Linuxカーネル資源管理機能の動向

2005年11月11日

富士通株式会社
サーバシステム事業本部
Linuxソフトウエア開発統括部
前田　直昭

概要

FUJITSU

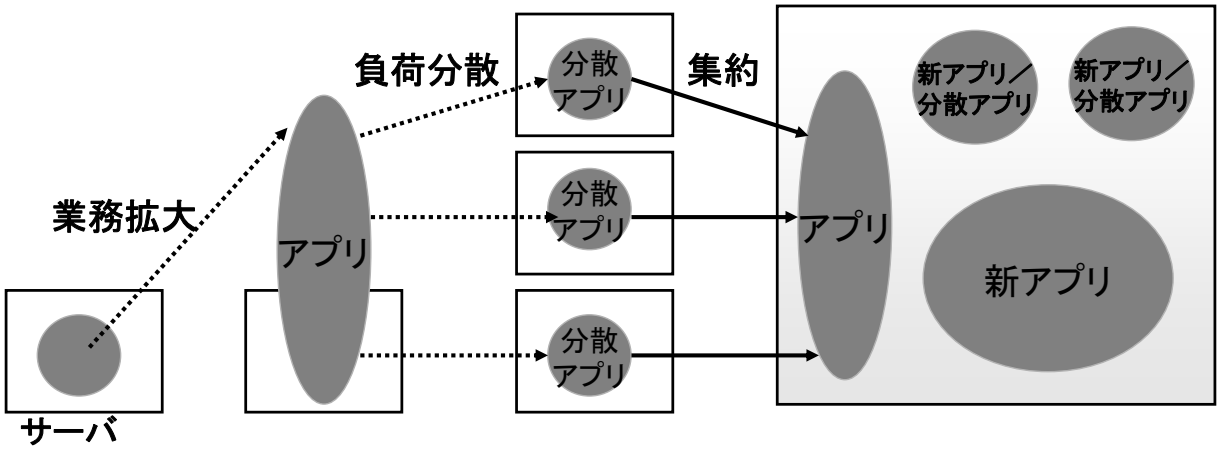
- 資源管理が必要とされる背景
- 資源管理フレームワークの動向
- CPUSETS
- CKRM (Class-based Kernel Resource Management)
- CPUリソースコントローラ
- メモリリソースコントローラ
- CKRMの使用例

資源管理が必要とされる背景

資源管理が必要とされる背景

FUJITSU

- 今後のサーバのトレンドはマルチコアと６４ビット
- サーバ能力に余裕が生じ、負荷分散のために分散させたサーバが再び集約
- 余った能力をつかって、さらに新しいアプリケーションを同一サーバ上に実装



サーバに複数のアプリを詰め込むと問題も

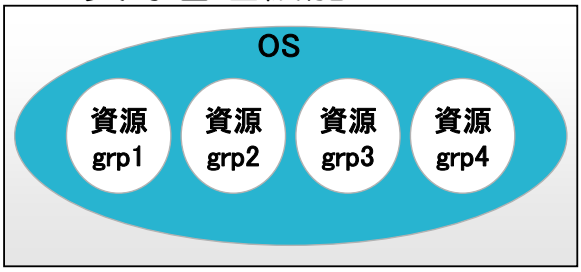
- ハード故障の波及範囲の拡大
- ソフトトラブルの波及範囲の拡大
- セキュリティー
- 計算機資源の干渉（取り合い）による性能不安定化

