Vol.2009-OS-112 No.1
2009/8/5

マルチコアCPU環境におけるL2キャッシュの影響を考慮したVMスケジューラ

本橋 剛[†1] 山田浩史[†1,†2]
吉田哲也[†1] 河野健二[†1,†2]

仮想化技術を用いたサーバ統合化のためのマシンの CPU として Chip Multiprocessor (CMP) が広く普及している．CMP は一つのチップに複数のコアを持ち，同一チップ上に載っているコア同士で共有の L2 キャッシュを持つ．L2 キャッシュミスは CPU の性能を大きく低下させるため，CMP のスケジューリングではコア同士によるキャッシュの競合に注意する必要がある．CMP に対応したプロセススケジューラの研究がされているが，仮想化環境には対応していない．本研究では，仮想化環境において CMP の L2 キャッシュミスが性能に与える影響を調査し，L2 キャッシュを考慮した仮想マシン (VM) スケジューラの設計を行った．Xen 3.3 を用いて実験を行ったところ，VM 上で Memory-bound なワークロードを実行したとき，最大で約 540%のスループットの差があることがわかった．実験結果より，L2 キャッシュを考慮したスケジューリングを行うためには 2 つの要素が必要であることがわかった．第一に，VM 上で実行するワークロードが L2 キャッシュの影響を受けやすいかを判別することである．第二に，VM と L2 キャッシュの状態を基に，L2 キャッシュが有効に働くように仮想 CPU をスケジューリングすることである．

L2 cache aware VM scheduler for multi-core CPU environment

Tsuyoshi Honbashi,[†1] Hiroshi Yamada,[†1,†2]
Tetsuya Yoshida[†1] and Kenji kono [†1,†2]

A chip multiprocessor (CMP) architecture is becoming widely used for server systems which often consolidate many servers through system virtualization. On a CMP architecture, one CPU chip has multiple processing cores sharing an L2 cache, and thus cores on the same chip compete the shared L2 cache. Since L2 cache misses severely degrade the CPU performance, we carefully pay attention to an L2 cache state in scheduling tasks on CMP. While many researchers are proposing several process schedulers for CMP, a virtual machine (VM) scheduler for CMP is also demanded. In this paper, we study an impact of L2 cache misses on a virtualized system constracted on a CMP machine, and design an VM L2 cache aware scheduler. We conducted some experiments using Xen 3.3 and micro-benchmarks. The experimental results report that our memory-bound benchmark running on a VM is severely affected by L2 cache behavior. This observation brings two requirements for a novel L2 cache aware VM scheduler. First, we need a mechanism to check whether workloads running on a VM are sensitive to L2 cache miss or not. Second, we need to schedule virtual CPUs, taking into account the state of VMs and L2 cache.

1. はじめに

サーバマシンの CPU として chip multiprocessor (CMP) が広く普及している．CMP は 1 つのチップに複数のコアを持ち，同一チップ上に載っているコア同士で共有キャッシュを持つ．共有キャッシュを持つことで，プロセッサ間通信の速度向上や，キャッシュミスの減少が見込める．CMP を採用した CPU として Intel の Xeon や，AMD の Opteron などがある．サーバ用途の CPU のトップシェアを持つ Intel は 2015 年までに数十から数百の物理コアを持つメニーコア CPU の生産を目指している[1)]．よって，今後も CPU の物理コア数の増加は続いていくと考えられる．

CPU の性能が低下する主な要因の一つはキャッシュミスであることが知れらている[2)]．特に，L2 キャッシュミスはメモリアクセスを伴うため，性能に大きな影響を与える．CMP では，複数のコアが L2 キャッシュを共有するので，共有キャッシュの競合が発生し，キャッシュミスの増加を招きやすい[3)]．

L2 キャッシュミスを減少させるアプローチの一つとして，CPU の物理的なキャッシュサイズの増加が挙げられる．しかし，キャッシュサイズを増やすにはハードウェアの変更による再設計のコストや，増加分のキャッシュメモリのコストなど，大きなコストがかかる．また，キャッシュサイズを増やすことによりキャッシュ面積が増加し，コア用のスペースが減ってしまう．そのため，チップに載せるコア数を増やせなくなるという問題もある．こうした

†1 慶應義塾大学
Keio University
†2 科学技術振興機構
CREST

ことから，ソフトウェアが CMP の L2 キャッシュを効率的に用いることで L2 キャッシュミスを抑えることが必要である[4]．

そこで現在，CMP に対応したプロセススケジューラを備えた OS の研究が行われている[5)–7)]．一方，仮想化環境における CMP の対応は遅れている．近年，仮想化技術を用いたサーバの統合化が行われている．サーバの統合化とは，複数台のサーバを 1 台の物理マシン上の仮想マシン (VM) で稼働させる手法である．VM を管理する仮想マシンモニタ (VMM) の役割の 1 つに，物理マシンの CPU リソースの管理がある．VMM 上では複数の VM を稼働させるため，仮想マシン (VM) に効果的に CPU リソースをスケジューリングしなければスループットが落ちてしまう．例えば，複数の VM 上でそれぞれ異なるワークロードを動作させたとき，同一チップ上の物理コアで同時に稼働する VM の組み合わせによっては L2 キャッシュミスが増加し，性能が低下する．という研究結果が報告されている[3]．こうしたことから，仮想化の統合化環境において CMP の L2 キャッシュを効率的に利用する VM スケジューラが求められている．

