Vol.2012-OS-122 No.15
2012/8/1

仮想環境の特性を考慮したリソース調整機構の提案

東 賢一朗[1]

概要：Linux Kernel に組み込み使用する KVM(Kernel-based Virtual Machine) のように，既存 OS のリソース管理を利用する仮想化機構が存在する．このような仮想化機構の特長として，OS のリソース管理方針を利用した効率的な仮想化基盤の開発が実現できると同時に，既存 OS との親和性が高い仮想環境を提供できる．一方，仮想環境は既存 OS のリソース管理から直接影響を受けるため，個々の仮想環境の特性を考慮した柔軟なリソース管理を実現することは容易ではない．この課題を解決するために，仮想環境の特性を考慮したリソース調整機構を提案する．リソース調整機構は，仮想環境の管理方針をまとめた *policy* と，*policy* に基づき仮想環境間のリソース調整を実施する *peacemaker* とを実装している．試作したリソース調整機構を KVM に実装し性能実験を実施した結果，仮想環境の性能が約 16%向上し，リソース調整機構の有効性を確認することができた．

Proposal of the Resource Adjustment Mechanism for Virtual Machines

Kenichiro Higashi[1]

Abstract: There is a mechanism for virtualization of the Operating System to use the existing resource management. For example, KVM(Kernel-based Virtual Machine) uses Linux Kernel's resource management. The virtualization of such mechanism has not only efficiently virtualization development but the advantage of providing a virtual machines with high affinity existing the Operating System. On the other hand, it is not easy to realize flexible resource management in consideration of an individual virtual machine characteristic to come under an influence of the resource management of the existing OS directly. To solve this problem, This article proposes the resource adjustment mechanism considering the characteristics of the virtual machines. The resource adjustment mechanism has *policy* and *peacemaker*. *Policy* is summary of virtual machines policies. *Peacemaker* gets every virtual machine resource information and adjusts resource allocation using *policy* via the Operating System. Using prototype of the resource adjustment mechanism in KVM, Performance test indicates improvement of virtual machine about 16%.

1. はじめに

近年，リソースの有効活用やレガシシステムの延命，クラウドコンピューティングへの適用などの用途において，仮想化技術は重要な技術要素である．特に Linux の領域では，Linux Kernel 自体が仮想マシンモニタの役割を担う KVM(Kernel-based Virtual Machine) が注目を集めている．

KVM におけるリソース管理は，Linux Kernel が持つプロセススケジューリングやメモリ管理などを活用する．そのため，仮想環境のリソース管理を KVM 自体に実装する必要がなく，仮想環境提供に特化した効率的な仮想化基盤の開発が可能となる．また，KVM が提供する仮想環境は，Linux Kernel において他プロセスと同等の操作で管理できるため，既存 OS と仮想環境との親和性は高くなる．

一方，仮想環境の管理方針や仮想環境のリソース調整なども Linux Kernel に強く依存する．このような状況では，仮想環境の特性を考慮したリソース管理を施しにくく，管

[1] 株式会社 日立製作所 IT サービス事業部
IT Management Services Division, Hitachi, Ltd.

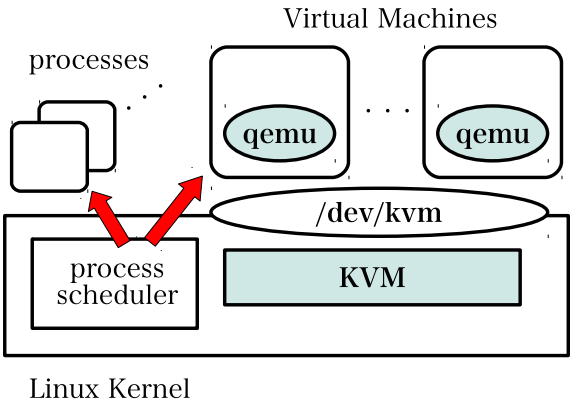

図 1 KVM の概要

理者が意図しない挙動になる可能性もある．また，システム規模によっては数百以上の仮想環境を運用する管理者にとって，各仮想環境のリソース管理を細かく調整することは運用負荷増加を招く．

この問題を解決するために，仮想環境の特性を考慮し Linux Kernel に対してリソース調整を働きかける，リソース調整機構を提案する．リソース調整機構には，管理方針をまとめた *policy* と，Linux Kernel にリソース調整を働きかける *peacemaker* を新規に開発した．*policy* は，Linux Kernel と独立した状態で管理者自身が作成可能であるため，管理者が意図する管理方針を仮想環境に適用可能である．また，*policy* をリソース調整機構に適用後，仮想環境間のリソース調整は *policy* と連携した *peacemaker* が実施するため，運用時の管理者負担を抑制できる．

