル（330 行）．

☆☆☆ ソースコード変更量：2 ファイル（18 行）．

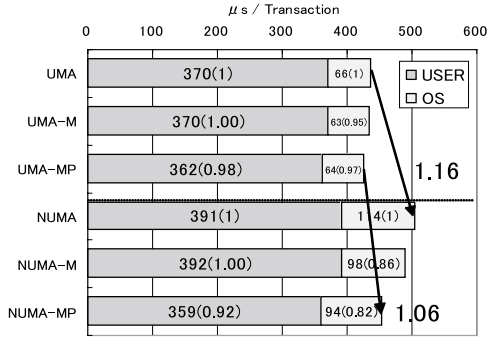


図 9 ServerBench 性能
（カッコ内は base カーネルを 1 としたときの比率）
Fig. 9 Performance of ServerBench.

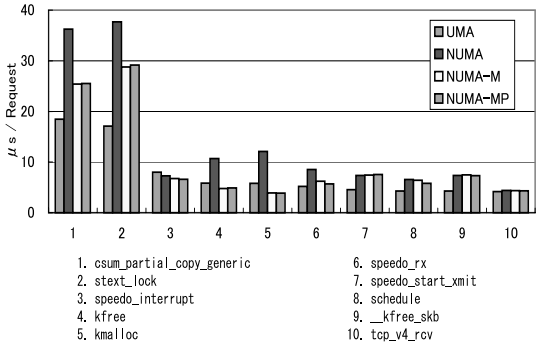


図 10 WebBench プロファイル上位 10 関数
Fig. 10 WebBench profile top 10 functions.

- WebBench での実行時間の減少のほとんどは，メモリ管理の最適化による OS 部への効果である．
- 本最適化により，OS 部の実行時間は UMA カーネルで 10%，NUMA カーネルで 20%減少した．
- UMA と NUMA を比較すると，NUMA の方が最適化効果が大きく，NUMA オーバヘッドは最適化前 43%に対して最適化後 28%に減少した（NUMA オーバヘッドは 2/3 になった）．

WebBench では，プロセス・バインドの効果がないことから，データ寿命がプロセス切換えよりも短いアプリケーションであるといえる．プロセス・バインドを行っても以前にスケジューリングされていた際のデータの再利用が行われないため，プロセス・バインドの効果が出ていない．

ServerBench

ServerBench の性能を**図 9** に示す．

本結果により以下のことが分かる．

- ServerBench では，メモリ管理の最適化およびプロセス・バインドの双方の効果が表れているが，特にプロセス・バインドの効果が大きい．
- 本最適化により，OS 部の実行時間は UMA カーネルで 3%，NUMA カーネルで 18%減少した．
- UMA と NUMA を比較すると，WebBench 同様に NUMA の方が最適化効果が大きく，NUMA オーバヘッドは最適化前 16%に対して最適化後 6%に減少した（NUMA オーバヘッドは 1/3 になった）．

ServerBench では，プロセス・バインドの効果が表れており，特に USER 部で効果が大きい．このことから，ServerBench はプロセス切換えに対して USER 部のデータ寿命が比較的長いアプリケーションであるといえる．また，NUMA カーネルでは OS 部においても若干ではあるがプロセス・バインドの効果が見られ，データ寿命の長いアプリケーションにおいては OS レベルでもマイグレーション抑制による効果が得られることが分かる．

これらの性能結果から，アプリケーションの性質により異なった最適化手法が有効であり，また得られる効果が異なることが検証できたが，双方の最適化の適用有無で UMA，NUMA ともに性能劣化はなく，これらの最適化を組み合わせて Linux カーネルに適用することが有用であると考える．

性能結果をさらに詳細に分析するため，性能測定と同一環境において分析ツールを用いデータ採取を行った．カーネル・プロファイラにより，NUMA カーネルにおいて OS のどの部分がボトルネックとなっているのか，また最適化によりどの部分が改善されたのかを分析した．さらに，ハードウェア・バストレーサによりメモリアクセスの局所化により実際にバス上に出るメモリトランザクションはどう変化したのかを分析した．以下にそれぞれの分析結果について述べる．