本研究では，仮想化環境において CMP の L2 キャッシュミスがスループットに与える影響を調査し，L2 キャッシュを考慮した VM スケジューラの設計を行う．仮想化環境における L2 キャッシュの影響を調べるために，CMP を搭載する 1 台の物理マシン上で仮想 CPU を 2 つ持つ VM を複数稼働させ，L2 キャッシュを考慮したスケジューリングの場合と，考慮していないスケジューリングの場合のそれぞれで VM 上で様々なワークロードを実行した．

実験結果より，VM 上で Memory-bound なワークロードを実行したとき，L2 キャッシュを考慮したスケジューリングと，そうでないスケジューリングとでは最大で約 540%のスループットの差があることがわかった．また，既存の VMM である Xen の VM スケジューラが L2 キャッシュを考慮したスケジューリングを行っているかについて分析した．その結果，Xen の VM スケジューラは L2 キャッシュを考慮に入れないスケジューリングを行うため，スループットが低下することがわかった．実験結果を基に，L2 キャッシュの振る舞いを考慮した VM スケジューラを設計した．提案スケジューラでは，L2 キャッシュミスの影響を大きく受ける VM を判別するために VM 上で動作するワークロードを判別する．また，VM の状態と L2 キャッシュの状態を基に，L2 キャッシュが有効に働くように仮想 CPU をスケジューリングする．

本論文の構成は以下の通りである．2 章では関連研究について説明する．3 章では VM スケジューリングについての実験を行い，その結果について分析する．4 章では L2 キャッシュを考慮した VM スケジューラの設計について考察する．最後に 5 章で本研究の内容をまとめると共に，今後の課題について考察する．

2. 関 連 研 究

2.1 プロセススケジューラ

CMP の L2 キャッシュの影響を考慮したプロセススケジューラが提案されている．

Cheny ら[6] は，変数やロックを共有するスレッドを同時に実行することでキャッシュの再利用率を高めるコンストラクティブキャッシュシェアリングの評価を行っている．複数のコアがそれぞれ別のタスクを実行すると，共有キャッシュとメインメモリへの帯域の競合が発生する．コンストラクティブキャッシュシェアリングではワーキングセットがオーバーラップしてるスレッドを同時に実行して全体のワーキングサイズを節約する．従来の Work Stealing (WS) スケジューラとコンストラクティブキャッシュシェアリングを実装する Parallel Depth First (PDF) スケジューラを CMP 環境で比較した結果，PDF が良い性能を発揮することを実証した．仮想化環境では VM 同士が共有リソースを持たないため，PDF を適用するのは困難である．

Thread Clustering[5] は CMP 環境下において，変数やロックを共有するスレッド同士を同じチップ上の物理 CPU にスケジューリングすることで，オーバーヘッドとなるチップ外へのメモリアクセスを減らす手法である．マルチスレッドプログラミングでは共有変数やロックなどへのアクセスがボトルネックとなる．そこで，Thread Clustering では各スレッドごとに一定期間中のメモリへのアクセスを記録する．そして，全てのスレッドのメモリアクセスを比較して類似しているアクセスパターンのスレッドをクラスタリングする．仮想化による統合化環境で動くワークロードの場合，異なる VM 同士で変数やロックを共有することは無いため，Thread Clustering を仮想化環境に適用することは困難である．

2.2 VM スケジューラ

前述したように，OS のプロセススケジューラに関しては，CMP に対応するための手法が研究・提案されている．一方，VM スケジューラを CMP に対応させるための研究は行われていない．

2.2.1 マルチプロセッサに対応した VM スケジューラ

Disco[8] は MIPS アーキテクチャの CPU 上で動作する VMM である．Disco では，以前に仮想 CPU を割り当てた物理 CPU に，再び仮想 CPU を割り当てる Affinity スケジューリングを行う．こうすることで，キャッシュの再利用性を高めている．Cellular Disco[9] は Disco を拡張したものである．物理 CPU ごとに仮想 CPU のキューを持つ．アイドル状態

の物理 CPU が他の物理 CPU から仮想 CPU を奪うギャングスケジューリングを行うことで負荷分散する．さらに，定期的に物理 CPU 全体のグローバルな負荷分散を行っている．Disco と Cellular Disco 共に，Symmetric multiprocessor (SMP) での利用を前提としており，CMP については考慮していない．

Xen[10] は x86 アーキテクチャの CPU 上で動作するオープンソースの VMM で，商用のサーバでも利用されている．現在のバージョンの Xen はクレジットスケジューラを採用している[11]．クレジットスケジューラは物理 CPU 間の負荷分散を行うことで，SMP を効率よく利用する．物理 CPU 間の負荷バランスが崩れた時に仮想 CPU が動作する物理 CPU を変更することにより，物理 CPU 間の負荷バランスを取る．クレジットスケジューラは物理 CPU 間の負荷分散を重視しており，CMP の共有キャッシュの利用に関しては考慮していない．

Time ballooning[12] は仮想 CPU に割り当てる物理 CPU のシェアを動的に変更する．ゲスト OS にバルーンモジュールを疑似デバイスドライバとしてロードすることで．ゲスト OS のスケジューリングコードに変更を加えることなく VMM がゲスト OS 内の負荷分散を行うことを可能にする．バルーンモジュールは仮想 CPU の速度の偏りを検出したとき，仮想的なワークロードを作り出してゲスト OS のスケジューリングを調整する．Time ballooning は物理 CPU の使用率を上げることを目的としており，本研究の目的とは異なる．