加えて，*policy* はリソース調整機構間で流用することが可能であるため，類似する管理方針を持つシステム間で同じ *policy* を利用でき，仮想環境管理のナレッジとして共有，蓄積することも可能となる．

2. 背景

2.1 KVM の概要

KVM(Kernel-based Virtual Machine) とは，仮想マシンモニタの機能を Linux Kernel に付加するカーネルモジュール [1] のことである．KVM の概要を**図 1** に示す．

KVM は，プロセッサの仮想支援機能を使用し，ゲスト OS の修正が不要な完全仮想化を実現している．仮想環境の I/O エミュレーションは，各仮想環境に存在する QEMU と，KVM が提供するデバイスファイル/dev/kvm を経由し，カーネルモードとして I/O 処理を実現するよう Linux Kernel に処理を発行する．また，KVM が生成した仮想環境のプロセス管理は，一般的なプロセスと同様に Linux Kernel のプロセススケジューラが管理する．

2.2 Linux Kernel のプロセススケジューラの遷移

Linux Kernel におけるプロセススケジューリングは，以下に示す 3 つの方針を採用してきた．

- 線形探索 (Linux 2.4 系まで)
- O(1) スケジューラ (Linux2.6〜Linux2.6.22)
- Completely Fairness Scheduler (Linux2.6.23〜)

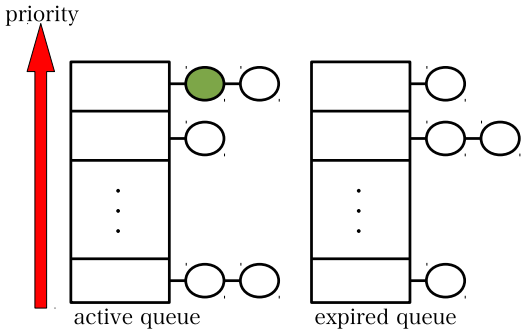

図 2 O(1) スケジューラの概要

Linux 2.4 系までのプロセススケジューラは，システム上に存在するプロセスを線形リストに登録し，最も優先度の高いプロセスをリストの先頭から末尾まで探索していた．線形探索によるプロセススケジューリングは，実装方法が単純という利点はあるが，システムに存在するプロセス数の増加に比例し，スケジューリングコストも増加する．そのため，プロセスを多く生成するエンタープライズ用途には適していなかった．

この問題を解決するため，Linux2.6 系ではシステム上に存在する最も優先度が高いプロセスを一意に決定できる O(1) スケジューラに変更した．O(1) スケジューラの概要を**図 2** に示す．

O(1) スケジューラには，タイムスライスを使い果たしていないプロセスを登録する active queue と，タイムスライスを使い果たしたプロセスを登録する expired queue が存在する．次に実行するプロセスは，active queue の最も優先度の高いプロセスを選択する．プロセスがタイムスライスを使い果たすと，O(1) スケジューラは該当プロセスの優先度を再計算し，再計算した優先度に対応する expired queue に接続する．すべてのプロセスが expired queue に遷移し active queue が空になると，active queue と expierd queue の役割を入れ替える．

O(1) スケジューラは，次に実行するプロセスを選択するスケジューリングコストを抑えることができる反面，スケジューリングアルゴリズムの複雑さや，プロセス優先度を計算するヒューリスティックに合致しないプロセスは，公平に CPU リソースを割り当てられない可能性があるなどの課題を抱えていた．

この課題を克服するため，Linux 2.6.23 以降のプロセススケジューラは，CFS(Completely Fair Scheduler) へ変更となった．**図 3** に CFS の概要を示す．

CFS では，システムに存在するプロセスを二分木の一種である red-black tree で管理する．この red-black tree は，

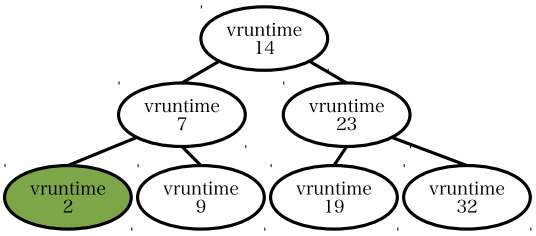

図 3 red-back tree の概要

CFS が独自に持つ仮想的な CPU 実行時間である vruntime をキーにプロセスを並べ替える．vruntime が常に最小となる red-black tree 左下に位置するプロセスに CPU を割当て，各プロセスが CPU リソースを均等に利用できるようスケジューリングする．

