キャッシュミス削減によるLinuxプロセススケジューラの高速化

山村周史[†] 平井聡[†] 佐藤充[†]
山本昌生[†] 成瀬彰[†] 久門耕一[†]

本論文では，高負荷時における Linux スケジューラの高速化を目的とした，カーネル内部のタスク構造体に対するキャッシュカラーリングの実装方式を提案する．従来の Linux カーネルでは，タスク構造体が物理ページ境界に配置されていなければならないという制約があった．本方式は，タスク構造体の先頭アドレスを，その上位ビットを用いてハッシュすることによってこの制約を解消し，これまで困難とされていたタスク構造体のカラーリングを Linux カーネルで実現した．これにより，8-way Pentium III サーバ上で Web サーバ性能が最大 23.3%，Chat サーバにおけるメッセージスループット性能が最大 89.6%向上した．本論文では，性能向上を評価するだけではなく，実機上でのメモリバストランザクションやキャッシュミス率を実測することで，キャッシュカラーリングのメモリ特性を分析し，カラーリングを効率的に利用するための条件についても明らかにする．

Speeding Up Linux Process Scheduler by Reducing Cache Misses

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

In this paper, we propose the experimental implementation of cache coloring for a task structure to speed up Linux process scheduler. In the current Linux kernel implementation, the task structure is always aligned on a page boundary in a physical address space. By hashing the base address of the task structure with its upper bits, our coloring scheme can cancel this restriction and realize the cache coloring for a task structure which has been difficult to implement until now on the Linux kernel. The experimental results on an 8-way Pentium III server machine showed that the Web server performance and Chat server performance achieves a maximum of 23.3% and 89.6% improvement compared to the standard kernel, respectively. Moreover, we also demonstrate the memory characteristics of our coloring scheme and clarify about its effective conditions by measuring memory-bus transactions and a cache miss ratio on a real machine.

1. はじめに

近年，Linux は Web サーバや Mail サーバ，あるいは PC クラスタシステムなどの幅広い分野において盛んに利用されている．ここ数年は，さらに大規模なサーバをプラットフォームとするエンタープライズ分野への適応が Linux の重要なテーマとなっている．このような分野では，Linux システムの高負荷時の安定性や搭載 CPU 数に見合った性能スケーラビリティの向上が要求される．

これに対して，大規模 SMP や NUMA マシンなどを対象として，高速 I/O，プロセススケジューラ，障害解析，大容量メモリ管理などについて様々な研究や改良が行われてきた[1)~3)]．しかしながら，エンタープライズ分野での利用時に発生するような高負荷時（多プロセス動作時）における Linux カーネルの挙動について，メモリシステムの観点からの調査・分析は十分に行われているとはいえなかった．

そこで，我々は，最も普及したプラットフォームである IA-32（32-bit Intel Architecture）を対象として，独自に開発したハードウェアバストレーサ GATES[4)] を用いて実システム上での Linux の評価に取り組んだ．その結果，Linux カーネルのプロセススケジューラ内部において，キャッシュメモリの利用に問題があり，これが高負荷時における性能のボトルネックとなることを確認し，さらにカーネルに改良を加えることで性能向上を達成した．この結果は，文献 5) においてすでに報告している．

しかしながら，キャッシュメモリに対するアクセス

†（株）富士通研究所
Fujitsu Laboratories, Ltd.

の最適化のみで Linux システムがどの程度高負荷な状況に対応できるかについて，スケジューラ内部のデータ構造やアルゴリズムを変更して性能向上を図るアプローチとの比較を十分に検討したとはいえない．本論文では，文献 5) の内容に加え，スケジューラ内部のデータ構造やアルゴリズムを大幅に変更した他の高速スケジューラとの性能比較を通して，メモリシステムの最適化のみでどれだけ性能が向上するか実験を行った結果について報告する．

以降，2 章で，メモリ評価ツールについて簡単に説明するとともに，本論文の対象とする Linux カーネル（バージョン 2.4）のスケジューラの構成とその問題点についてまとめる．続いて 3 章において，プロセスを管理するためのタスク構造体に対するキャッシュカラーリングによる解決手法について説明する．4 章において，実装したカラーリング手法の実験結果について報告し，バストランザクション統計情報を用いてカラーリングの特性について詳しく考察する．続いて，5 章において CPU 数が増加する場合の性能スケーラビリティについて，カラーリングが与える効果について述べる．6 章において，現在の関連する研究事項についてまとめ，最後に 7 章で本論文の総括を行う．

2. Linux2.4 スケジューラの問題点

2.1 メモリバストレース調査環境

近年のコンピュータシステムでは，プロセッサの動作周波数の向上により，メモリアクセスにかかるコストが相対的に増加しており，これがシステム全体の性能を左右するに至っている．しかしながら，本論文の対象とするサーバ・エンタープライズ分野で利用されるようなアプリケーションの場合，そのメモリアクセス特性を把握して性能ボトルネックを特定することは一般に困難であると考えられていた．特に，SMP マシン上で動作する大規模な商用ワークロードでは，コードサイズの増大やプログラムの並列・協調実行がともなうため，その挙動を把握することはさらに困難である．

そこで我々は，システム性能のボトルネックとなり得るメモリアクセスを容易に特定することを目的として GATES（General-purpose memory Access Trac E System）を開発した．GATES は，IA-32 サーバ用の汎用メモリアクセストレーサである．GATES を搭載するシステムは，**図 1** のような構成をとる．この図は，Pentium Pro を 4 台搭載する GRANPOWER 5000 MODEL 570 上でのメモリバストランザクションを観測するシステムである．図のように，通常プロセッサが搭載される CPU ソケットに対して直接 GATES を搭載することで，共有メモリバス上に流れるトランザクションを観測する．これにより，各プロセッサからのメモリアクセスや I/O アクセス，2 次キャッシュの応答（スヌープ結果）などを，被測定環境になんら影響を与えずに測定することができる．

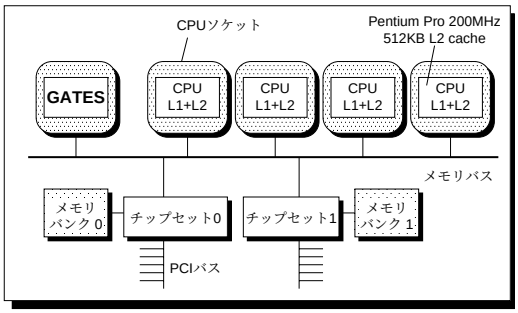


図 1 GATES を装備した実験システムの構成
Fig. 1 Experimental system configuration with GATES.