4.2 プロファイルによる分析

NUMA カーネルにおける最適化効果を OS の関数レベルで検証するために，カーネル内に一定時間ごとに実行命令アドレスなどを記録するプロファイルコードを追加し，タイムベースサンプリングを行った．これを基に，ベンチマーク実行時における各 OS 関数の処理時間を求め，ボトルネック箇所の特定および改善効果の分析を行った．

WebBench

UMA での上位 10 関数を基にした，各 NUMA カーネルでの処理時間の増減を**図 10** に示す．

WebBench は，1. csum_partial_copy_generic および 2. stext_lock の処理時間割合が大きいという特徴がある．csum_partial_copy_generic は，ネットワーク

送受信時（このケースでは特に送信時）に USER 領域と OS 領域の間でチェックサムをとりながらデータのコピーを行う関数である．NUMA 構成時に処理時間が増加しているのは，データのコピー時に転送元および転送先の一方，もしくは双方がリモートである場合にメモリアクセス・レイテンシが増加することに起因する．

stext_lock は，カーネル内でのスピンロック・コンテンション時におけるロック待ち処理を行うセクションであり，スピンロック処理を記述した関数内でロックが確保できなかった場合に実行される特別な処理部である．NUMA 構成時に処理時間が増加しているのは，ロック変数を保持した状態でリモートメモリ・アクセスが発生した場合に，ロック保持時間が増えることにより他のプロセッサがロックを確保できずロック待ち処理となることが主因である．

csum_partial_copy_generic および stext_lock ともにメモリ管理の最適化により処理時間が減少している．また，4. kfree，5. kmalloc のメモリ領域開放/確保関数もメモリ管理の最適化により処理時間が減少している．

効果の要因として，csum_partial_copy_generic ではデータコピーを行う際に OS が確保するワーク領域に，処理を実行している CPU と同一 Node のメモリが使用されることで他 Node メモリアクセスが減少し，また同一 CPU でキャッシュしているメモリが使用されることでキャッシュがミスが減少し処理時間が削減されたと推測される．stext_lock では，メモリ管理の最適化で他 Node アクセスを減らしキャッシュ効率を高めたことによりロック保持時間が減少したことに加え，カーネル内バッファ領域の細分化によりロック粒度が細分化され競合が減少したことでロック待ちになる回数自体が減少したことで処理時間が削減されたと推測される．kmalloc，kfree 関数は，確保/解放する領域および管理領域がローカルメモリとなったことにより，初期化/終了処理の際のメモリ操作時のアクセス時間が減少したことより処理時間が削減されたと推測される．

なお，プロセス・バインドに関してはプロファイルからも OS に対する効果は見られない．

ServerBench

UMA での上位 10 関数を基にした，各 NUMA カーネルでの処理時間の増減を**図 11** に示す．

ServerBench は，1.file_read_actor の実行時間割合が突出して大きいという特徴がある．file_read_actor は，ファイルキャッシュからの読込み処理関数であり，

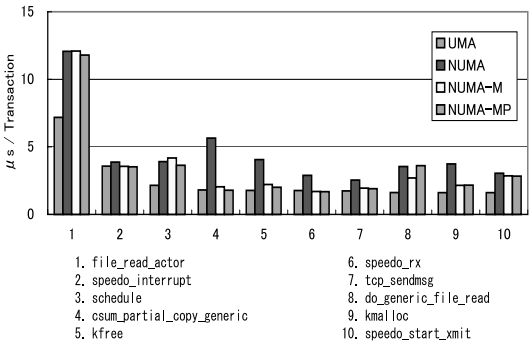


図 11 ServerBench プロファイル上位 10 関数
Fig. 11 ServerBench profile top 10 functions.