2.3 課題

既存 OS のリソース管理を仮想環境に適用する際の課題を以下に示す．

- 既存 OS に強く依存したリソース管理
- 仮想環境の特性を考慮した運用の難しさ

第一の課題として，既存 OS に強く依存したリソース管理を挙げることができる．既存 OS のリソース管理を活用し，仮想化基盤の開発効率化を図ることができるという利点はある．しかし，KVM のように Linux Kernel のプロセススケジューラの影響を直接受ける環境では，CFS が仮想環境と一般的なプロセスとを同等に扱うため，一般プロセスよりも仮想環境に対して優先的にリソースを割り当てなければならない状況で，仮想環境の特性を考慮した適切なリソース調整を実現できない可能性もある．

また，Linux Kernel のプロセススケジューラが変更になった場合，仮想環境の挙動に大きな影響を与えるケース [2] もある．その際，管理者主体でリソース管理を調整できないため，Linux Kernel が変更になった場合は，影響調査や仮想環境間の負荷調整を多数の仮想環境ごとに再実施する手間が発生するケースもある．

第二の課題として，システム上に存在する仮想環境の特性を考慮したリソース管理の難しさを挙げることができる．既存の仮想環境の運用管理を考えると，管理者が主体となり仮想環境管理ツールを用いて仮想環境を管理・監視する手法が一般的である．しかし，既存の仮想環境管理ツールでは，仮想環境生成時に割り当てるリソースを指定することはできるが，Linux Kernel のリソース管理方針に影響を与えるような設定変更はできない．また，数百以上に及ぶ仮想環境に対して，リソース利用状況の定期的な監視や，負荷状況に合わせたリソース調整を，管理者が各仮想環境に対して実施することは困難である．

これらの課題を解決するために，管理者が生成した簡易的な管理方針を KVM に組み込み，その後は KVM が Linux Kernel と協調し仮想環境のリソース管理を調整する，リソース調整機構を提案する．

3. 設計方針

Linux Kernel に依存したリソース管理の課題を解決するために，仮想環境の特性を考慮可能なリソース調整機構を試作した．今回試作したリソース調整機構の要件を，以下に挙げる．

- KVM が生成する仮想環境に特化したリソース調整が可能であること
- 管理者による定期的なリソース調整が必要ないこと
- リソース調整機構で定義したリソース調整方針は，他のリソース調整機構を備えたシステムでも流用可能であること
- Linux Kernel に対して原則修正を加えないこと

KVM が生成する仮想環境は，Linux Kernel では一般的なプロセスとして扱う．KVM が生成した仮想環境と一般的なプロセスすべてをリソース調整機構で管理対象とすると，リソース調整が必要な仮想環境の特定，リソースの利用状況取得，リソース割当て変更などの処理を実施する際，効率的な管理が実現できない可能性がある．そこで，今回試作するリソース調整機構では，KVM が生成した仮想環境に限定し，リソース調整を施す．

次に，仮想環境のリソース調整を実施する管理者側負担を軽減するために，仮想環境のリソース管理方針をまとめた *policy* という概念をリソース調整機構に導入する．管理者は，システム上に存在する仮想環境の管理方針を *policy* としてまとめ，KVM に適宜組み込む．リソース調整機構は *policy* に基づき，Linux Kernel と KVM にリソース調整を働きかける．*policy* を導入することによって，管理者は仮想環境に対し，定期的なリソース監視やリソース再配分を主体的に実施する必要がなくなり，運用負荷軽減につながる．

policy には，他システムで動作するリソース調整機構間で共通のインタフェースを提供する．そのため，一度あるシステムで作成した *policy* は，管理方針が類似する他システムへ容易に移植することが可能となり，*policy* を利用したシステム間での管理方針のナレッジ共有化を図ることもできる．

また，リソース調整機構を実装する際，Linux Kernel に対する修正は，原則施さないこととする．Linux Kernel の特定機能に依存した設計・実装をした場合，今後の開発方針によっては修正や削除対象になることも予想でき，リソース調整機構の保守性を維持することが困難になる．