- 隔壁を設けて相互干渉を防ぐ仕掛けが必要

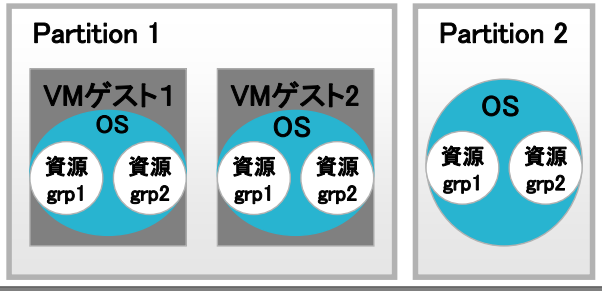
どこに隔壁を設けるか？

■OSの中に隔壁を設ける
１システム運用

- 資源管理機能

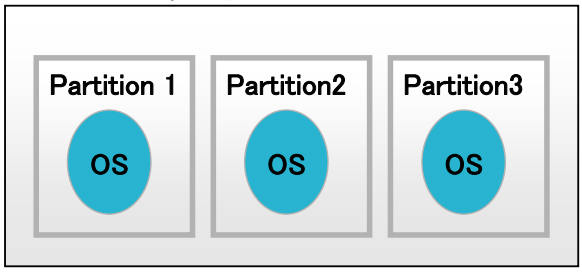


■組み合わせ

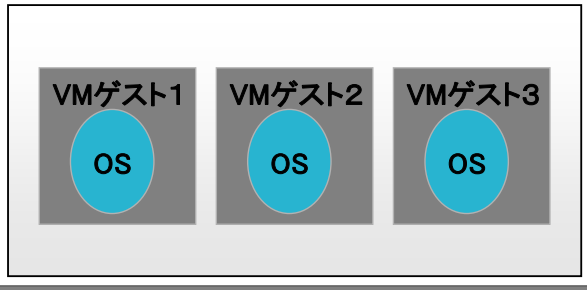


■OSの外に隔壁を設ける
複数システム運用

- パーティション



- 仮想計算機



各方式における隔壁の厚さ

FUJITSU

		ハード故障	ソフトトラブルセキュリティー	分割の柔軟性
OSの外に隔壁を作る	パーティション	○	○	×
	仮想計算機	×	○	○
OSの中に隔壁を作る	資源管理	×	×注	◎

注）ソフトトラブルやセキュリティーについては、資源乱用起因の波及は防止

どうやって選択するか？

FUJITSU

■必要な隔壁の厚さ
- ハード故障の影響
  →パーティション
- ソフトトラブルやセキュリティー
  →仮想計算機
- 計算機資源の干渉
  →資源管理

■アプリの関連性
- アプリが資源を共有するなど、密接に関連している（ファイル共用、アプリ間通信）
  →資源管理
- アプリの関連性が無い
  →仮想計算機・パーティション

本公演では資源管理に的を絞って説明

■運用コスト・柔軟性
- システムの数を減らすことによる、運用管理コストの低減
- 動的な資源変更の柔軟性
  →資源管理
  （１システム運用・分割の柔軟性）

■ラップトップ・エントリ機
- 能力の低い計算機で分割運用したい
  →資源管理
  （低オーバヘッド）

■仮想計算機固有のメリット
- 実行中環境のサスペンド・リジューム
- ハードウエアのマイグレーション

資源管理フレームワークの動向

資源管理のフレームワーク

FUJITSU

- 資源を分割するための資源グループ定義
  - 階層型、非階層型
- 資源グループに計算機資源を割り当て
  - 制御できる資源の種類と分割単位
- プロセスが特定の資源グループを使うように関連づけ
  - PID、UID etc.
- 資源グループ内の資源使用状況のモニタ

資源管理フレームワークの動向

FUJITSU

- 2.6カーネルでは、資源管理フレームワーク機能を持つ、２つの似たような実装が存在
  - CPUSETS
    - NUMAマシンにおける最適資源割り当てを目的に設計されており、資源を積極的に分割はしないが、粗い分割単位で資源管理を行うことができる
    - 2.6標準カーネルに取り込み済み
  - CKRM
    - 資源管理を目的に設計されており、細かい分割単位で資源管理を行うことができる
    - まだ、標準カーネルに取り込まれていない

FUJITSU THE POSSIBILITIES ARE INFINITE

CPUSETS

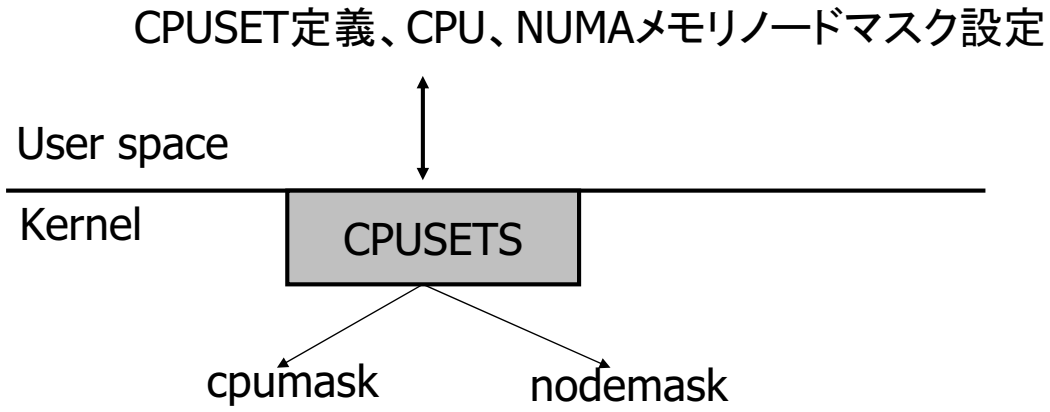
CPUSETS

FUJITSU

- 階層型資源グループ定義(CPUSET)
- 資源としてCPUとメモリを制御できるが、分割単位は物理資源単位と粗い（CPU, NUMAノード）
- PIDによる資源グループとの関連づけと、子供への継承
- 資源グループ内の資源使用状況を表示する機能はない
- BULLとSGIが中心となって開発が進められ2.6.12で標準カーネルに取り込まれた