2.2.2 VM 上で実行するワークロードを考慮した VM スケジューラ

VM スケジューリングを効果的に行うために，VM 上で実行するワークロードを考慮してスケジューリングする研究が行われている．

Ongaro ら[13] は VMM による I/O スケジューリングについての分析を行い，仮想環境での I/O 性能を向上させるための VM スケジューリングについて議論している．I/O-bound なワークロードを実行する VM と，CPU-bound なワークロードを実行する VM が混在するとき，I/O boosting や Run queue sort による最適化が有効であることを実証した．また，Response-sensitive なワークロードを実行するときは専用の VM で実行する必要性を示した．Kim ら[14] は仮想化環境での I/O 性能を向上させる VM スケジューラを考案した．VM 上で実行しているワークロードが I/O-bound であるかを識別し，実行するワークロードが I/O-bound であると判断した VM の優先度を上げることで応答時間を短縮する．これらの研究では Memory-bound なワークロードに関しては考慮していない．

3. VM スケジューリングの分析実験

本章では，L2 キャッシュを考慮した VM スケジューラを設計するために，L2 キャッシュミスが性能に与える影響の検証と VM スケジューリングの分析をするための実験を行った．

3.1 実験環境

VM スケジューラの挙動を知るため，物理 CPU で発生したイベントや，仮想 CPU のパラメータなどといった詳細な情報を取得した．これらの情報を得るために，本研究では，商用サーバに広く用いられているオープンソースである仮想化環境の Xen 3.3 に分析システムを実装した．

図 3.1 にシステムの概要を示す．VMM 内部でハードウェアパフォーマンスカウンタ (HPC) を用いて物理 CPU のイベントを監視する．HPC は CPU の Machine Specific Register (MSR) の一種[15] で，CPU 内部で起きたイベントを計測する．HPC はカウンタ用 MSR と制御用 MSR が対になっている．制御用 MSR でイベントカウントの有効/無効化と監視するイベントを設定する．指定したイベントの発行数がカウンタ用 MSR に格納される．HPC は各コアに独立に備わっているため，コアごとにデータを取ることができる．HPC は主にシステムのプロファイリングに用いられ，プロファイリング用のツール VTune[16] や OProfile[17]，PAPI[18] は内部で HPC を利用している．また，HPC を用いてシステムの挙動を解析する研究が行われている[19]．

VMM が定期的に HPC をサンプリングし，Xen VMM の内部メモリに内容を出力する．サンプリングデータを出力するために，ドメイン 0 から Xen VMM の内部メモリに格納されたサンプリングデータを取得し，取得したデータの内容をファイルに出力する．出力したファイルを解析して分析に用いる．

VM スケジューラの挙動の分析のために，HPC の値と各仮想 CPU の状態をタイマ割り込みをきっかけに定期的に出力する．本研究では，Cycles Per Instruction (CPI) を性能評価の指標の一つとして用いる．CPI を求めるために halt 状態でないコアサイクル数を得るための `CPU_CLK_UNHALTED.CORE_P` と完了した命令数を得るための `INST_RETIRED.ANY_P` を HPC を用いてモニタリングする．また，仮想 CPU がどの物理 CPU で実行されたかを知るために，スケジューリングした仮想 CPU の ID，仮想 CPU が属する VM の ID，仮想 CPU がスケジューリングされた物理 CPU の ID，物理 CPU が属する物理チップの ID を出力する．

実験に用いたマシンの構成は表 3.1 に示した通りである．特に，CPU の構造を図 3.2 に

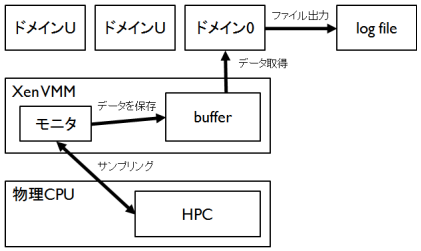

図 3.1 分析システムの概要図

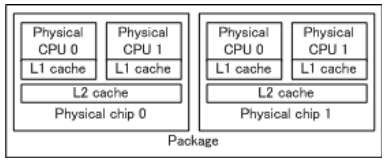

図 3.2 Intel Core2 Quad の構造

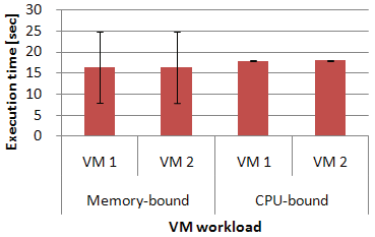

図 3.3 クレジットスケジューラによるワークロードの実行時間

表 3.1 物理マシンの構成

OS	Linux 2.6.18
VMM	Xen 3.3.0
CPU	Intel Q6600 2.4GHz L1 cache size 64KBx4 L2 cache size 4MBx2
Memory	2GB

表 3.2 VM の構成

OS	Linux 2.6.18
仮想 CPU 数	2
Memory	256MB

示す．CPU は Intel Core2 アーキテクチャの Quad コア CPU で，1 つのパッケージ内に 2 つの物理チップを搭載する．各物理チップは，それぞれ 2 つの物理 CPU と 1 つの L2 キャッシュを持ち，L2 キャッシュは 2 つの物理 CPU で共有する．VMM は前述の分析システムを実装した Xen を用いた．VM の構成を表 3.2 に示す．

VM 上で動作させるワークロードとして，systbench 4.0[20] を用いる．sysbench は CPU-bound, Memor-bound, I/O-bound なワークロードを生成できるベンチマークである．CPU-bound なワークロードでは素数判定を行う．Memory-bound なワークロードではメモリへの書き込みを行う．I/O-bound なワークロードではファイル I/O のランダムリード/ライトを行う．単純なワークロードを実行することで，どのような場合に，どのようなワークロードを実行したときに性能に影響が出るかを明確化する．