これまで我々は，商用ワークロードが動作する様々な実システムに対して本ツールを適用し，アプリケーションのメモリアクセス特性の分析や，問題発見・性能改善に取り組んできた[6)~8)]．本論文では，本ツールを高負荷時での Linux カーネルの挙動分析に対して適用した．

2.2 プロセススケジューラの問題点

バージョン 2.4 の Linux カーネル（以下，単にカーネルと略記）は，高負荷時（多プロセス動作時）におけるスケジューラの挙動について，これまで以下のような問題点が指摘されてきた．

(1) 実行可能状態にあるすべてのタスクを「runqueue」と呼ぶ単一のキューで管理する．スケジューラは，次に実行すべきプロセスを選択する際，runqueue を全探索する．そのため，実行可能状態にあるプロセスが増加した場合，走査に要する時間が長くなる．
(2) SMP システムでは，runqueue は複数の CPU 間で排他的にアクセスされる．したがって，(1) の長い走査時間はロック保持時間の増加とロック競合の増大を招く．これにより，CPU 数が増加した場合の性能向上を妨げる．

我々は，スケジューラの挙動を確認するため，図 1 に示す 4 CPU 構成のサーバ上で，高負荷時における Linux システムのメモリアクセスを GATES により採取した．

結果を**図 2** に示す．この図は，サーバ上で Web サーバである Apache を動作させ，複数のクライアントマシンから並列にリクエストを送信して高い負荷をかけ

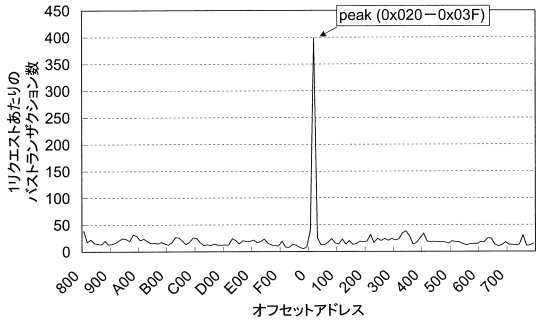


図 2 Apache 動作中のメモリアクセス統計（4-way Pentium Pro 200 MHz）
Fig. 2 Memory access statistics of running Apache on the 4-way Pentium Pro server.

た状態にあるときのメモリアクセス統計である．横軸はページ内のオフセットアドレス，縦軸は 1 つの Web リクエストあたりで正規化したメモリバストランザクション数である．IA-32 を対象とした Linux カーネルは，物理ページサイズを 4 KB としているため，そのオフセットアドレスは 0x0〜0xFFF をとる．なお，横軸については，トランザクションのピークを分かりやすくするために，中央をオフセット 0 として折り返して表示している点に注意されたい．図から分かるように，オフセットアドレス 0x20〜0x3F におけるメモリバストランザクション数が突出している．これは，当該アドレスへのメモリアクセスに対してキャッシュミスが多発した結果であると考えられる．当該オフセットアドレスとカーネル内部のデータとのマッピングを行ったところ，プロセス情報を管理するタスク構造体へのアクセス時に上記のような大量のバストランザクションが発生していることが分かった．

カーネルは，各実行中のプロセス固有の情報を管理するためにタスク構造体と呼ぶデータ構造を用いている．**図 3** にタスク構造体と runqueue の構造を示す．タスク構造体は，各プロセスごとに用意された 8 KB のカーネルスタック領域の先頭に配置されている．そして，runqueue は，これらタスク構造体をポインタで連結した構造を保持している．Linux カーネルは，すべてのカーネルスタックを物理メモリ上の 8 KB 境界に配置するという方式を採用している．この方式には，スタックポインタの下位 13 ビット（$2^{13} = 8\,\mathrm{KB}$）をマスクするだけでタスク構造体の先頭アドレスを高速に決定できるという特徴がある[9]．

しかしながら，スケジューラが次に実行すべきプロセスを選択する際に上記実装が問題となる．このときの優先度計算は，runqueue を走査し，各タスク構造体内部に格納された変数値，たとえばプロセス

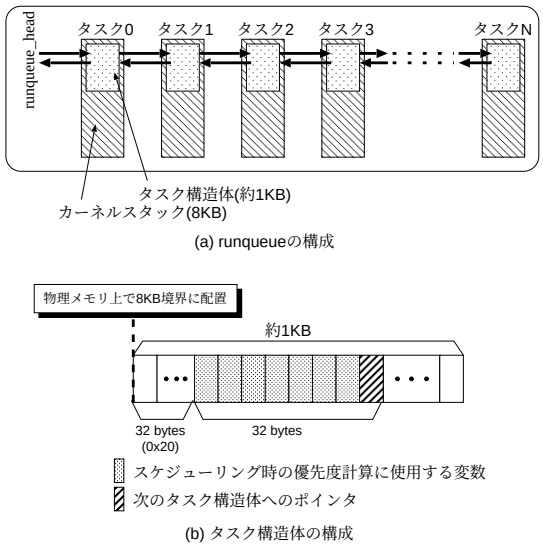


図 3 タスク構造体と runqueue の構造
Fig. 3 Task structure and runqueue.