CPUSETSの構成

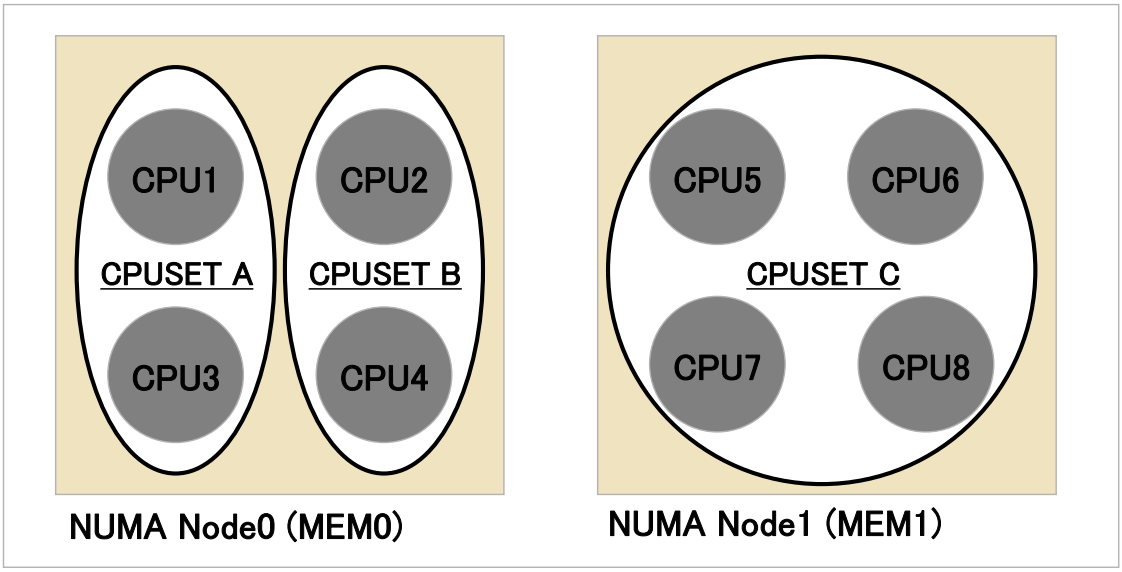
FUJITSU



プロセスのcpumask/nodemask を設定することで、プロセスが使用する CPU, メモリを制限

CPUSETSによる資源分割

FUJITSU



CPUSETの定義

CPUSET A： cpus=1,3 mems=0
CPUSET B： cpus=2,4 mems=0
CPUSET C： cpus=5-8 mems=1

資源管理として見た場合のCPUSETSの問題点

FUJITSU

- 資源の分割単位が粗い
  - 元々、NUMAマシンにおいて性能に最適化した、CPU・メモリ割り当てを行う利用方法を想定
  - 資源に壁を作るのが目的ではなく、ハード的な壁が存在する資源を効率良く、占有的に割り当てることが目的
  - 1CPUマシンではCPUを分割不可、非NUMAマシンではメモリを分割不可

CPUSETSを資源管理として利用できるケース

- １システム中のCPU数は増加傾向にあり、メモリの干渉が問題にならない（運用で回避できる）ならば、CPUSETSも有効
- X86_64アーキ限定だが、カーネルのブートオプションでメモリノードの分割数を指定できる。これを利用すればCPUSETSから、分割されたメモリを制御可能

- それでも、資源管理としては制約が大きいことに変わりはない

FUJITSU THE POSSIBILITIES ARE INFINITE

CKRM (Class-based Kernel Resource Management)

CKRM (Class-based Kernel Resource Management)