3.2 既存の VM スケジューラの挙動の分析

現在の Xen のクレジットスケジューラの挙動を分析する．まず，2 つの VM 上で Memory-bound，CPU-bound なワークロードを実行した場合の実行時間を測定する．Memory-bound なワークロードは L2 キャッシュによる影響が大きいと考えられる．

実行結果を図 3.3 に示す．横軸はワークロードの種類を示し，縦軸は実行時間を示す．CPU-bound なワークロードを実行した場合の実行時間の分布は平均実行時間の± 2%以内に収まっているのに対して，Memory-bound なワークロードを実行した場合の実行時間の分布は平均実行時間の± 50%以上と広い分布になっている．

平均実行時間の分布の違いの原因を調べるため，2 つの VM 上で Memory-bound なワークロードを実行したときの CPI の分布を測定する．実行結果を図 3.4 に示す．図 3.4 は CPI に関しての確率分布を示すヒストグラムになっている．また，L2 キャッシュの影響が少ないと考えられる CPI の分布と比較するために CPU-bound なワークロードを実行したときの CPI の分布を測定した．実行結果を図 3.5 に示す．図 3.4 より，Memory-bound なワークロードを実行した場合は分散が大きく，二つの山ができていることがわかる．一方，CPU-bound なワークロードを実行した場合は図 3.5 のように分散の小さな一つの山ができるのみである．よって，クレジットスケジューラで Memory-bound なワークロードを実行すると，L2 キャッシュの影響により，CPI が低くなる場合と CPI が高くなる場合があると考えられる．

VM 上で Memory-bound なワークロードを実行したとき CPI の分布で二つの山ができる原因を調べるため，Memory-bound なワークロードを実行したときの物理チップのスケジューリングパターンと CPI を測定した．スケジューリングパターンとは，物理チップ上の各物理 CPU にスケジューリングされている仮想 CPU の組み合わせを指す．測定の結果，合計 56 種類のスケジューリングパターンを確認した．その中から，頻度の高かった 12 種類のスケジューリングパターンの一覧を表 3.3 に示す．

CPI の分布を測定した結果を図 3.6 に示す．横軸はスケジューリングパターンを表し，縦軸は CPI の分布を箱ひげ図によって表したものである．スケジューリングパターン ID 2, 4, 13, 36 の 4 種類の CPI の分布を見ると，他のスケジューリングパターンに比べて，20%近くまで小さくなっている．CPI は 0.5 の近傍に分布しており，これは図 3.4 の 1 つ目の山

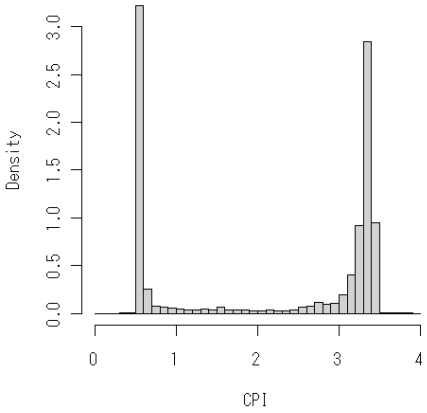

図 3.4 Memory-bound workload の CPI の分布

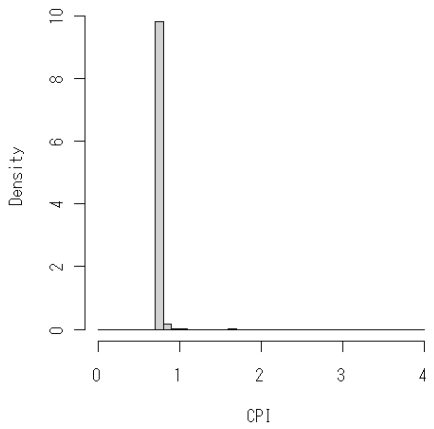

図 3.5 CPU-bound workload の CPI の分布

の分布に対応している．CPI が小さくなるのスケジューリングパターンに共通していることは，VM に属する仮想 CPU が全て同一の物理チップ上にスケジューリングされていることである．これにより，仮想 CPU 間で L2 キャッシュが効果的に利用され，CPI の低下につながったと考えられる．CPI が小さくなるスケジューリングパターンを一般化したものを図 3.7 に示す．このように，VM に属する全ての仮想 CPU が同一の物理チップ上にスケジューリングされている状態を *All share* と定義する．

また，スケジューリングパターン ID 2, 4, 13, 36 を除く 8 種類のスケジューリングパターンに共通することは，別々の VM に属する仮想 CPU が同一の物理チップ上にスケジューリングされていることである．別々の VM が実行する Memory-bound なワークロードが L2 キャッシュの競合を引き起こし，L2 キャッシュミスが増えた結果，CPI が上昇したと考えられる．CPI は 3.2 の近傍に分布しており，これは図 3.4 の 2 つ目の山の分布に対応している．CPI が大きくなるスケジューリングパターンを一般化したものを図 3.8 に示す．このように，VM に属する仮想 CPU が別々の物理チップ上にスケジューリングされている状態を *Partial share* と定義する．

スケジューリングによる性能差を比較するために All share と Partial share，クレジットスケジューラの 3 種類のスケジューリングについて，2 つの VM 上で Memory-bound なワークロードを実行し，実行時間を測定した．実行結果を図 3.9 に示す．横軸はスケジューリングを示し，縦軸はクレジットスケジューラの実行時間で正規化した値を示す．Partial share がもっとも悪いスループットを示す．クレジットスケジューラは Partial share の半分以下の実行時間である．All share はクレジットスケジューラの半分以下の実行時間となっている．クレジットスケジューラは図 3.6 のように，All share と Partial share が入り混じったスケジューリングとなるため，その中間のスループットになったと考えられる．