なお，今回の試作で対象とするリソースは CPU に限定し，対象アーキテクチャは x86 のみとした．

4. リソース調整機構の実装

本節では，仮想環境の特性を考慮したリソース調整機構

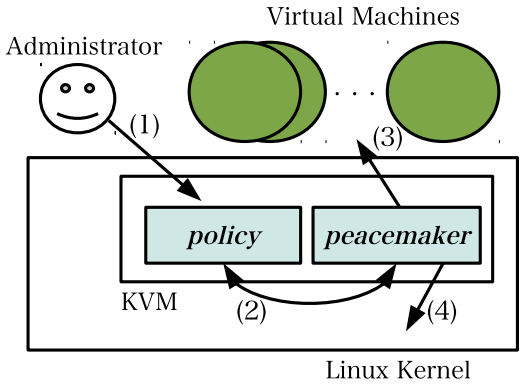

図 4 リソース調整機構の概要

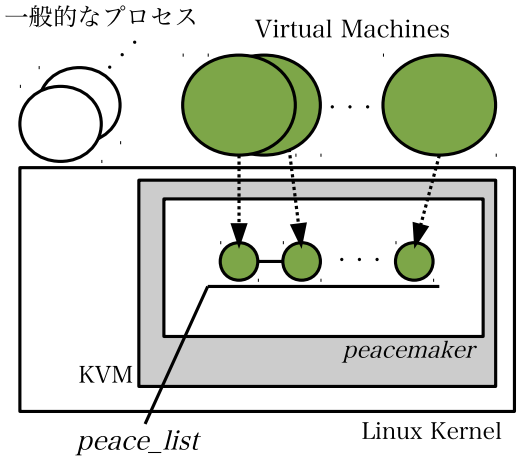

図 5 *peace_list* の概要

の実装手法について述べる．リソース調整機構は以下の機能を持ち，それぞれ KVM 内に配置する．

- *peacemaker*: 仮想環境や Linux Kernel からリソース情報を収集し，仮想環境間のリソースを調整する機能
- *policy*: 仮想環境間のリソース調整方針を保持し，*peacemaker* にリソース調整の指示を出す機能

リソース調整機構の概要を**図 4** に示す．システム管理者は，*peacemaker* が提供するリソース調整を実施する関数群を用いて，物理 CPU の負荷情報，各仮想環境のプロセス優先度，CPU 使用時間などの情報収集や，仮想環境間のリソース調整を指示する *policy* を作成する (図 4 (1))．

KVM が仮想環境を生成すると，*policy* は *peacemaker* と連動し，リソース情報の収集と各仮想環境へのリソース配分の方針を *peacemaker* へ指示する．*policy* からリソース調整の指示を受けた *peacemaker* は，Linux Kernel や仮想環境から CPU に関するリソース情報を収集し，その結果を *policy* へフィードバックする (図 4 (2))．

並行して，*policy* から受けたリソース調整の指示を，仮想環境や Linux Kernel に対して実行する (図 4 (3)，(4))．

peacemaker の具体的な実装方法は，4.1 節で述べ，*policy* の作成方法については，4.2 節に記載する．

4.1 *peacemaker* の実装

リソース調整機構において，リソース情報の収集とリソース調整の実施を司る *peacemaker* は，以下の機能を有する．

- 一般的なプロセスと KVM が生成した仮想環境の管理分離 (4.1.1 節)
- 仮想環境と Linux Kernel からリソース情報を収集 (4.1.2 節)
- 仮想環境に対してリソース調整を実施 (4.1.2 節)

以降の節では，この 3 つの機能について述べる．

4.1.1 一般的なプロセスと仮想環境の管理分離

Linux Kernel では，KVM が生成した仮想環境も一般的なプロセスと同様に扱う．そのため，既存の Linux Kernel や KVM の機能だけでは，一般的なプロセスと KVM が生成した仮想環境が区別できず，KVM が生成した仮想環境のみに対象を絞ったリソース調整が困難となる．

この課題を解決するため，*peacemaker* 内に KVM が生成した仮想環境のプロセス情報のみを対象とする管理リスト *peace_list* を準備した．*peace_list* の概要を**図 5** に示す．

peace_list は，仮想環境の情報をリストとして保持する．リストに存在する各要素の情報は，KVM が生成した仮想環境の task_struct 構造体へのポインタと，*peace_list* のリストをつなぐ list_head 型の kvms をメンバに持つ kvm_list 構造体を保持する．

peace_list への仮想環境情報登録は，KVM が仮想環境の生成処理を行う kvm_create_vm 関数実行時に *peacemaker* を呼び出し実施する．その際，引数として current マクロの情報を基に生成中の仮想環境のプロセス情報を *peacemaker* に渡す．*peacemaker* に処理が移ると，*peacemaker* は *peace_list* に，KVM が生成した仮想環境の task_struct 構造体が保持するプロセス情報をリストとして登録する．

peace_list からの仮想環境情報削除は，KVM が仮想環境の削除処理を行う kvm_destroy_vm 関数実行時に *peacemaker* を呼び出し実行する．呼び出しを受けた *peacemaker* は，削除対象となる仮想環境のプロセス ID を取得し，*peace_list* から該当する仮想環境のプロセス情報を削除する．