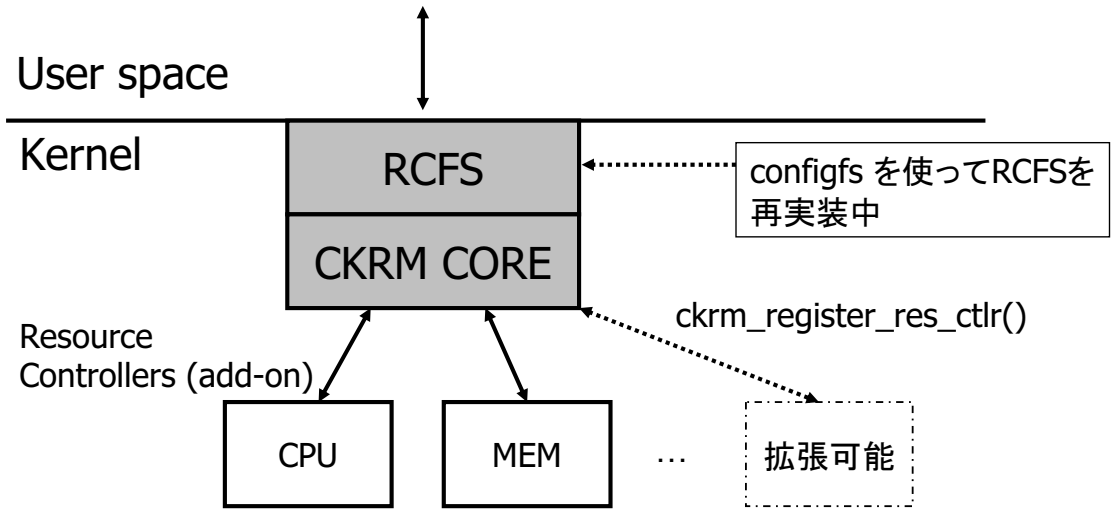
FUJITSU

- 階層型資源グループ（クラス）定義
- 資源の制御はリソースコントローラが実施
- PIDによる資源グループとの関連づけと、子供への継承
- 資源グループ内の資源使用状況の表示
- 2003年頃より、IBMが中心となり、オープンソースプロジェクトとして開発