表 3.3 物理チップ上のスケジューリング

Scheduling pattern ID	Physical CPU 0	Physical CPU 1
2	VM 2 Virtual CPU 0	VM 2 Virtual CPU 1
4	VM 1 Virtual CPU 1	VM 1 Virtual CPU 0
6	VM 2 Virtual CPU 0	VM 1 Virtual CPU 0
7	VM 1 Virtual CPU 1	VM 2 Virtual CPU 1
10	VM 1 Virtual CPU 1	VM 2 Virtual CPU 0
11	VM 2 Virtual CPU 1	VM 1 Virtual CPU 0
13	VM 2 Virtual CPU 1	VM 2 Virtual CPU 0
17	VM 1 Virtual CPU 0	VM 2 Virtual CPU 0
19	VM 1 Virtual CPU 0	VM 2 Virtual CPU 1
23	VM 2 Virtual CPU 0	VM 1 Virtual CPU 1
24	VM 2 Virtual CPU 1	VM 1 Virtual CPU 1
36	VM 1 Virtual CPU 0	VM 1 Virtual CPU 1

以上のことから，クレジットスケジューラでは VM 上で Memory-bound なワークロードを実行するとき，L2 キャッシュを考慮せずにスケジューリングを行うため，最適なスケジューリングとは言えないことを実証した．L2 キャッシュを考慮したスケジューリングを行う必要がある．

3.3 同時に実行するワークロードの組み合わせが性能に与える影響

前節で 2 つの VM 上で Memory-bound なワークロードを実行したとき，スケジューリングによってスループットが増減することを示した．本節では，様々なワークロードを組み合わせて実行したときに性能に与える影響を調べる．

2 つの VM 上で別々のワークロードを実行したときに，L2 キャッシュの影響によって互いの VM の性能に与える影響を測定する．2 つの VM 上で，それぞれ CPU-bound, Memory-bound, I/O-bound, Idle のいずれかのワークロードを実行する．Idle はアイドル状態のワークロードを示す．その他のワークロードについては 3.1 節で説明した sysbench によって生成したワークロードである．1 つの VM では 1 つのワークロードのみを実行するものとする．Idle-Idle を除く 9 通りの組み合わせについて実行し，実行時間を測定した．また，ス

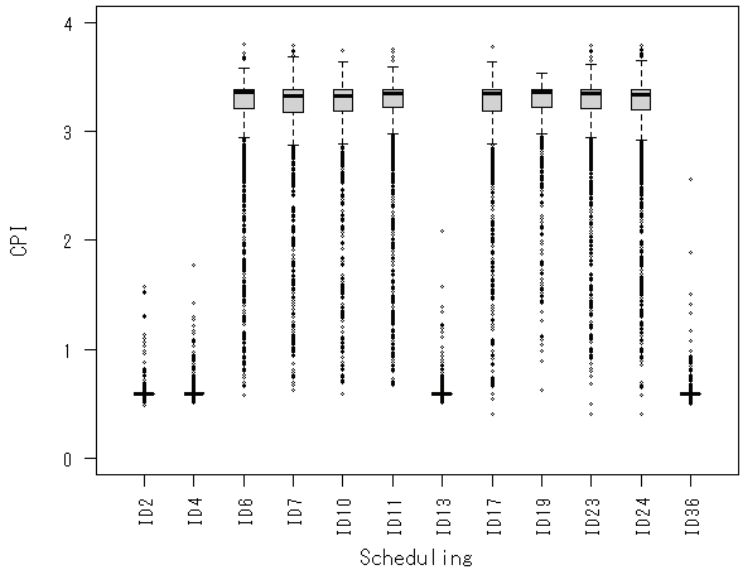

図 3.6 仮想 CPU のスケジューリングによる CPI の分布

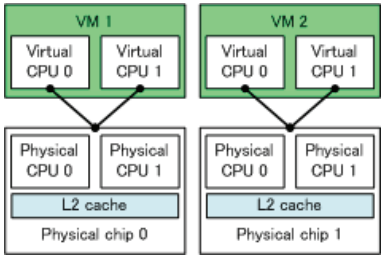

図 3.7 All share

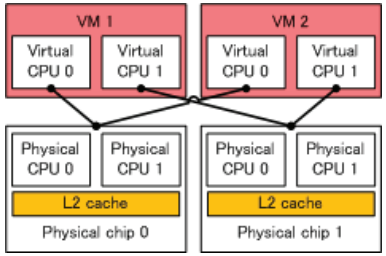

図 3.8 Partial share

ケジューリングが性能に与える影響を比較するために All share，Partial share の双方の場合について実験する．実験結果を図 3.10 に示す．横軸は VM 上で実行するワークロードの種類を表す．CPU，Mem，I/O はそれぞれ CPU-bound，Memory-bound，I/O-bound なワークロードを表す．まず，Memory-bound なワークロードを実行している VM は Partial