この手法では，KVM に修正を加える必要があるが，KVM への修正は 4 行の追記のみで実現できる．また，KVM へ追加する部分も，仮想環境の生成，削除を実行する関数であるため，これらの関数自体が LVM から削除になる可能性は極めて低い．追加するソースコード量や，KVM へのソースコード追加先の関数の機能を考慮すると，このレベルの KVM に対する追記であれば，リソース調整機構の保守性を十分確保できる．

このように，*peace_list* に仮想環境情報をまとめ，一般的なプロセスと分離して仮想環境を管理するため，リソース調整の対象となる仮想環境の特定が容易になる．また，

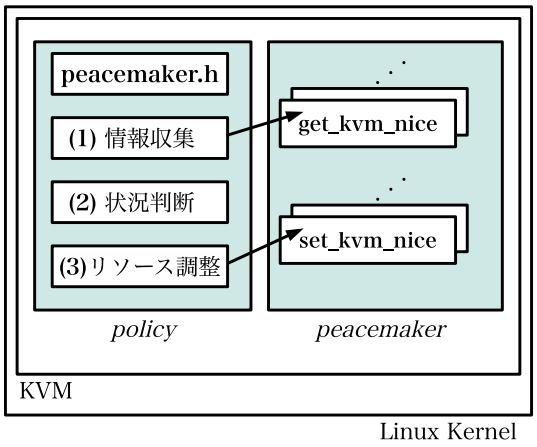

図 6 リソース調整関数の利用例

peacemaker に *peace_list* を準備するため，Linux Kernel のプロセススケジューラや task_struct 構造体の改変を伴わず，*peacemaker* 単独で各仮想環境の状況を把握できる．よって，Linux Kernel や KVM とリソース調整機構との独立性を保つことができる．

4.1.2 リソース調整関数の実装

Linux Kernel や KVM が生成した仮想環境からリソース調整に必要な情報の取得や，仮想環境に対するリソース調整を *policy* で指示するために，*peacemaker* 側にリソース調整関数を実装した．リソース調整関数の概要を**図 6** に示す．

peacemaker が *policy* に対して提供するリソース調整関数の実体は，リソース調整機構が使用するリソース情報収集やリソース調整関数を定義している peacemaker.c である．また，リソース調整関数を *policy* にインクルードして使用できるよう，リソース調整関数の宣言は peacemaker.h としてまとめる．

policy は，リソース調整に必要な CPU 利用率やプロセススケジューリングに影響を与えるパラメータなどの情報を収集するリソース調整関数を利用し，物理リソースや各仮想環境の状況を把握する (図 6 (1))．

仮想環境間でのリソース調整の実施は，収集した情報を基に *policy* が必要可否の状況判断を実施する (図 6 (2))．

policy がリソース調整を必要と判断した場合，*peacemaker* が提供するリソース調整関数を利用して，仮想環境間のリソースを調整する (図 6 (3))．

今回試作したリソース調整関数の一覧を**表 1** に示す．*peacemaker* が提供するリソース調整関数のうち，リソース調整を実施する関数には，関数名の先頭に set を使用することとした．また，リソース使用状況などの情報収集を担う関数は，関数名の先頭に get をつけた．これらのリソース調整関数は，Linux Kernel が保持するハードウェアリソース情報を保持する変数や，プロセス管理に使用する task_struct 構造体などの情報から，仮想環境間のリソース調整に必要な情報を抽出・生成し，*policy* に提供する．

peacemaker のリソース調整関数を用いた *policy* の作成方法は，4.2 節で説明する．

4.2 *policy* の作成方法

リソース調整関数を用いて仮想環境間のリソース調整を実施する *policy* には，以下に示す処理を準備する．

- *policy* の実行間隔を指定する処理
- 各仮想環境のリソース利用状況の取得し，リソース調整実行をする処理
- 仮想環境間のリソース調整可否を判断する処理

policy の例として，物理 CPU 利用率が 90%を超える場合，優先度の低い仮想環境の CPU 利用率を抑制し，物理 CPU 利用率が 10%未満の場合，優先度の高い仮想環境の CPU 利用率上昇を促す *policy* の例を，**図 7** に示す．

policy の実行間隔指定は，Linux Kernel が提供する schedule_timeout 関数を用いる．*policy* 内で実行間隔を指定できることにより，システム要件の変更や他システムへの *policy* の移植が必要になった場合などにも，柔軟にリソース調整の実行間隔を変更できる．

policy で各仮想環境のリソース利用状況の取得と，該当仮想環境に対してリソース調整を実行する機能は，*peacemaker* が提供するリソース調整関数を組み合わせて使用する．