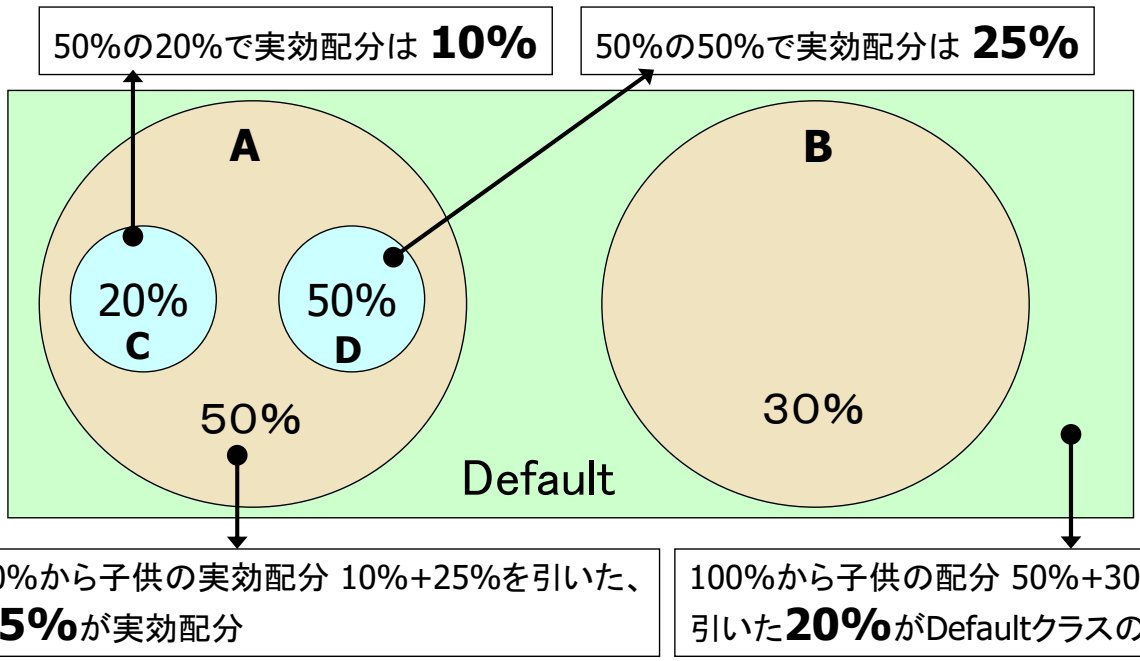
CKRMの構成

FUJITSU

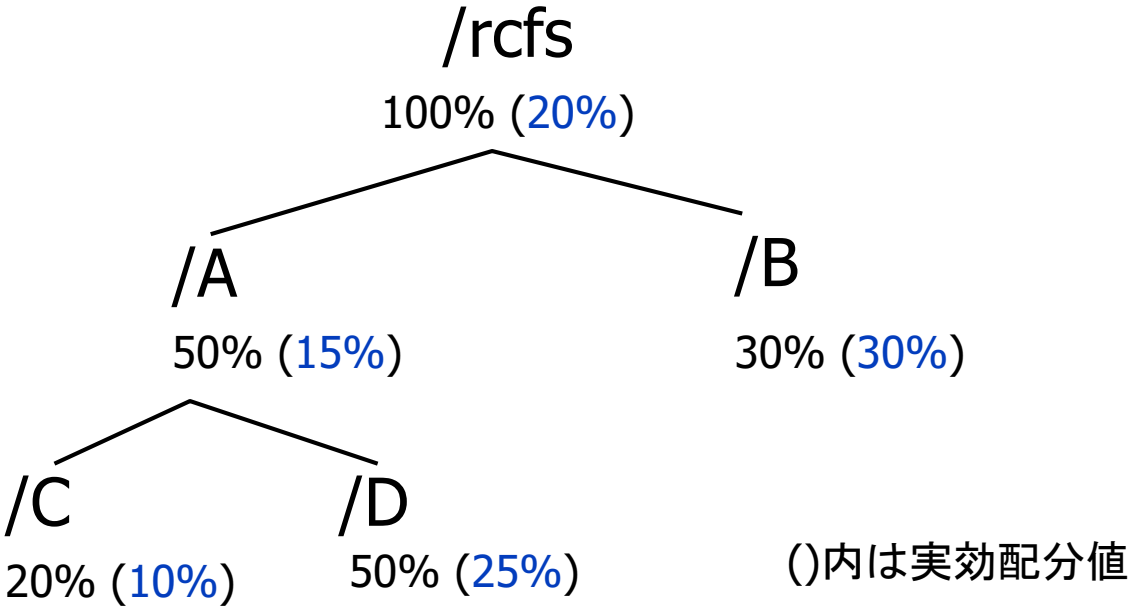
クラス定義、配分比設定など、CKRMの設定と資源使用状況のモニタ



階層型資源クラス定義とは？



RCFSで表現すると



階層型クラス定義のメリット

FUJITSU

- 実際の運用形態にマッチしている
  - 資源をおおまかに分割して配分した上で、その中をさらに細分化
  - 例えば、Ｗｅｂフロントとアプリサーバで大きく分け、さらにアプリサーバの中をサービス毎に分割
- 資源配分を変更すると、自動的に配下のクラスの実効配分が再計算されるので配分変更が簡単

CKRMの配分比設定

FUJITSU

- guarantee (最低保障）
  - 資源の最低保障量を配分比で指定
- limit （制限）
  - 資源の上限量を配分比で指定
- Total 配分比
  - デフォルト値は 100%だが, 任意の値を設定することも可能
  - 例えば、メモリの場合、ここに総量を指定しておくと、配分を絶対値で指定できる

CKRMの開発状況

FUJITSU

- –mm (Andrew Mortonの管理する機能追加のレビューツリー）に取り込まれたが、コミュニティーのレビューにより、いくつか問題点が指摘され、ツリーから落とされた
  - 2.6.13-rc3-mm1 に取り込まれる (7/15)
  - 2.6.13-rc4-mm1 で落とされる　(7/31)
- コミュニティーからのフィードバック
  - コード規模が巨大で複雑、メンテナンスコストが高い
  - 既存のLinuxカーネルの部品を上手く利用していない
  - コードレビューに耐えうる、実用的なリソースコントローラの実装が無い
- CKRMのフレームワークは評価された

CKRMの開発状況

FUJITSU

- 機能を必要最小限に絞りこみ、スクラッチから再実装
  - 旧(E版)CKRM　14925行（パッチの行数）
  - 新(F版)CKRM　　2803行（configfs版2060行）
  - 富士通は主にデザイン／コードレビューやデバッグ作業に貢献
- CPUリソースコントローラを実装
  - 富士通と VA Linux JapanでLinuxコミュニティーと共に開発中
  - メモリリソースコントローラについても同様

F版CKRMのリリース状況

- F0　版　　9/15
- F0.1版　　9/20
- F0.2版　　9/29
- F0.3版　　10/12
  - この版より CPUリソースコントローラと結合して動作するようになる
  - 安定したら、LKMLへポストし、mmツリーへの復活を図る

FUJITSU THE POSSIBILITIES ARE INFINITE

CPUリソースコントローラ

CPUリソースコントローラの仕事

FUJITSU

- CPU単位での資源分割ではなく、より細かな配分比で分割する
  - CPU単位の分割は、CPUSETSで可能
- 配分比を使い切れなければ、余りは他の資源クラスにまわす
  - 食べきれない食べ物は、他におなかのすいている人々に分け与えられる
  - おなかがすいていれば、配分比までは食べることができる。また、全体で食事が余っていれば、さらにそれ以上に食べることができる

CPUコントローラの開発状況

FUJITSU

- 当初開発されていた CPUコントローラはカーネルへの取り込み活動で頓挫していた
  - ２レベルスケジューリングのため、スケジューリングオーバヘッドが大きい
  - ランキューの構造を変更したり、既存のO(1)スケジューラに大きな手が入る（約 4,900行）
- 6中頃より、新しいCPUコントローラの検討を開始
  - 既存スケジューラの構造にあまり手が入らない、コンパクトな実装（現在、約 500行）
  - これを F版 CKRM と結合

O(1)スケジューラと配分比制御は、あまり相性が良くない

FUJITSU

- O(1)スケジューラの特長
  - 実行可能プロセスの数に依存せず、次に動かすプロセスを一定時間で選択できる
- CPU配分制御 (CVT/BVT)
  - プロセスが消費したCPU時間を累積し、配分比に従うように、次に動かすもっとも好ましいプロセスを探す
    ＝＞　下手をすると 2.4カーネルの１本キュー方式に先祖帰り