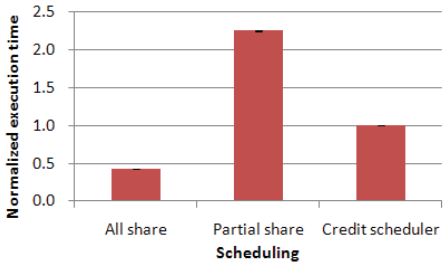

図 3.9 VM × 2 の実行時間

share では All share と比べて約 300%のスループットの差が出る．これは，Partial share では L2 キャッシュが効果的に利用出来なくなるためと考えられる．次に，Memory-bound なワークロードを実行している VM 同士の組み合わせでは大きく性能が落ち，All share と比較すると約 540%のスループットの差となる．これは，L2 キャッシュが効果的に利用出来なくなることに加えて，Memory-bound なワークロード同士でキャッシュの競合が発生するためキャッシュミスが増えるためと考えられる．

一方，CPU-bound, I/O-bound なワークロードを実行する VM では All share と Partial share の性能差は 5%以下にとどまった．これらのワークロードでは Memory-bound なワークロードに比べてメモリアクセスが少ないため，L2 キャッシュの影響が小さいためと考えられる．

以上のことから，Memory-bound なワークロードを実行する VM をどうスケジューリングするかが性能に大きな影響を与えると言える．また，実験結果から性能に影響を与える VM の状態と L2 キャッシュの状態を分類できる．VM の状態は All share, Partial share の 2 つの状態に分類できる．L2 キャッシュの状態は Clean, Dirty の 2 つの状態に分類できる．1 つの物理チップ上にスケジューリングされている Memory-bound な VM の数を n とすると，Clean は n が 1 以下，つまり，他の Memory-bound な VM によって L2 キャッシュが競合していないことを示す．Dirty は n が 1 より大きい，つまり，他の Memory-bound な VM によって L2 キャッシュが競合している可能性があることを示す．

3.4 VM の状態と L2 キャッシュの状態が性能に与える影響

前節で VM スケジューリングにおいて VM と L2 キャッシュの状態について分類した．これらの状態が性能へ与える影響を調べるため，VM と L2 キャッシュの状態の組み合わせを

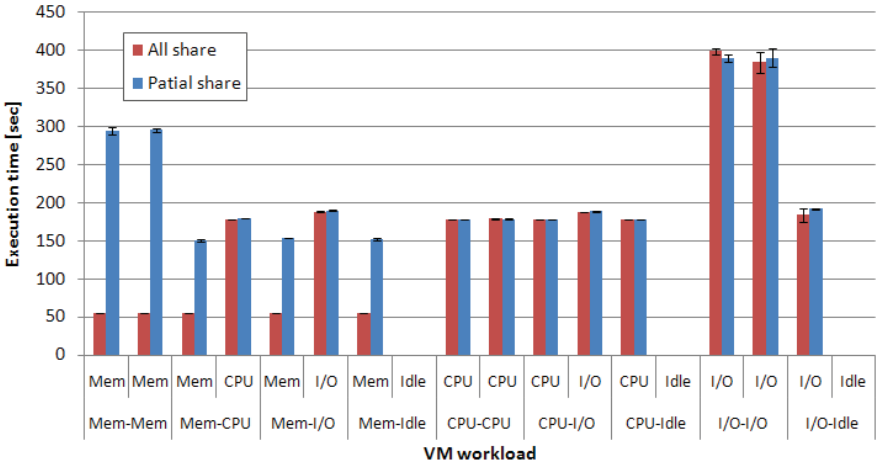

図 3.10 ワークロードの組み合わせによる実行時間

変えて VM 上で Memory-bound なワークロードを実行したときの CPI を測定する．状態の組み合わせによる CPI を比較することで共有キャッシュの影響の大きさを見る．状態の組み合わせは，All share-Clean, All share-Dirty, Partial share-Clean, Partial share-Dirty の 4 つである．All share-Clean は図 3.7 と同じ割り当てとした．Partial share-Dirty は図 3.8 と同じ割り当てとした．Partial share-Clean と Partial share-Dirty の各組み合わせについて，物理チップへの VM の仮想 CPU の割り当てを図 3.11, 3.12 に示す．

実験によって得られた各状態の組み合わせについての CPI の分布を図 3.13 に示す．横軸は状態の組み合わせを表し，縦軸は CPI の分布を箱ひげ図で表したものである．Clean と Dirty は Partial share のとき最大で 2 倍以上の CPI の悪化を招くが，All share のときは 10 数%の違いであり，影響が小さい．よって，Memory-bound なワークロードを実行する VM は All share になるようにスケジューリングする必要がある．

3.5 VM 数による性能の変化

前節までで VM が 2 つのとき，Memory-bound なワークロードを実行する VM を All share にするスケジューリングがスループットの面で有効であることを示した．VM 数が増えたときにもこのスケジューリングが有効かを調べるために，VM 数を増やしたときに性能がどのように変化するかを測定する．VM を 4 つに増やし，All share, Partial share, クレジットスケジューラそれぞれの場合について各 VM 上で Memory-bound なワークロードを実行し，実行時間を計測した．結果を図 3.14 に示す．横軸はスケジューリングを表し，縦軸はクレジットスケジューラの実行時間で正規化した値を表す．実験結果から，All share の実行時間はクレジットスケジューラよりも約 20%短い．よって，VM 数が増えたときにも Memory-bound なワークロードを実行する VM を All share にするスケジューリングが