仮想環境間のリソース調整可否は，リソース調整関数で取得した情報を基に，物理リソースや各仮想環境のリソース利用状況を基に，どの仮想環境にリソースを優先的に割り当てるかの条件を組み合わせ作成する．

このように，*policy* は Linux Kernel とは独立して作成することができ，一度 *policy* を作成すれば，類似のリソース調整が必要な他システムへは *policy* のみを移植すれば良い．また，システム上に存在する仮想環境の特性が，運用を進めていく上で変更になった場合でも，リソース調整可否を決定づける判断部分のみを変更すれば対応可能となる．また，一度 *policy* を作成すれば，仮想環境の運用管理を自動化でき，運用管理者の負荷低減も実現できる．

5. 実験

リソース調整機構の有効性を確認するために，以下に示す環境を準備した．

- 標準 KVM
- リソース調整機構を実装した KVM

各環境をそれぞれをひとつ準備し，演算処理，I/O 処理などの負荷を用いたベンチマークソフト UnixBench [3] を用いて実験を実施した．両環境に共通するスペックを**表 2** に示す．

リソース調整機構を組み込んだ KVM には，物理 CPU の過去 15 分間の利用率が 90%を超えた場合，システム上

表 1 *peacemaker* が提供するリソース調整関数一覧

No.	関数名	機能
1	void set_kvm_nice(struct task_struct *, long)	第一引数で指定した仮想環境に第二引数の nice 値を設定
2	int get_kvm_nice(const struct task_struct *)	第一引数で指定した仮想環境の nice 値を出力
3	int get_nr_kvms(void)	システム上に存在する仮想環境数を出力
4	void get_physical_cpus_loadavg(struct cpu_loadavg *)	1 分間，5 分間，15 分間の CPU 負荷情報を第一引数に保存
5	struct kvm_list *get_max_kvms_nice(void)	システム上で最も高い nice 値を持つ仮想環境を出力
6	struct kvm_list *get_min_kvms_nice(void)	システム上で最も低い nice 値を持つ仮想環境を出力
7	struct kvm_list *get_max_kvms_prio(void)	システム上で最も高い優先度を持つ仮想環境を出力
8	struct kvm_list *get_min_kvms_prio(void)	システム上で最も低い優先度を持つ仮想環境を出力
9	struct kvm_list *get_max_kvms_usage(void)	システム上で最も usage が高い仮想環境を出力
10	struct kvm_list *get_min_kvms_usage(void)	システム上で最も usage が低い仮想環境を出力
11	struct kvm_list *search_pid_from_peace_list(pid_t)	第一引数で指定したプロセス ID から該当仮想環境を探索し出力
12	void add_kvm_list(struct task_struct *)	第一引数で指定した仮想環境を管理対象に追加
13	void del_kvm_list(pid_t)	第一引数で指定したプロセス ID の仮想環境を管理対象から削除

```
# include ”peacemaker.h”

int peacemaker_policy_main(void *arg){
 利用する変数の初期化;
 schedule_timeout 関数による実行間隔指定;

 while (システム上に仮想環境が 1 つ以上存在){
  /* CPU 利用率を取得 */
  load_ave = get_physical_cpus_loadavg();
  if(load_ave > 90%){
   /* 最も優先度が低い仮想環境を選択 */
   target_vm = get_min_kvms_prio();
   /* カーネル関数 task_nice() から nice 値を取得 */
   nice = task_nice(target_vm);
   if(nice < 15)
     /* 優先度の低い仮想環境の CPU 割当を削減 */
      set_kvm_nice(target_vm, nice+1);
  }else if(load_ave < 10%){
   /* 最も優先度が高い仮想環境を選択 */
   target_vm = get_max_kvms_prio();
   nice = task_nice(target_vm);
   if(nice > -15)
     /* 優先度の高い仮想環境の CPU 割当を増加 */
      set_kvm_nice(target_vm, nice-1);
  }
 }
}
```

図 7 仮想環境の nice 値を調整する *policy* の抽象的な例

に存在する最もプロセス優先度が低い仮想環境の CPU 利用率を抑制し，物理 CPU の過去 15 分間の利用率が 10%未満の場合，システム上に存在する最もプロセス優先度の高い仮想環境に CPU リソースを多く割り当てる *policy* を準備した．実験結果を**表 3** に示す．