O(1)スケジューラと配分比制御は、あまり相性が良くない(続き)

FUJITSU

- ２レベルスケジューリング
  - 1stレベル: 配分比に従うように次に動かすクラスを選択
  - 2ndレベル:選択されたクラスの中でO(1)スケジューラにより、次に動かすプロセスを選択
- 既存のO(1)スケジューラに大きく手を入れずに、かつ、あまりオーバヘッドを増やさずに配分比制御を行う方法を模索

新CPUコントローラの設計

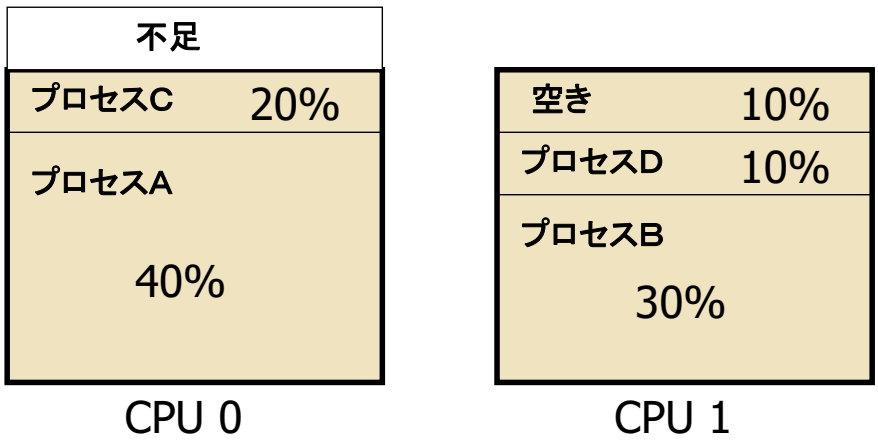
FUJITSU

- CPU使用量の統計を取り、それと配分比を比較してタイムスライスを調整することで配分比制御を実現
  - nice値を調整すると応答性に影響してしまう
- 統計情報を採取する所と、タイムスライスを調整する部分以外、スケジューラには手が入らない
- UPシステムでは、うまく動いているが、SMPシステム固有の課題がある

SMPにおけるCPU配分制御の課題 (Case 1)

FUJITSU

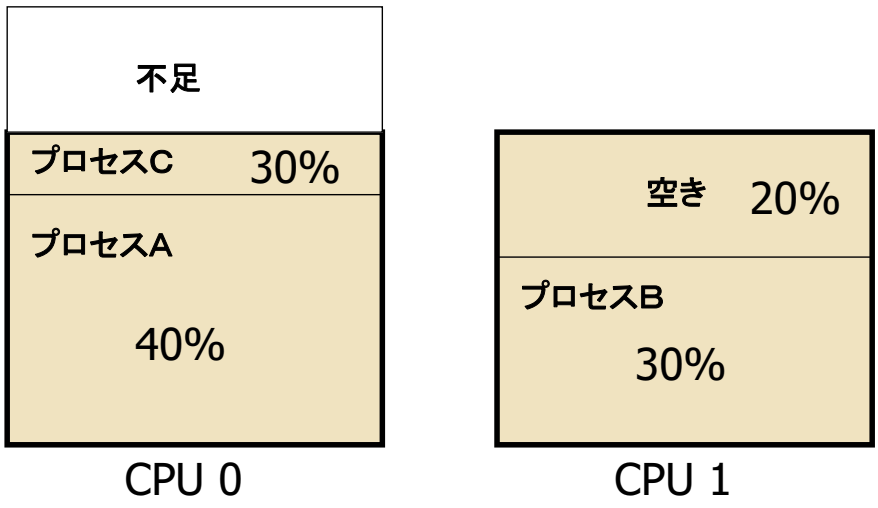
- SMPの場合、プロセスのCPU割り付けに依存して、指定した配分比にならない場合がある



SMPにおけるCPU配分制御の課題 (Case 2)

FUJITSU

- こんな意地悪な場合も



どうやって解決するか？

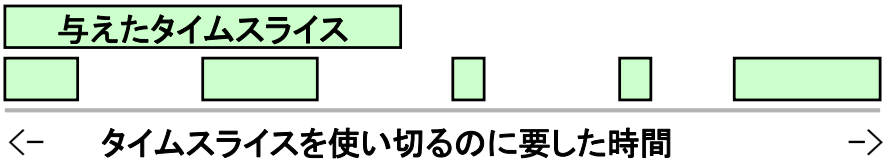
FUJITSU

- プロセスをどのCPUで動かすかは、以下のタイミングできまる
  - execシステムコール発行時に、負荷の少ない CPUに移動
  - 定期的にロードバランサが動作し、CPU負荷が平滑化するようにプロセスを移動する
- ロードバランサに手を入れる対処を検討中
  - exec時では、プロセスがCPUを消費する傾向にあるのか予測不能
  - 配分比を守ることを優先すると、プロセスマイグレーションが多発して、CPUキャッシュミスが増加し、全体のスループットが落ちるのでトレードオフがある