ServerBench はディスクアクセス（Read）の比重が高いプログラムであることが分かる．

file_read_actor に関しては本最適化では実行時間は減少しない．このため，ファイル読込みのオーバヘッドを削減するためには，キャッシュ領域に対するノード間のメモリ・レプリケーションなど NUMA 固有の最適化手法を用いる必要があると考えられる．

4. csum_partial_copy_generic，5. kfree，9. kmalloc に関しては，WebBench と同様にメモリ管理の最適化による効果が現れている．

なお，プロセス・バインドに関しては性能結果において OS の実行時間が若干（4%）減少している結果が得られたが，プロファイルからは特に大きく改善されている関数は見られない．

4.3 メモリアクセスによる分析

ハードウェア・バストレーサ GATES[11] を使用し，NUMA 構成における各ベンチマーク実行時のバス・トランザクションを分析した．バス・トランザクションには大別して，以下のものがある．

- CPU からのメモリアクセス（Read および Write）
- CPU からの I/O アクセス（in および out）
- I/O からのメモリアクセス（DMA）

これらのうち，性能（命令の実行速度）に大きく影響を与えるのは CPU からのメモリ Read アクセスである．メモリ Write アクセスおよび DMA はコマンドの突き放しが可能であること，I/O アクセスは本サーバ機環境ではほとんどが Memory Mapped I/O アクセス（メモリアクセスとしてカウントされる）であり絶対数が無視できるほど少ないためである．このため，本分析では，CPU からのメモリ Read トランザクションのみを抽出し分析した．なお，本分析ではバス上に出たすべてのメモリ Read トランザクションを対象としており，USER/OS の区別は行っていない．

表 3 メモリアクセスのノード・ローカリティ
Table 3 Node locality of memory accesses.

	WebBench	ServerBench
NUMA	53%	51%
NUMA-M	56%	51%
NUMA-MP	55%	75%

まず，メモリアクセスにおいて，同一 Node のローカルなメモリに対するアクセスの比率（以降「ノード・ローカリティ」と記述する）を**表 3** に示す．

メモリ管理の最適化に関して WebBench，ServerBench ともに，ノード・ローカリティはほとんど高まっていない．

プロセス・バインドに関しては，WebBench はノード・ローカリティは変わらないが，ServerBench では 24%高まっている．ただし，ServerBench におけるプロセス・バインドの効果は USER 部の実行時間の減少が主であることから，ノード・ローカリティに関しても主に USER 部が高まったと推測される．このことから，今回実施した最適化手法においては，OS 部のノード・ローカリティは高まらないといえる．

次に，メモリアクセスをその応答の種類により次の 4 つに分類した．**図 12** と**図 13** に分類結果を示す．

r: Remote
Remote Node の CPU もしくはメモリからデータを転送した場合（以下の Local の場合と異なりすべてのキャッシュ状態を含む）

l-n: Local no-hit
Local Node 内の他 CPU はデータを持っておらず，メモリから転送した場合

l-s: Local Shared
Local Node 内の他 CPU が Clean 状態でデータを持っていた場合

l-m: Local Modified
Local Node 内の他 CPU が Dirty 状態でデータを持っていた場合

WebBench（図 12）

WebBench ではメモリ管理の最適化により，バス上に出たメモリアクセス数が減少しており，特に Remote と Local Modified の数がそれぞれ約 15%減少している．

WebBench は ServerBench と比較して Local Modified の割合が高いことから，Dirty 状態のキャッシュブロック転送が多く，プロセス間，プロセッサ間での排他制御あるいは false sharing が多いことが推測される．

メモリ管理の最適化により Local Modified が減少

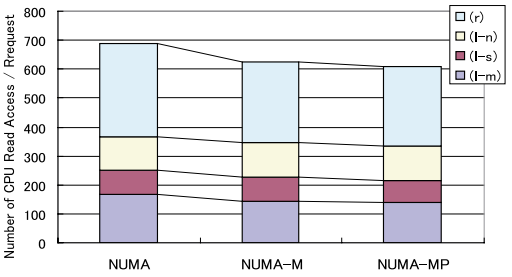


図 12 WebBench メモリアクセス分類
Fig. 12 Memory access classification of WebBench.