の CPU 利用時間などを参照することによって行われる．図 3 (b) のように，スケジューラが優先度を計算するための変数が，タスク構造体内部の先頭から 0x20〜0x3F（32 バイト）のオフセット位置に格納されている．このため当該 32 バイトの物理アドレスは，$8\mathrm{KB}(2^{13}) * n + (\mathtt{0x20...0x3F})$ となり，下位 13 ビットがすべてのタスク構造体で同一の値となる．IA-32 アーキテクチャでは，キャッシュラインとメモリアドレスとのマッピングを下位ビットを用いて決定するため，スケジューラがポインタをたどってタスク構造体に連続してアクセスすると，同一キャッシュライン上で競合する可能性が非常に高くなる．結果，プロセスが多いと runqueue 走査中にキャッシュメモリが有効に機能せず，図 2 のように特定のオフセットアドレスに対して多数のバストランザクションが発生する．

特に，SMP カーネルにおいては，runqueue はスピンロックにより保護されている．そのため，キャッシュミスの発生により runqueue 走査時間が長くなると，それと比例してロック保持時間も長くなる．そのため，CPU 数を増加してもロック競合の割合が高くなり，性能スケーラビリティが得られない可能性がある．

3. タスク構造体に対するキャッシュカラーリング手法

3.1 カラーリングを実装するうえでの問題点

前節で述べたような，メインメモリ上での配置によって生じるキャッシュライン競合を避ける手法として，「カラーリング」がよく知られている[10),11)]．カラー

```
static inline struct task_struct * get_current(void)
{
  struct task_struct *current;
  __asm__("andl %%esp,%0; ":"=r" (current) : "0" (~8191UL));
  return current;
}
```

(a) オリジナルのget_current関数

```
static inline struct task_struct * get_current(void)
{
  struct task_struct *current;
  __asm__("andl %%esp,%0; ":"=r" (current) : "0" (~8191UL));
  (unsigned long)current |= ((unsigned long)current >> CACHE_INDEX_BIT) & 0x00000060;
  return current;
}
```

(b) 修正したget_current関数

図 4 修正した get_current 関数
Fig. 4 Modified get_current fuction.

リングとは，競合を発生するデータについて，キャッシュラインサイズを単位としてメモリ上で分散配置することで競合を回避する手法である．したがって，カラーリングをタスク構造体に対して適用することで問題を解決できると考えられる．

しかしながら，Linux カーネルにおいてタスク構造体をカラーリングすることは困難と考えられていた．これは，カーネルが現在実行中のプロセスの情報（たとえば pid）を得るときに，特殊なテクニックを用いているからである．カーネルはカレントプロセスの情報を以下のような手順で取得する．

(1) `get_current` 関数を呼び出す．
図 4 (a) の関数は，`%esp`（現在のスタックポインタの値）と `0xffffe000` との論理積をとり，その結果を返す．この値は，`%esp` のアドレスからアドレス 0 の方向に向かって，8 KB 境界のアドレスである．

(2) 得られた先頭アドレスにオフセットを加える．取得したいデータのオフセットアドレスを戻り値に足し込み，実効アドレスを計算する．

(3) そのアドレスにアクセスする．

もし，カラーリングを行い，タスク構造体の位置を安易にずらすと (1) で得られたベースアドレスが間違ったものとなってしまいカーネルが正常に動作できなくなる．このように，単純にカラーリングをタスク構造体に対して適用することは困難であり，現在まで行われていなかった．

3.2 カラーリングの実装方式

タスク構造体は，メモリ上の配置について前節のような制限があるが，我々は `get_current` 関数に対するごくわずかな修整でカラーリングを実現できることに気づいた．

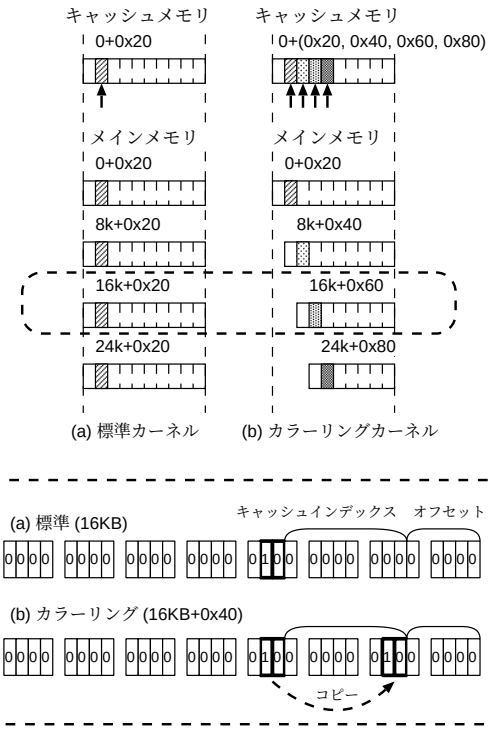


図 5 カラーリングの例（8 KB キャッシュ，4 カラーリング）
Fig. 5 Example of coloring (8 KB cache memory, the number of colorings is four).

図 4 は，修整した `get_current` 関数である．図から分かるように，太字で示した 1 行のみの修整である．この関数は，キャッシュラインサイズの倍数だけシフトしたベースアドレスを呼び出し元に返す．`get_current` 関数は，システム内部で頻繁に呼び出される．したがって，数回のビット操作命令を追加するだけにとどめ，オーバヘッドを抑えている．

図 5 の例を用いて，本カラーリング手法の実装につ

いて説明を行う．簡単のため，キャッシュメモリのサイズが 8 KB，マッピング方式はダイレクトマッピングとする．また，カーネルスタック（タスク構造体）がメインメモリ上で連続して配置されているものとする（実際に，このように連続しているとは限らないが，必ずメインメモリ上で 8 KB 境界に整列されている）．

図 5 (a) のように，標準カーネルでは，タスク構造体内のデータはキャッシュメモリに転送されたときに競合を発生する．一方，図 5 (b) では，修正した `get_current` 関数がカラーリングされたアドレス，具体的には，キャッシュメモリのインデックスビットより上位のビットを使ってハッシュしたアドレスを返す．たとえば，点線で囲まれた 16 KB の位置に配置されたカーネルスタックは，カラーリング後は，上位 2 ビット (10) を下位ビットにコピーすることで (16 KB+0x40) の値を算出して返す．