表 2 実験環境

項目	内訳
物理 CPU	Core 2 Duo @ 2.53GHz
物理メモリ	4GB
ホスト OS	Ubuntu 11.04 (Linux2.6.38)
KVM module	kvm-kmod-2.6.38-6
ゲスト OS	Fedora 15 (Linux2.6.38)
ゲストメモリ	2GB
ベンチマークソフト	UnixBench 5.1.3

全体としての性能指標を示す System Benchmarks Index Score の数値は，標準 KVM と比較してリソース調整機構を組み込んだ KVM は，約 16%の性能向上を果たした．このことから，リソース調整機構の *policy* や *peacemaker* が効果的に機能し，仮想環境性能の向上を図ることが可能と言える．また，数値演算，I/O 処理系の評価項目も，標準 KVM と比較してリソース調整機構を組み込んだ KVM では約 8%から約 40%と高い性能向上を示した．

これらの結果から，各プロセスに対し均等に CPU リソースを割り当てる CFS を使用した Linux Kernel においても，CPU の利用状況によって仮想環境に提供する CPU リソースを調整できることが確認できた．また，大幅な性能劣化は確認できなかったため，KVM にリソース調整機構を組み込んだ場合でも，オーバーヘッドの影響は少ないと判断できる．

6. 関連研究

Linux Kernel におけるリソース割当て改善方法として，Linux Kernel のプロセススケジューラを改変し，利用環境

表 3 実験結果

評価項目	標準の KVM	リソース調整機構を組み込んだ KVM	性能向上率
Dhrystone 2 using register variables	1004.2	1085.7	8 %
Double-Precision Whetstone	334.5	376.2	12 %
Execl Throughput	52.5	55.1	5 %
File Copy 1024 bufsize 2000 maxblocks	76.1	97.0	27 %
File Copy 256 bufsize 500 maxblocks	41.3	52.7	28 %
File Copy 4096 bufsize 8000 maxblocks	142.3	192.1	35 %
Pipe Throughput	20.0	28.0	40 %
Pipe-based Context Switching	13.5	17.3	28 %
Process Creation	13.3	13.8	4 %
Shell Scripts (1 concurrent)	34.6	39.5	14 %
Shell Scripts (8 concurrent)	36.7	35.1	-4 %
System Call Overhead	112.7	113.3	1 %
System Benchmarks Index Score	63.2	73.1	16 %

に最適化したスケジューリングポリシを適用する手法 [4–7] がある．プロセススケジューラを改変する手法では，KVM を利用する環境やマルチスレッド環境などの特定用途に適したリソース割当ての改善を見込むことができる．しかし，利用用途の変更や Linux Kernel 自体のプロセススケジューラが大幅に変更となった場合，Linux Kernel に対してその都度改変を施さなければならない．

Linux Kernel のプロセススケジューラの挙動を変更する特殊ファイルをあらかじめ準備し，システムの利用状況やユーザの希望に合わせて適宜変更する手法 [8] がある．この手法では，システム稼働中に Linux Kernel の改変を伴わず，プロセススケジューラの動作を変更することが可能である．しかし，変更可能なパラメータがプロセスのタイムスライス調整やプロセスマイグレーションに限られている点，仮想環境に特化したリソース調整が不可能な点，ユーザがシステムの稼働状況を定期的に監視し，システムの状況に合わせて特殊ファイルを操作しなければならない点など課題が多い．

Linux におけるリソース管理手法の一つとして，Control Groups(Cgroups) [9] がある．Cgroups は，CPU 時間，メモリ，ネットワーク帯域などのリソースを管理するポリシ */cgroup* 以下に階層として準備する．新規に実行するプロセスは，*/cgroup* 以下の最適な管理階層を指定し実行する．Cgroups では，プロセス単位で特定のリソース調整を実施することができるが，システム全体の負荷状況やプロセス単位の利用状況を定期的に監視する際には適さない．

その他の仮想環境管理ツールとして，OpenStack [10] がある．OpenStack は，仮想環境の起動，終了やリソース情報を取得することは可能である．しかし，仮想環境のリソース調整は，OpenStack から実施することはできない．