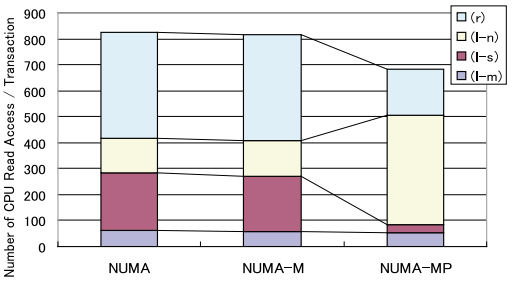


図 13 ServerBench メモリアクセス分類
Fig. 13 Memory access classification of ServerBench.

していることから，OS 内のワーク領域の開放/再割当て処理にともなう排他制御およびメモリオーナの移動が削減されたと考えられる．

ServerBench（図 13）

ServerBench ではメモリ管理の最適化によるメモリアクセス分類の変化は見られない．Local Modified の割合が少なく，Remote の数も減少していないことから，プロセス間，プロセッサ間で処理が独立しており，カーネルのワーク領域の利用度が低いアプリケーションであると推測される．

一方プロセス・バインドに関しては，メモリアクセス数が減少していることから，データの寿命が長いアプリケーションであると推測される．Remote の数が約半減しており，さらに Local Share が減り Local no-hit が増加していることから，プロセス・バインドによりプロセスごとの空間の独立性が高まり，キャッシュミスが削減され，性能が向上したと考えられる．

5. ま と め

本研究では，Linux を用いて NUMA システムの Commercial Workload 向け OS 最適化の実験を行った．

UMA システムと NUMA システムの性能差は，実験に用いた Commercial Workload においては OS 部

に大きく依存しており，OS レベルの最適化を行うことで性能を向上させることが可能であることを検証できた．

OS 最適化では，NUMA 固有の最適化手法ではなく，一般的なメモリアクセスの局所化を用い，UMA システムおよび NUMA システムの双方に対して実装した．その結果，NUMA システムは UMA システムと比べて最適化に対する効果が高いことが検証でき，OS の実行時間を 18%〜20%削減でき，NUMA オーバヘッドを 1/3〜2/3 に削減することができた．

最適化手法においては，WebBench のように OS の実行時間の比率が高く，OS 内のワーク領域の使用頻度が高いアプリケーションには，カーネル・メモリ・アロケータの最適化が有効であること，また ServerBench のように USER の実行時間比率が高く，プロセスごとに独立したメモリ配置および処理を行っているプログラムにはプロセス・バインドが有効であることが検証できた．

さらに，カーネル・メモリ・アロケータの最適化ではノード・ローカリティは向上せず，メモリアクセス数の削減が性能向上に寄与していることが検証できた．

なお，プロファイルから見た NUMA オーバヘッドの分析では，ファイルキャッシュ関連を除いて主なオーバヘッドは半減されており，本評価で用いたベンチマークプログラムのように，クライアント間でのデータ共有がなく，リクエストパターンに相関がないアプリケーションに関しては，レプリケーションなどの NUMA 固有の最適化を追加しても本最適化ほどの効果は期待できないと推測される．ただし，本最適化では ServerBench においてファイル読込みのオーバヘッドを削減できなかったことから，ファイルアクセスが頻繁でかつその実行比率が高いようなアプリケーションに関しては，ノード間のメモリ・レプリケーションなど NUMA 固有の最適化手法を検討する必要があると考えられる．

今後は，NUMA 構成でのスケーラビリティ評価，OLTP やファイルサーバ，メールサーバなどより広範な Commercial Workload での動作分析，大規模環境での検証などを行っていく予定である．