なお，実際にカーネルを正常に動作させるためには，`get_current` 関数以外のカーネルソースファイルに対しても若干の修正が必要となる．しかし，変更は非常に少なく，すべて合わせても 200 行程度である．

3.3 キャッシュカラーリングによる性能への悪影響

キャッシュカラーリングは，runqueue 走査時のキャッシュミスを減少させるという効果がある一方で，性能に対する悪影響も懸念される．

図 5 を用いて説明すると，(a) 標準カーネルの場合には，1 つのキャッシュブロックに対してデータが転送される．一方，カラーリングを施した場合は図中 (b) のようにデータが転送されるキャッシュラインが 4 つに分散される．これにより，本来であればキャッシュに格納されていたはずのデータが逆に追い出されてしまうという悪影響が懸念される．プロセス数が多くなった場合には，逆にメモリバストランザクションが増加して性能の低下を招く可能性がある．

以降の性能評価では，この点についても焦点をあて詳しい評価を行う．

4. 性能およびメモリシステムの評価

4.1 実験環境

キャッシュカラーリングの効果を確かめるために，我々は Web サーバとして Apache 1.3.19 を用いて，Web リクエスト処理性能を測定した．カラーリングの数は 32 とし，カーネルはバージョン 2.4.4 を用いた．

サーバマシンの構成を**表 1** に示す．2 次キャッシュサイズは 256 KB，512 KB，1,024 KB の 3 種類の構成で評価した．これは，キャッシュサイズによるカラーリングの効果の違いを確認するためである．サーバ側

表 1 サーバ構成
Table 1 The specification of the server machine.

CPU	Pentium Pro 200 MHz × 4
メモリ	1GB
2 次キャッシュメモリ	256 KB, 512 KB, 1,024 KB 4-way set associative
ネットワークカード	Intel EtherExpress Pro/100 × 4

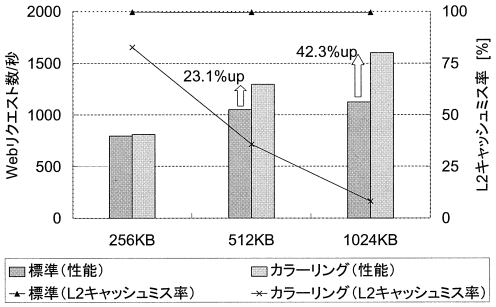


図 6 カラーリングの効果（Web リクエスト処理性能および 2 次キャッシュミス率）
Fig. 6 Effects of cache coloring (Web performance and L2 cache miss ratio).

では Web サーバプロセス（httpd）を 1,024 個起動して，クライアントが要求したリクエストを処理する．この実験では，Pentium Pro という数世代前のプロセッサを使用している．これは，GATES の対応する CPU が Pentium Pro であることと，キャッシュサイズを変更してカラーリングの影響を調査するためである．しかし，最新の IA サーバ上において，メモリアーキテクチャについての大きな差はなく，ここで得られた評価結果は同様の傾向を示すと考えられる．また，クライアントマシンは，28 台の PC/AT で構成する．クライアントから発生させる Web リクエストの生成には，WebBench 3.0 [13] を使用した．なお，ベンチマーク実行時には，ファイルはファイルキャッシュに載る．したがって，ディスクアクセスはほとんど発生しない．

4.2 実験結果

性能測定結果を**図 6** に示す．図の横軸はキャッシュサイズ，縦軸は 1 秒あたりの Web リクエスト処理数である．また，スケジューラ内部で runqueue を探索する部分（`list_for_each` ループ）で発生した L2 キャッシュミス回数と L2 リード回数を測定し，L2 ミス率を測定した．L2 ミス率は以下の式で計算する．これらの測定には，プロセッサに装備されている性能モニタリングカウンタを使用した．この結果もあわせて図 6 に示す．

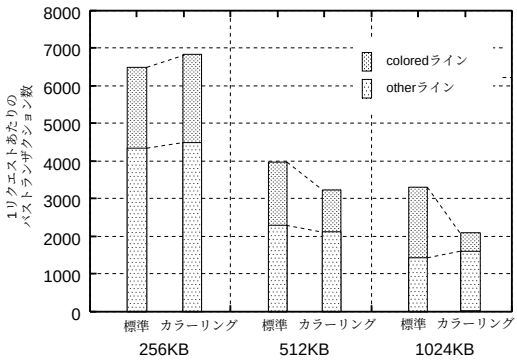


図 7 バストランザクション数（1,024 httpd 稼働時）
Fig. 7 Number of memory-bus transactions (1,024 httpd processes are running).

$$L2\ cache\ miss\ ratio = \frac{L2\ cache\ miss\ counts}{L2\ read\ counts} \tag{1}$$

加えて，1Web リクエストあたりで正規化したバストランザクション数の変化を**図 7** に示す．この図では，GATES で計測中に発生したバストランザクションを以下の 2 種類に分類し表示している．

「colored ライン」 カラーリングした場合，タスク構造体が複数のキャッシュラインに分散して格納される．これらのキャッシュラインでキャッシュミスした場合に発生するバストランザクション．

「other ライン」 上記以外のキャッシュラインでキャッシュミスした場合に発生するバストランザクション．

まず，256 KB の場合については，図 6 から分かるようにわずかな性能向上しかみられない．バストランザクションのデータから分かるように，カラーリングを行うことで図 7 中の colored ラインに関するバストランザクションがわずかに増加している．すなわち，前章で述べたカラーリングによって競合が発生するキャッシュラインが増加するという悪影響が若干現れている．この場合，カラーリング数が 32 なので，理想的には (キャッシュサイズ 256 KB/カーネルスタックサイズ 8 KB) × 32 = 1,024 のプロセスがキャッシュに載るはずである．しかしながら，図 6 から分かるように，L2 ミス率は 17%の減少にとどまっている．実際には，タスク構造体は必ずしもメインメモリ上で連続して確保されているわけではないので，256 KB とキャッシュメモリが小さい場合には，カラーリングをしてもキャッシュに載りきらず，その効果が得られていない．