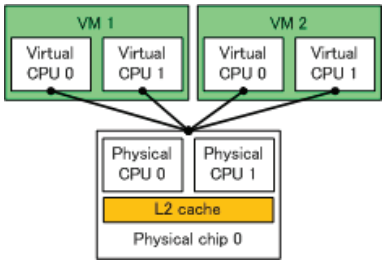

図 3.11 All share-Dirty

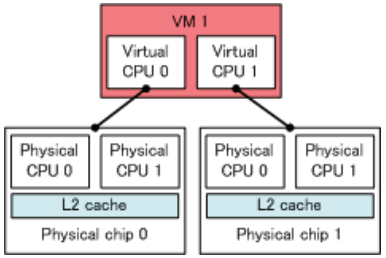

図 3.12 Partial share-Clean

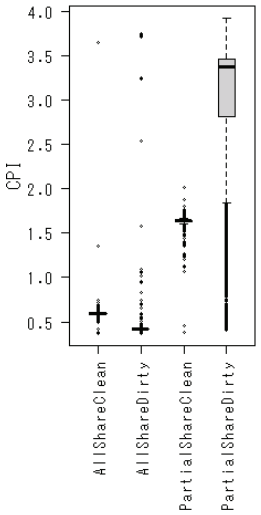

図 3.13 VM の状態と L2 キャッシュの状態の組み合わせによる CPI の分布

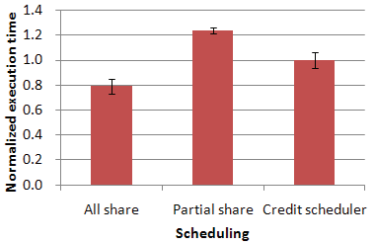

図 3.14 VM × 4 の実行時間

有効に動作すると言える．

4. L2 キャッシュを考慮した VM スケジューラ

前章で，VM 上で Memory-bound なワークロードを実行したとき，L2 キャッシュを効果的に利用しなければ VM のスループットが低下することを示した．また，既存の VM スケジューラは L2 キャッシュを考慮した VM スケジューリングを行っていないため，Memory-bound なワークロードを実行するときスループットが低下することを示した．以上のことから，L2 キャッシュを考慮した新たな VM スケジューラが必要であると言える．本章では，L2 キャッシュを考慮した VM スケジューラの設計について述べる．

4.1 スケジューリングアルゴリズムの概要

L2 キャッシュを考慮したスケジューリングを行うためには 2 つの機能が必要となる．第一に，VM 上で動くワークロードの種類，VM の状態，L2 キャッシュの状態等の VM をスケジューリングする上で必要な情報を得る機能である．3 章において，L2 キャッシュが VM の性能に与える影響は VM 上で実行するワークロードによって異なることを示した．特に，Memory-bound なワークロードは L2 キャッシュによる影響が大きい．よって，Memory-bound なワークロードを実行する VM は L2 キャッシュを効果的に利用できるようにスケジューリングをする必要がある．一方，L2 キャッシュによる影響が小さいワークロードを実行する VM は L2 キャッシュの利用よりも負荷分散を重視したスケジューリングをする必要がある．VM 上で動くワークロードに応じて適切なスケジューリングを行うためには，VM 上で実行しているワークロードの種類を VMM から判別する必要がある．また，Memory-bound な VM が有効に L2 キャッシュを利用できているかをを判別できること，Memory-bound な VM の移送先となる物理チップを判断するための L2 キャッシュの状態を判別できる必要がある．第二に，取得した情報を基に L2 キャッシュを効果的に利用できるように仮想 CPU をスケジューリングする機能である．

4.2 システムの状態の判別

4.2.1 VM 上で動くワークロードの判別

3 章の実験結果より，Memory-bound なワークロードを実行する VM の特徴は All share と Partial share それぞれで実行した場合の CPI の分布の差である．All share と Partial share の CPI の分散は小さく，ある値に集中している．また，集中している CPI の値に両者で明確な違いがある．Memory-bound なワークロード以外を実行する VM では，CPI の分布に明確な差は表れない．よって，VM が実行する未知のワークロードが Memory-bound であるかを判別するには，All share と Partial share それぞれでワークロードを実行して CPI を一定期間サンプリングする．得られたサンプリング結果の CPI 分布を統計処理によって比較し，その差が有意であれば Memory-bound な VM であると言える．

4.2.2 VM の状態の判別

VM の状態を判別するには，定期的に VM の仮想 CPU がどの物理チップにスケジューリングされているかを確認する．VM の仮想 CPU が全て同じ物理チップの物理 CPU にスケジューリングされていたら All share であり，そうでなければ Partial share である．

4.2.3 L2 キャッシュの状態の判別

L2 キャッシュの状態を判別するには各 VM が実行するワークロードの種類が判別されている必要がある．物理チップに割り当てられている Memory-bound なワークロードを実行する VM が 1 つならば Clean，複数ならば Dirty である．

実際にスケジューリングを行うには Clean，Dirty の 2 つの状態の他に，Memory-bound なワークロードを実行する VM によって使われていない L2 キャッシュであることを示す状態が必要である．この状態を Empty と定義する．Empty は Dirty を避けるようにスケジューリングするとき，仮想 CPU の移送先の物理チップを探すために必要になる．

4.3 スケジューリング

VM が L2 キャッシュミスによってスループットが低下することを防ぐために VM の状態を All share にするスケジューリングを行う．VMM では通常のクレジットスケジューラと同様の負荷分散を行う．Memory-bound なワークロードを実行する VM が All share 状態になったとき，その時点で VM を実行している物理チップの物理 CPU に全ての VM の仮想 CPU を関連付ける．こうすることで，All share の状態を保つことができる．