CPUコントローラの主要な関数

FUJITSU

- cpu_rc_account()
  - プロセスがタイムスライスを使い切った時に呼び出し、CPU使用率(load) を計算
  - load = 100 ×（前回与えた Timeslice ÷ Timeslice を使い切るのに要した時間）



- cpu_rc_collect_hunger()
  - プロセスにスイッチする時に呼び出し、待ち時間を計算して、配分比分のCPUが使えていない場合、空腹フラグを立てる

CPUコントローラの主要な関数（続き）

FUJITSU

- cpu_rc_recalc_tsfactor()
  - プロセスのタイムスライスを計算する時に呼び出し、空腹フラグや loadの状況に基づき、タイムスライスの倍率を決定する
  - 新倍率 = 今までの倍率 ×（配分比 ÷ load）
  - 倍率はタイムスライスを割り引く方向に働く（初期値１）

- 自クラスの空腹フラグが立っている
  - 自クラスの倍率を少しだけ上げる
- 自分以外、どれかのクラスの空腹フラグが立っている
  - 自クラスの倍率を配分比÷loadの比だけ下げる
- どのクラスの空腹フラグも立ってない
  - 自クラスの倍率をすこしずつ１に戻していく（みんな満足しているので、タイムスライスを割り引く必要が無い）

メモリリソースコントローラ

メモリリソースコントローラの仕事

- NUMA Node 単位での資源分割では無く、より細かな単位で物理メモリを分割する
  - NUMA Node 単位の分割は CPUSETSで可能
- 資源を使いきれなくても、余りを他の資源クラスにまわさない
  - CPUと違って、メモリにはデータが乗り、それをページアウト（I/O完了待ち）しないと再利用できないので、
    クラス間の移動にとても時間がかかる
    （CPUはレジスタをメモリに退避するだけでOK）
  - ただし、これはクラス内でメモリを再利用する場合にも当てはまる

メモリコントローラの開発状況

FUJITSU

- 当初開発されていたメモリコントローラが F版CKRMにも移植されているが、色々と改善の余地がある
  - 管理方法が複雑でオーバヘッドが大きい
  - 制限値を超えると、メモリアロケーションを失敗させているため、プロセスの動作が変わってしまう
  - バグが多く、安定していない
  - 実現機能の割に、コードが複雑
  - 過去、何度か linux-mm メーリングリストにポストされているが、ほとんどコメントがついていない
- 機能的な不足がある
  - ファイルアクセス性能の安定化には、ページキャッシュ量の制御が有効だが、ページキャッシュとプロセスメモリを個別に制御できない

メモリコントローラの開発状況（続き）

FUJITSU

- CPUコントローラのように、局所的な対処ではうまくいかない
  - 柔軟な制御を行うには、カーネルのメモリ管理を改善していく必要がある
  - メモリ管理は複雑でシビアなコンポーネントであり、インクリメンタルな開発で合意を取っていくことが成功の鍵
- 現在のアクティビティー
  - ページキャッシュとプロセスメモリを別々に扱うことを見越して、先ずは mapped/unmapped ページで、LRUリストを分割するパッチを提案中
  - これが受け入れられれば、ページキャッシュ制御の土台ができあがる
  - 平行して、現状のCKRMメモリコントローラとは、別の手法でメモリの隔壁をつくる実装を開発中

CKRMの使用例

F版 CKRMの使用例 (1/9)

- SourceForgeより最新のF版パッチ一式をダウンロード
  - http://ckrm.sourceforge.net/
- パッチを適用し、以下のCONFIGを設定してカーネルを作成
  - CONFIG_CKRM=y
  - CONFIG_RCFS_FS=m　（yでも良い）
  - CONFIG_CKRM_RES_CPU=y
  - CONFIG_CPU_RC=y

F版 CKRMの使用例 (2/9)

FUJITSU

- 作成したカーネルでリブートし rcfsをマウント
  - mkdir /rcfs
  - mount –t rcfs none /rcfs
  - ls /rcfs

    attrib config members shares stats
- クラス / (デフォルトクラス) の実効CPU配分を確認
  - cat /rcfs/stats | grep cpu

    cpu:effective_guarantee=*100*, load=0

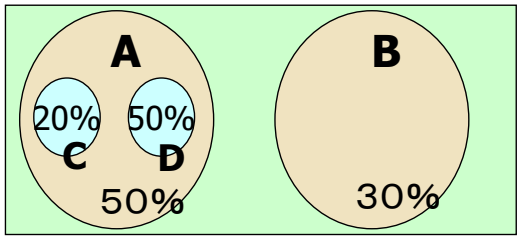
    (はじめは、デフォルトクラスが、全配分 100%を握っている)