一方，512 KB および 1,024 KB の場合は，23.1%，42.3%の大きな性能向上を達成している．これと同時に，キャッシュミス率は劇的な減少を示している．カラーリングを行わない場合は，ほぼ 100%と L2 キャッシュとしては異常な値を示していたものが，カラーリングを行った場合は，512 KB L2 キャッシュの場合は 35.8%，1,024 KB L2 キャッシュの場合は 8.2%にまで減少している．図 7 から分かるように，colored ラインに関するバストランザクションがそれぞれ 33.3%，73.0%と大幅に減少している．

キャッシュカラーリングは，本節の実験結果から分かるように，大容量のキャッシュであるほどタスク構造体が分散して転送されるキャッシュラインの数が増加するので，より効果的である．このとき，スケジューラ内のキャッシュミスが激減しシステム全体の性能が向上する．

4.3 プロセス数とカラーリング数との関連性

本論文では，カラーリング数を変化させた場合のカラーリングの効果も調査した．本節では，その結果について述べる．

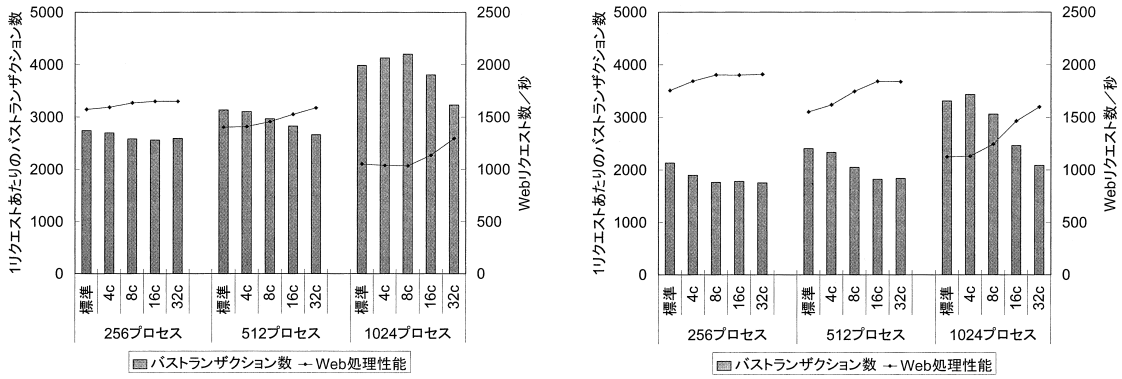
標準カーネルにおいて，スケジューリング中に競合が発生したキャッシュラインは，カラーリングされたカーネルでは異なるキャッシュラインに分散される．システム上のタスク構造体を十分にキャッシュメモリに格納するための適切なカラーリング数は，以下の式が目安になる．

$$c \geq \frac{p}{(s/8\,[\mathrm{KB}])} \tag{2}$$

ここで，c はカラーリング数，p はプロセス数，s はキャッシュサイズ [KB] である．式 (2) 中の 8 KB はカーネルスタックのサイズである．サイズ s のキャッシュであれば，$s/8$ 個のカーネルスタックがキャッシュできるはずである．たとえば，キャッシュサイズ 512 KB で 1,024 プロセス動作する場合は，$1,024/(512\,\mathrm{KB}/8\,\mathrm{KB}) = 16$ カラーリングすれば，その効果が確認できると考えられる．

キャッシュサイズが 512 KB および 1,024 KB の場合について，カラーリング数を（4，8，16，32），プロセス数を（256，512，1,024）に変化させた場合のカラーリングの効果を測定した．結果を**図 8** に示す．図中には，Web リクエスト処理性能およびバストランザクション数の変化を示している．

キャッシュサイズが 512 KB の場合については，式 (2) によれば，プロセス数が 256，512，1,024 それぞれについて，4，8，16 カラーリング必要である．図 8 を見ると，実際にそれらのカラーリング数から性能が向上していることが分かる．バストランザクションを



(a) 512 KB L2 キャッシュ (b) 1,024 KB L2 キャッシュ

図 8 カラーリング数を変化させたときのバストランザクション数と Web リクエスト処理性能の変化
Fig. 8 The number of bus transactions and performance changing the number of colorings.

見ると，それに対応するように標準カーネルの場合に比べて減少していることが分かる．しかし，1,024 プロセスが動作する場合，4，8 カラーリングでは，カラーリング数が不足している．この場合，カラーリングを行わない場合にキャッシュミスが集中して発生するキャッシュラインが増加したにすぎず，これによって逆に総トランザクション量が増加している．実際には，runqueue 走査時のキャッシュミス減少による性能向上と，バストランザクションの増加による性能低下が相殺された形で，システム全体の性能はほとんど変化がない．

キャッシュサイズが 1,024 KB の場合についても，512 KB の場合と同様のことがいえる．プロセス数が 256，512，1,024 それぞれについて，2，4，8 カラーリング必要である．実際に，カラーリング数がこれらの値より大きい場合にはバストランザクションが減少し，性能が向上していることが分かる．

このように，カラーリングの効果を得るためには，式 (2) に従ってカラーリング数を計算し，この値より大きなカラーリング数を選択する必要がある．

5. カラーリングによる性能スケーラビリティの向上

この章では，CPU 数が増加した場合の性能スケーラビリティに対するカラーリングの効果について述べる．ここでは，前章までで取り上げた標準カーネルおよびカラーリングカーネルに加えて，O(1) スケジューラ[14),15)] を実装したカーネル（以下，単に O(1) カーネルと略記）をあわせて評価する．

O(1) スケジューラは，各プロセスをプロセッサごとに用意された優先度付き待ち行列で管理するように標準カーネルに対して大幅に構造を変更する．O(1) カーネルでは，プロセスディスパッチ時において線形リストの探索が行われないので，本論文で問題としているスケジューラ内部でのキャッシュミスが原則として発生しないという特徴がある．本カーネルとカラーリングカーネルとを比較することで，リスト探索の有無による性能差を確認することができる．O(1) スケジューラは，既存の Linux カーネルのプロセススケジューラの実装方式の中でも高速なプロセススケジューラであると考えられている．しかし，現在はまだテスト実装段階であり，安定性を保証するには至っておらず，またサーバ・エンタープライズ用途として広く使われているバージョンのカーネルには簡単に適用できないという問題がある．

表 2 サーバ構成
Table 2 The specification of the server machine.