5. おわりに

本研究では，CMP の L2 キャッシュを効果的に利用する VM スケジューラの必要性を示し，L2 キャッシュを考慮した VM スケジューラの設計について示した．CPU の性能低下の要因の 1 つは L2 キャッシュミスである．L2 キャッシュが効果的に利用できるかによってスループットが大きく変動する．また，同じ物理チップ上で同時に実行する VM 同士の L2 キャッシュの競合がスループットに影響を与える．既存の VM スケジューラでは，L2 キャッシュが性能に与える影響を考慮せずにスケジューリングするため，スループットの低下を引き起こすことがある．本研究では，これらのことを実証する実験を行った．その結果，Memory-bound なワークロードを実行する VM は L2 キャッシュの影響によってスループッ

トが大きく変化することを確認した．

実験結果から，L2 キャッシュを考慮した VM スケジューラが備えるべき要件を二つ示した．第一に，システムの状態を取得する機能である．L2 キャッシュを考慮したスケジューリングをするためには，Memory-bound なワークロードを実行する VM，VM の状態，L2 キャッシュの状態等の情報を VMM が知る必要がある．第二に，システムの情報を基に L2 キャッシュを効果的に利用するために Memory-bound なワークロードを実行する VM の状態を All share に保つことである．

今後の課題は，本研究で示した VM スケジューラの実装である．また，既存の VM スケジューラとの比較実験を行う予定である．

参 考 文 献

1) Held, J., Bautista, J. and Koehl, S.: From a Few Cores to Many: A Tera-scale Computing Research Overview, Technical report, Intel Corporation (2006).
2) Fedorova, A., Seltzer, M., Small, C. and Nussbaum, D.: Performance of multithreaded chip multiprocessors and implications for operating system design, *Proceedings of the annual conference on USENIX Annual Technical Conference* (2005).
3) Apparao, P., Iyer, R., Zhang, X., Newell, D. and Adelmeyer, T.: Characterization & analysis of a server consolidation benchmark, *VEE '08: Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, New York, NY, USA, ACM, pp.21–30 (2008).
4) Fedorova, A., Small, C., Nussbaum, D. and Seltzer, M.: Chip multithreading systems need a new operating system scheduler, *EW11: Proceedings of the 11th workshop on ACM SIGOPS European workshop*, New York, NY, USA, ACM, p.9 (2004).
5) Tam, D., Azimi, R. and Stumm, M.: Thread clustering: sharing-aware scheduling on SMP-CMP-SMT multiprocessors, *EuroSys '07: Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, New York, NY, USA, ACM, pp.47–58 (2007).
6) Chen, S., Gibbons, P.B., Kozuch, M., Liaskovitis, V., Ailamaki, A., Blelloch, G.E., Falsafi, B., Fix, L., Hardavellas, N., Mowry, T.C. and Wilkerson, C.: Scheduling threads for constructive cache sharing on CMPs, *SPAA '07: Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures*, New York, NY, USA, ACM, pp.105–115 (2007).
7) Boyd-Wickizer, S., Chen, H., Chen, R., Mao, Y., Morris, F. K.R., Stein, A. P.L., Wu, M. and Zhang, Y. D. Y. Z.Z.: Corey: an operating system for many cores, *Proceedings of the Operating Systems Design and Implementaion (OSDI)* (2008).
8) Bugnion, E., Devine, S., Govil, K. and Rosenblum, M.: Disco: running commodity operating systems on scalable multiprocessors, *ACM Trans. Comput. Syst.*, Vol.15, No.4, pp.412–447 (1997).
9) Govil, K., Teodosiu, D., Huang, Y. and Rosenblum, M.: Cellular disco: resource management using virtual clusters on shared-memory multiprocessors, *ACM Trans. Comput. Syst.*, Vol.18, No.3, pp.229–262 (2000).
10) Bugnion, E., Devine, S., Govil, K. and Rosenblum, M.: Disco: running commodity operating systems on scalable multiprocessors, *ACM Trans. Comput. Syst.*, Vol.15, No.4, pp.412–447 (1997).
11) Chisnall, D.: *The definitive guide to the xen hypervisor*, Prentice Hall Press, Upper Saddle River, NJ, USA (2007).
12) Uhlig, V., LeVasseur, J., Skoglund, E. and Dannowski, U.: Towards scalable multiprocessor virtual machines, *VM'04: Proceedings of the 3rd conference on Virtual Machine Research And Technology Symposium*, Berkeley, CA, USA, USENIX Association (2004).
13) Ongaro, D., Cox, A.L. and Rixner, S.: Scheduling I/O in virtual machine monitors, *VEE '08: Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, New York, NY, USA, ACM, pp.1–10 (2008).
14) Kim, H., Lim, H., Jeong, J., Jo, H. and Lee, J.: Task-aware virtual machine scheduling for I/O performance., *VEE '09: Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, New York, NY, USA, ACM, pp.101–110 (2009).
15) Intel Corporation: *Intel® 64 and IA-32 Architectures Software Developer's Manual* (2008). `http://www.intel.com/products/processor/manuals/`.
16) Corporation, I.: VTune Performance Analyzer. `http://www.intel.com/cd/software/products/asmo-na/eng/vtune/`.
17) John Levon, P. E.: OProfile - A System Profiler for Linux. `http://oprofile.sourceforge.net/`.
18) Jack, D., Shirley, M., Phil, M., Keith, S., Dan, T. and You, H.: PAPI. `http://icl.cs.utk.edu/papi/`.
19) Shen, K., Zhong, M., Dwarkadas, S., Li, C., Stewart, C. and Zhang, X.: Hardware counter driven on-the-fly request signatures, *ASPLOS XIII: Proceedings of the 13th international conference on Architectural support for programming languages and operating systems*, New York, NY, USA, ACM, pp.189–200 (2008).
20) Kopytov, A.: SysBench: a system performance benchmark. `http://sysbench.sourceforge.net/`.