F版 CKRMの使用例 (3/9)

FUJITSU

- クラス /A, /Bを作成
  - cd /rcfs
  - mkdir A B
  - ls

    A attrib B config members shares stats



- クラス /A, /Bの CPU配分を設定
  - cd /rcfs/A
  - echo “res=cpu,guarantee=50” > shares
  - cd /rcfs/B
  - echo “res=cpu,guarantee=30” > shares

F版 CKRMの使用例 (4/9)

FUJITSU

- 再度、クラス / (デフォルトクラス）の実効CPU配分を確認
  - cat /rcfs/stats | grep cpu

    cpu:effective_guarantee=*20*, load=0

    （子供 /A, /Bに 50%, 30%を与えたので、自分の実効配分は 20%になった）
- クラス /A, /Bの実効配分を確認
  - cat /rcfs/A/stats | grep cpu

    cpu:effective_guarantee=*50*, load=0
  - cat /rcfs/B/stats | grep cpu

    cpu:effective_guarantee=*30*, load=0

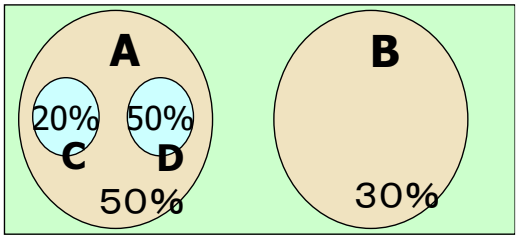
F版 CKRMの使用例 (5/9)

FUJITSU

- クラス /Aの下に、クラス /A/C, /A/Dを作成
  - cd /rcfs/A
  - ls

    attrib members shares stats
  - mkdir C D
  - ls

    attrib C D members shares stats



- クラス /A/C, /A/Dの CPU配分を設定
  - cd /rcfs/A/C
  - echo “res=cpu,guarantee=20” > shares
  - cd /rcfs/A/D
  - echo “res=cpu,guarantee=50” > shares

F版 CKRMの使用例 (6/9)

FUJITSU

- クラス /A/C, /A/D の実効配分を確認
  - cat /rcfs/A/C/stats | grep cpu
    cpu:effective_guarantee=*10*, load=0
  - cat /rcfs/A/D/stats | grep cpu
    cpu:effective_guarantee=*25*, load=0
- クラス /A, /B の実効配分を再確認
  - cat /rcfs/A/stats | grep cpu
    cpu:effective_guarantee=*15*, load=0
    （子供の /A/C, /A/D に 10%, 25%を与えたので、自分は残りの１５%になった）
  - cat /rcfs/B/stats | grep cpu
    cpu:effective_guarantee=*30*, load=0
    （こちらは子供がいないので、30%のままで変わらない）

F版 CKRMの使用例 (7/9)

FUJITSU

- ここでクラス /A の配分を 50%から40%に変更してみる
  - cd /rcfs/A
  - echo “res=cpu,guarantee=40” > shares
- そうすると、子供の /A/C, /A/D の実効配分もそれぞれ２割ダウンになる
  - cat /rcfs/A/C/stats | grep cpu
    cpu:effective_guarantee=*8*, load=0
  - cat /rcfs/A/D/stats | grep cpu
    cpu:effective_guarantee=*20*, load=0

F版 CKRMの使用例 (8/9)

FUJITSU

- クラスにプロセスを投入するには members ファイルに pid を書き込む
  - echo 1234 > /rcfs/A/C/members
  - cat /rcfs/A/C/members
    1234
- そのプロセス配下の子プロセスは自動的に同じクラスに入る
  - cat /rcfs/A/C/members
    1234
    1235
    ...
- クラス内のプロセスが動き出すと、CPU使用率の統計情報が statsファイルの load に表示される
  - cat /rcfs/A/C/stats | grep cpu
    cpu:effective_guarantee=8, load=*30*

F版 CKRMの使用例 (9/9)

FUJITSU

- これで、基本的な使い方を一通り説明したので、是非、ご自分でダウンロードして色々と試してみてください。
- バグレポート、コメント等は下記MLへ
  - ckrm-tech@lists.sourceforge.net
  - MLのアーカイブも sourceforge のサイトで公開されています
- 開発への参画もお待ちしております

商標

※ Linuxは、米国およびその他の国におけるLinus Torvalds氏の登録商標または商標です。

※ 本資料中の社名、商品名はすべて各社の商標または登録商標です。

※ 本資料に掲載されているシステム名、製品名などには、必ずしも商標表示（TM, R）を付記しておりません。

FUJITSU

THE POSSIBILITIES ARE INFINITE