CPU	Pentium III Xeon 550 MHz × 8
メモリ	1 GB
2 次キャッシュメモリ	1,024 KB (4-way set associative)
ネットワークカード	Netgear GA622T (1 GbE) × 4

5.1 実験環境およびベンチマークプログラム

前章での実験で用いたサーバよりもさらに搭載 CPU 数の多い 8-way Pentium III サーバ上で評価を行った．サーバ構成を**表 2** に示す．カーネルのバージョンは前章と同様 2.4.4 とし，カラーリング数は 32 とした．ただし，O(1) カーネルについては，バージョン 2.4.18 を使用した．これは O(1) スケジューラがそれ以前のカーネルには未対応のためである．これらカーネルのバージョンは若干異なるが，スケジューラの性能を測定する今回の性能評価にはほとんど影響を与えない．

ベンチマークプログラムとして WebBench 3.0 および Chat ベンチマーク[3),16),17)] を使用した．

WebBench を実行する場合，サーバ側で 256 個の

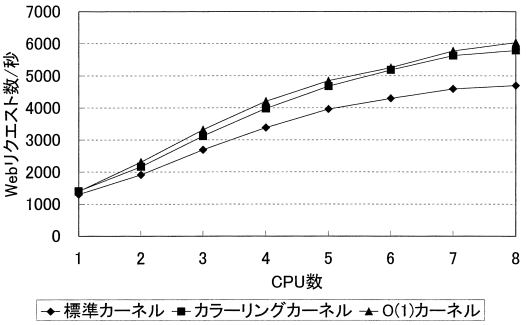


図 9 WebBench の性能（256 httpd 稼働時）
Fig. 9 WebBench performance (256 httpd processes are running).

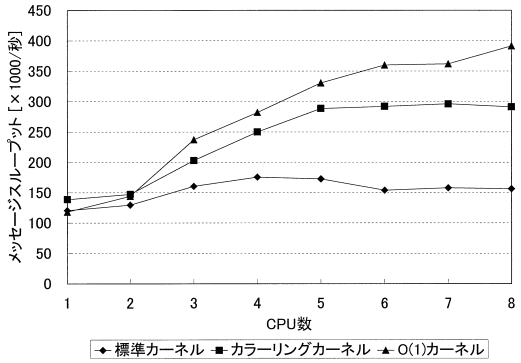


図 10 Chat スループット性能（30 ルーム，300 メッセージ）
Fig. 10 Chat performance (30 rooms, 300 messages).

httpd を起動し，前章と同様に 28 台の PC/AT クライアントマシンから Web リクエストを同時に発行する．

Chat ベンチマークは，クライアントマシンを必要としない．このベンチマークは，TCP ソケット通信を行う複数のユーザがいるチャットルームをシミュレーションする．各チャットルームには 20 人のユーザがおり，各ユーザは 100 バイトを単位としてメッセージの送受信を行う．サーバ側とクライアント側にそれぞれメッセージ送受信のスレッドを起動するので，1 ユーザあたり 4 スレッド生成する．したがって，1 チャットルームあたり 80 個のスレッドが生成されることとなる．Chat ベンチマークでは，チャットルームの数および各ユーザが送受信するメッセージの個数をパラメータとして渡すことができる．本章の評価では，30 チャットルーム，1 ユーザあたり 300 メッセージに設定した．この場合，2,400 プロセス（スレッド）がシステム上に生成される．したがって WebBench と比較して，さらに高い負荷がサーバにかかる．

5.2 実験結果

【WebBench】

サーバの CPU 数を 1 台から 8 台まで変化させた場合の標準カーネル，カラーリングカーネル，O(1) カーネルの Web リクエスト処理性能を**図 9** に示す．

図から分かるように，カラーリングしたカーネルは，標準カーネルに比べ大きくスケーラビリティが改善されている．1 秒あたりに処理したリクエスト数は，8CPU の場合で最大 23.3%増加している．

CPU 数が増加するにしたがってカラーリングによる性能向上比は大きくなっている．これは，キャッシュミスが減少して runqueue の走査時間が短縮された結果，単一の runqueue におけるロック競合が軽減されたためと考えられる．これを確かめるために，複数 CPU 構成のシステム上で runqueue 走査時におけるロック競合率を Lockmeter [18] を用いて測定した．結果を**表 3** に示す．この結果から分かるように，カラーリングを行うことで，ロック競合が発生する割合が 8CPU の場合に 45.6%から 23.4%にまで大きく低下している．

表 3 runqueue 走査時におけるロック競合率
Table 3 Lock contentions during runqueue traversal.

		2 CPU [%]	4 CPU [%]	8 CPU [%]
WebBench	標準	9.2	19.7	45.6
	カラーリング	2.3	6.0	23.4
Chat	標準	33.0	51.9	85.8
	カラーリング	23.6	48.3	69.7

また，カラーリングカーネルと O(1) カーネルの性能を比較すると，いずれの CPU 数においてもその性能差はほとんどみられない．WebBench での実験では，サーバ上で起動・動作する httpd プロセスの総数は 256 個であり，この程度のプロセス数であれば，カラーリングによって十分にスケジューリングオーバヘッドが軽減できている．

【Chat】

標準カーネルおよびカラーリングカーネルの Chat ベンチマークにおけるスループット性能を**図 10** に示す．グラフから分かるように，カラーリングを行うことでスケーラビリティが大幅に向上している．標準カーネルでは，4CPU までは性能が若干向上するが，それより CPU 数が多い場合には性能が劣化する．これに対して，カラーリングしたカーネルでは，6CPU まで性能が大きく向上している．このとき，標準カーネルに対して 89.6%もの性能向上が得られている．WebBench の場合と同様に，runqueue 走査におけるロック競合率を採取した．表 3 に結果を示す．これから分かるように，ロック競合率が大きく減少している．Chat ベンチマークプログラム実行中，runqueue にリンクさ