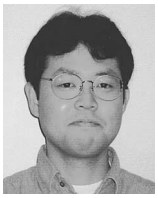
参 考 文 献

1) Laudon, J. and Lenoski, D.: The SGI Origin: A ccNUMA Highly Scalable Server, *Proc. 24th Ann.Int'l Symp.on Computer Architecture* (*ISCA24*) (1997).
2) Lovett, T. and Clapp, R.: STiNG: A CC-NUMA Computer System for the Commercial Marketplace, *Proc. 23rd Ann. Int'l Symp. on Computer Architecture* (*ISCA23*), pp.308–317 (1996).
3) Lenoski, D., Laudon, J., Gharachorloo, K., Weber, W.-D., Gupta, A., Hennessy, J., Horowitz, M. and Lam, M.S.: The Stanford Dash Multiprocessor, *IEEE Computer*, Vol.25, No.3, pp.63–79 (1992).
4) Kuskin, J., Ofelt, D., Heinrich, M., Heinlein, J., Simoni, R., Gharachorloo, K., Chapin, J., Nakahira, D., Baxter, J., Horowitz, M., Gupta, A., Rosenblum, M. and Hennesy, J.: The Stanford FLASH Multiprocessor, *Proc. 21st Ann. Int'l Symp. on Computer Architecture* (*ISCA21*), pp.302–313 (1994).
5) Hagersten, E. and Koster, M.: WildFire: A Scalable Path for SMPs, *Proc. 5th Int'l Symp. on High Performance Computer Architecture* (*HPCA5*) (1999).
6) GRANPOWER5000 (Model880). http://www.fujitsu.co.jp/hypertext/granpower/gp5/catalog/close/gp9901/m880/
7) Weber, W.-D., Gold, S., Helland, P., Shimizu, T., Wicki, T. and Wilcke, W.: The Mercury Interconnect Architecture: A Cost-effective Infrastructure for High-performance Servers, *Proc. 24th Ann. Int'l Symp. on Computer Architecture* (*ISCA24*), pp.98–107 (1997).
8) WebBench. http://www.zdnet.com/etestinglabs/stories/benchmarks/0,8829,2326243,00.html
9) ServerBench. http://www.zdnet.com/etestinglabs/stories/benchmarks/0,8829,2349035,00.html
10) Mindcraft benchmark Report. http://www.mindcraft.com/whitepapers/nts4rhlinux.html
11) 佐藤 充，成瀬 彰，久門耕一：GATES（PC サーバ用汎用メモリアクセストレースシステム）の開発，情報処理学会第 59 回全国大会講演論文集 (1999).

(平成 13 年 9 月 4 日受付)
(平成 14 年 2 月 13 日採録)

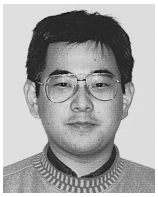
平井　聡（正会員）

（株）富士通研究所勤務．1997 年より同研究所にて IA プロセッサを使用した大規模 PC サーバ向け性能向上技術の研究に従事．現在，Linux カーネルを素材とした高性能，高信頼システムの研究を行っている．



山本　昌生

1993 年大阪府立大学工学部電子工学科卒業．同年富士通（株）入社．ビジネスサーバの開発に従事．1997 年（株）富士通研究所に配転．以来 IA サーバにおける性能評価・向上技術，IA-64 評価ツール等の研究開発に従事．

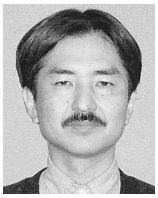


佐藤　充（正会員）

1969 年生．1992 年東京大学工学部電気工学科卒業．1997 年同大学大学院工学系研究科情報工学専攻博士課程修了．博士（工学）．同年富士通（株）入社．現在,（株）富士通研究所勤務．並列システムアーキテクチャの研究に従事．IEEE，ACM 各会員．

成瀬　彰（正会員）

1996 年名古屋大学大学院工学研究科修了（情報工学専攻）．同年富士通（株）入社．IA サーバに関わる研究・開発に従事．並列処理，計算機アーキテクチャに興味を持つ．



久門　耕一（正会員）

1979 年東京大学工学部電気工学科卒業．1981 年同大学大学院電子工学専門課程修士課程修了．1984 年同大学院博士課程中退．同年（株）富士通研究所入社．現在，同社コンピュータシステム研究所に所属．CPU，メモリ，並列計算機アーキテクチャに関する研究に従事．GCC，Linux カーネル等の改良にも興味を持つ．日本ソフトウェア科学会会員．