仮想化機構で各仮想環境のリソース調整を実施する手法として，仮想環境ごとのリソース利用状況を定期的に取得し，仮想環境の特性を仮想マシンモニタが持つスケジューラにフィードバックするシステム [11,12] がある．仮想環境の特性を考慮したリソース管理が実現可能となる一方，仮想マシンモニタ自体に大幅な修正が必要となる．今回提案したリソース調整機構では，Linux Kernel 本体に修正を加える必要はなく，今後 Linux Kernel のプロセススケジューラや KVM の実装が変更になった場合でも，大幅な修正を必要としない．また，リソース調整機構の構成をリソース割り当て方針としてまとめた *policy* と実質的なリソース調整を施す *peacemaker* とに分離している．そのため，今後リソース調整関数の追加実装が必要になった場合，*peacemaker* のみを拡張すれば柔軟に対応可能である．また，既存の *policy* は流用可能であり，リソース調整機構があれば，*policy* は他システムでも流用が用意に可能である．

7. おわりに

仮想環境の特性を考慮したリソース調整機構の提案と試作について述べた．今回提案したリソース調整機構は，仮想環境の管理方針をまとめた *policy* と，リソース調整関数を用いて仮想環境間のリソース調整を実施する *peacemaker* の機能を試作した．また，Linux Kernel 本体に修正を加える必要がないため，汎用性や保守性が高い機構となった．CPU の利用率に応じた CPU リソースの調整を実施する *policy* で実験したところ，通常の仮想環境と比較してリソース調整機構を用いた仮想環境では，約 16%の性能向上を図ることができた．

今後は，ひとつのシステムに複数の仮想環境が存在した場合の追加実験や，CPU 以外のリソースを対象に含めた，システム稼働中のシステムへの *policy* 適用方法を検討予定である．

参考文献

[1] Kivity, A., Kamay, Y., Laor, D., Lublin, U. and Liguori, A.: kvm: the Linux Virtual Machine Monitor, *OLS '07: the 2007 Ottawa Linux Symposium*, Vol. 1, Ottawa, Ontario, Canada, pp. 225–230 (2007).

[2] 東賢一朗：プロセススケジューラが仮想マシンへ与える影響に関する考察，情報処理学会 第 73 回全国大会 (2011 年).

[3] UnixBench: http://code.google.com/p/byte-unixbench/.

[4] Jiang, W., Zhou, Y., Cui, Y., Feng, W., Chen, Y., Shi, Y. and Wu, Q.: CFS Optimizations to KVM Threads on Multi-Core Environment, *Proceedings of the 2009 15th International Conference on Parallel and Distributed Systems*, ICPADS '09, Washington, DC, USA, IEEE Computer Society, pp. 348–354 (online), DOI: http://dx.doi.org/10.1109/ICPADS.2009.83 (2009).

[5] Tao, D., Qin-fen, H., Bing, Z., Tie-gang, Z. and Li-ting, H.: Scheduling Policy Optimization in Kernel-based Virtual Machine, *2010 International Conference on Computational Intelligence and Software Engineering(CiSE 2010)* (IEEE, ed.), pp. 1–4 (2010).

[6] Wong, C. S., Tan, I., Kumari, R. D. and Wey, F.: Towards Achieving Fairness in the Linux Scheduler, *SIGOPS Oper. Syst. Rev.*, Vol. 42, No. 5, pp. 34–43 (online), DOI: http://doi.acm.org/10.1145/1400097.1400102 (2008).

[7] Wong, C., Tan, I., R.D.Kumari, Lam, J. and Fun, W.: Fairness and Interactive Performance of O(1) and CFS Linux Kernel Schedulers, *Information Technology, 2008. ITSim 2008.*, Vol. 4 (2008).

[8] Kobus, J. and Szklarski, R.: Completely Fair Scheduler and its tuning, Technical report, Nicolaus Copernicus University (2009).

[9] Prpič, M., Landmann, R. and Silas, D.: *Red Hat Enterprise Linux 6 Resource Management Guide Manageing system resources on Red Hat Enterprise Linux 6*, 1801 Varsity Drive Raleigh, NC 27606-2072 USA, 1.0-3 edition (2010).

[10] OpenStack: http://www.openstack.org/.

[11] Kim, D., Kim, H., Jeon, M., Seo, E. and Lee, J.: Guest-Aware Priority-Based Virtual Machine Scheduling for Highly Consolidated Server, *Proceedings of the 14th international Euro-Par conference on Parallel Processing*, Euro-Par '08, Berlin, Heidelberg, Springer-Verlag, pp. 285–294 (2008).

[12] Kim, H., Lim, H., Jeong, J., Jo, H. and Lee, J.: Task-aware Virtual Machine Scheduling for I/O Performance., *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '09, New York, NY, USA, ACM, pp. 101–110 (2009).