れる実行可能プロセスは平均して 1,000 以上にも及ぶ．そのため，スケジューラは，前節の WebBench による実験の場合（256 プロセス）よりも多くのプロセスが並んだ runqueue を走査しなければならない．そのためカラーリングによるロック競合率の低減効果がより顕著に現れている．

O(1) カーネルについては，WebBench での結果と異なり，カラーリングカーネルとの性能差が現れている．プロセス数が 1,000 を超えるような高い負荷がサーバにかかる場合には，実験に用いたシステムの場合，カラーリングによってキャッシュミスを低減するだけでは不十分である．

5.3 考　　察

本章では，カラーリングカーネルとの性能比較のために O(1) カーネルを取り上げた．O(1) カーネルでは，標準カーネルに対してデータ構造やアルゴリズムに大幅な変更がなされているため，現在運用中のサーバシステムに適用した場合にその安定性が保証されるとは言い難い．本項でのベンチマークの結果から，カラーリングによるキャッシュアクセスの最適化のみで，線形探索を行う従来の単純なスケジューラの構造のままで高負荷に対応することができている．この改良は，若干のコード修正しか必要としないため O(1) と比べ有用である．Chat ベンチマークでは，カラーリングカーネルと O(1) カーネルとの性能差が現れたが，実験で用いたシステムは，搭載された 2 次キャッシュメモリのサイズが 1MB であり，近年の大容量キャッシュを使用すればカラーリングによってさらなる性能向上も期待できる．

一般的な OS において，プロセススケジューリング処理を実装する場合に，単一または複数の線形リストからなるプロセスキューを設けることが多い．Linux カーネルもまた，単一の線形リストでプロセスを管理し，プロセスディスパッチ時に全探索を行うというシンプルな構成を持っていた．一般に，既存の OS は，プロセス管理だけではなく様々な部分でキュー構造（リスト構造）を多用している[10),20)]．ポインタをたどるリスト探索処理は，元来メモリに対してランダムアクセスを発生するためにキャッシュメモリが有効に機能しないという問題点を内在している．しかし，キャッシュメモリの最適化を十分に行うことができれば，現在の高速な動作周波数を持つ CPU の性能を最大限に引き出すことが可能となり，データ構造やアルゴリズムを大きく変更するのと同等な効果をあげることも可能である．このような最適化を行ううえで，ソフトウェア開発者がメモリボトルネックを発見することが困難な場合には，本論文で使用した GATES が有効である．

以上のように，キャッシュアクセスを最適化するタスク構造体のカラーリングは，O(1) カーネルとは異なり，若干の修正でキャッシュアクセスを最適化して性能を向上することができる．本手法は，サーバ分野で運用されているカーネルに対しても安定性を損なわずに適応可能であり，大規模な SMP マシン上での Linux システムのスケーラビリティ向上に対して有効といえる．

6. 関 連 研 究

Linux スケジューラに関して，その高速化のためにいくつかの手法が提案されている．

Kravetz らは大規模 SMP マシン上での Linux2.4.x カーネルのスケーラビリティを向上するために，multi-queue スケジューラを提案している[3)]．multi-queue スケジューラは，CPU ごとに runqueue を分割して管理し，各 runqueue ごとにロック変数を持たせている．これにより，ロック競合が低減され，スケーラビリティが向上する．ロックの競合を抑えるという点において，カラーリングと効果を同じくしているが，カラーリングは 1 本の runqueue をサーチする時間を短縮することでこれを実現している．これら 2 つのスケジューラ高速化手法は併用することもでき，この場合さらなるスケーラビリティ向上が期待できる．

1 本のキューを優先度によって分割し管理する手法として，他に ELSC スケジューラ[2)] が提案されている．ELSC スケジューラは，スケジューラ内部において優先度計算を行う `goodness` 関数に着目している．この関数内で静的に計算可能な部分をあらかじめ求めておき，その値に基づいて 1 本のキューを 30 本の順序づけられた複数のキューに分割して管理する．また，優先度によってキューを分割する手法としては，他にも Priority Level スケジューラ[3)] も提案されている．しかし，いずれの手法も高負荷ではロック競合を抑えることができず，スケーラビリティの向上は小さいことが報告されている．

また，runqueue 走査時のキャッシュミスについては，Sears が文献 19) にてプリフェッチによる解決手法を提案している．実際，すでに最新版のカーネル（2.4.10 以降）にこの改良は組み込まれている．しかし，プリフェッチが十分効果を発揮するためには，メモリアクセス遅延時間と優先度計算にかかる時間とが十分オーバラップできなければならない．これらの値はマシン構成に依存するので，この点で十分な解決と

はいえない．

Linux カーネルに関する開発は，LKML（Linux Kernel Mailing List）を中心にスピーディに行われている．我々は，この LKML にて本論文で提案した改良コードをすでに提供しているが，これと同時期に，Spraul がスラブアロケータ[11] からタスク構造体を確保することでカラーリングを実現するための修正コードを提供している．この手法は，本論文で提案した手法のようにカラーリング数を固定する必要がなく，キャッシュサイズなどのハードウェア構成に依存しない．この点で優れた手法であるといえる．本研究では，タスク構造体のカラーリングがどの程度の効果を発揮できるかを調べるのが目的であった．それには，本論文で報告している get_current 関数へのわずかな修整という実装が最も効率的でかつ十分なものであった．しかし，実際のカーネルに組み込むためには，Spraul の提案を含め十分な検討が必要である．

7. おわりに

本論文では，IA-32 をプラットフォームとした Linux カーネル内部において，スケジューリング時に頻繁にキャッシュミスが発生し，これによって大幅なシステム性能低下が発生することを述べた．この問題を解決するために，タスク構造体に対するキャッシュカラーリング手法の提案，実装，および性能評価を行った．評価の結果，本カラーリング手法により標準カーネルと比較して 8-way Pentium III server 上で最大 23.3%の Web トランザクション性能の向上がみられた．また，Chat を用いた場合，メッセージスループット性能が最大約 89.6%向上した．キャッシュカラーリングを使用する場合には，対象システムのキャッシュサイズやプロセス数などを考慮したカラーリング数を設定することが重要となる．本論文では，カラーリングが有効に機能するためのカラーリング数の簡単なモデルを示した．これに基づいて適切なカラーリング数を設定した場合には，スケジューラのデータ構造やアルゴリズムを大幅に変更するアプローチと同等の効果をもたらすことを確認した．

本論文で問題となった線形リスト構造は，従来からシステムソフトウェア内部において多用されているデータ構造である．しかし，探索時に発生するメモリに対するランダムアクセスは，キャッシュメモリシステムとの親和性に乏しく，システムプログラマはこの点に十分な配慮が必要である．最適化のために本研究で行ったメモリチューニング手法は，Linux に限らずその他の OS，あるいはそのうえで動作するアプリケーションプログラムに対しても十分応用が可能であり，上記のような取り組みをサポートする有効な手法となり得る．

今後は，さらに本ツールの適用範囲を拡大するとともに，エンタープライズ分野への適用に向け Linux カーネルのチューニングやそのうえで動作するアプリケーションの性能評価を継続して行う予定である．

なお，本カラーリング方式の改良コードは，以下のホームページよりダウンロード可能である．

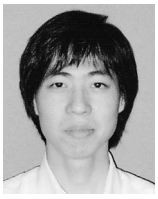
「技術情報：Linux カーネルに関する情報」
http://www.labs.fujitsu.com/techinfo/linux

参考文献

1) Linux Scalability Effort Project. http://sourceforge.net/projects/lse
2) Molloy, S. and Honeyman, P.: Scalable Linux Scheduling, *CITI Technical Report 01-7*, University of Michigan (May 2001).
3) Kravetz, M., Franke, H., Nagar, S. and Ravindran, R.: Enhancing Linux Scheduler Scalability, *5th Annual Linux Showcase & Conference* (Nov. 2001).
4) 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 (Sep. 1999).
5) Yamamura, S., Hirai, A., Sato, M., Yamamoto, M. Naruse, A. and Kumon, K.: Speeding Up Kernel Scheduler by Reducing Cache Misses, *Proc. USENIX 2002 Annual Technical Conf. FREENIX Track*, pp.275–285 (June 2002).
6) 佐藤　充，成瀬　彰，久門耕一：メモリバストレースを用いた共有バス型並列計算機のキャッシュ評価，情報処理学会研究会報告，2000-ARC-139, pp.1–6 (Aug. 2000).
7) 佐藤　充，成瀬　彰，久門耕一：メモリトレースを元にした大規模サーバの性能予測，情報処理学会研究会報告 2001-ARC-144，pp.13–18 (Sep. 2001).
8) 平井　聡，山本昌夫，佐藤　充，成瀬　彰，久門耕一：NUMA マシンでのコマーシャルワークロード向け Linux 最適化，情報処理学会論文誌，第 43 巻，第 4 号別冊 (2002).
9) Bovet, D.P., Cesati, M.（著），高橋浩和，早川仁（監訳），岡島順治郎，田宮まや，三浦広志（訳），詳解 LINUX カーネル，オライリー・ジャパン (July 2001).
10) Mauro, J. and McDougall, R.: *Solaris Internals Core Kernel Architecture*, Sun Microsystems Press.
11) Bonwick, J.: The Slab Allocator: An Object-

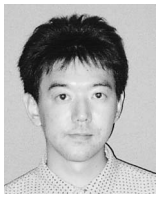
Caching Kernel Memory Allocator, *USENIX Conference Proceedings*, pp.87–98 (1994).
12) Bovet, D.P. and Cesati, M.: *Understanding the Linux Kernel*, pp.69–70, O'Reilly & Associates (Oct. 2000).
13) WebBench Homepage. http://etestinglabs.com/benchmarks/webbench/webbench.asp
14) [announce] [patch] ultra-scalable O(1) SMP and UP scheduler, Linux kernel mailing list. http://marc.theaimsgroup.com/?l=linux-kernel&m=101010394225604&w=2
15) 谷口宏樹，石川 裕，平木 敬：SMP 環境における Linux スケジューラの評価，信学技報，CPSY2002-17, pp.41–47 (June 2002).
16) Bryant, R. and Hartner, B.: Java Technology, Threads, and Scheduling in Linux, *Java Technology Update*, 4(1) (Jan. 2000).
17) Linux Benchmark Suite Homepage. http://lbs.sourceforge.net/
18) Bryant, R. and Hawkes, J.: Lockmeter: Highly-Informative Instrumentation for Spin Locks in the Linux Kernel, *4th Annual Atlanta Linux Showcase & Conference* (Oct. 2000).
19) Sears, C.B.: The Elements of Cache Programming Style, *4th Annual Atlanta Linux Showcase & Conference* (Oct. 2000).
20) Vahalia, U.: UNIX Internals: The New Frontiers, Prentice Hall.

(平成 14 年 12 月 21 日受付)
(平成 15 年 2 月 14 日採録)



山村 周史（正会員）

1998 年京都工芸繊維大学大学院電子情報工学科修士課程修了．2001 年同大学院情報・生産科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．プロセッサアーキテクチャに関する研究に従事．Linux システムの性能評価・チューニングに興味を持つ．電子情報通信学会，IEEE 各会員．



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．

佐藤 充（正会員）

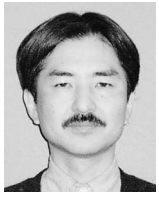
1969 年生．1992 年東京大学工学部電気工学科卒業．1997 年同大学大学院工学系研究科情報工学専攻博士課程修了．博士（工学）．同年富士通（株）入社．現在,（株）富士通研究所勤務．並列システムアーキテクチャの研究に従事．IEEE，ACM 各会員．

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．

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終了．1984 年同課程博士課程中退．同年（株）富士通研究所入社．現在，同社 IT コア研究所に所属．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．