hbstudy#19

Control Groups(cgroups) の概要

レッドハット株式会社
グローバルサービス本部
プラットフォームソリューショングループ
ソリューションアーキテクト
平　初
htaira@redhat.com

自己紹介

- レッドハット株式会社 グローバルサービス本部 ソリューションアーキテクト
- 平　初　（たいら　はじめ）
- レッドハット株式会社で Linux の標準仮想化技術 Linux KVM の普及、啓蒙活動に従事。最近では、レッドハットのクラウド・仮想化ビジネスの立ち上げを行っている。
- The Fedora Project で日本語翻訳チームに所属。 Anaconda や system-config-* 、 virt-manager 、 Rhythmbox 、 Brasero 、 Solang 、 Pino などのアプリケーションや、 Fedora のリリースノート、テクニカルガイド、電力管理ガイドなどのドキュメント翻訳などを行う。最近では fedoraproject.org のウェブを翻訳中。
- 著書：インプレス「 PS3 Linux 完全攻略ガイド」、技術評論社「 Fedora Core Expert 」、翔泳社「 KVM 徹底入門」、「 Xen 徹底入門」など。
- 趣味：カメラ、グルメ、カフェめぐり、翻訳作業

Control Groups(cgroups) とは

- Control Groups は、タスクセットに対する集約および分割のためのメカニズムを提供します。および、派生するすべての子プロセス、または特定のグループに属するタスクの挙動を制御することができます。
- 基本的なメカニズムとしては、カーネルがプロセスをグルーピングする方法と、それらのグループの管理権限を提供します。
- グルーピングは cgroup 仮想ファイルシステムを経由して行います。新しいグループを定義する場合は、サブディレクトリーを作成します。グループは単に新しいサブディレクトリをつくることによって任意に入れ子の階層を作ることでデプロイすることができます。
- cgroup を使う上でチューニング可能なものは、カーネルが呼ぶ ” controller” です。各 controller は 1 つまたは複数のチューニングパラメーター、および control があります。
- cgroup ファイルシステムがマウント中である時、つまり controller がアクティブな場合に指示することが可能となります。 その実体は役割の異なる controller の集まり (重複しない) を持つマウントポイントの集合体です。使用したい controller の個数回の cgroup 仮想ファイルシステムを複数回マウントすることによって作られます。
- キーとなるアイディアとしては、管理者が controller およびチューニングパラメーター

  の異なるセットを異なるグループの階層を構築できる点があります。

Control Groups(cgroups) とは

- Control Groups(cgroups) の歴史

  2006 年 9 月　google 社のエンジニアが Containers というパッチがポストした。

  2008 年 1 月　Linux 2.6.24 に Control Groups がマージされた

  2008 年 5 月　Fedora 9 のカーネルに組み込まれてリリース

  2008 年 12 月　Linux 2.6.28 で Freezer controller が追加された。

  2009 年 3 月　Linux 2.6.29 で Memory controller と Device controller 、
  Control group classifier network controller が追加された。

  2010 年 2 月　Linux 2.6.33 で Block I/O controller が追加された。

  2010 年 5 月　Linux 2.6.34　に　Modular cgroups subsystems がマージされた。

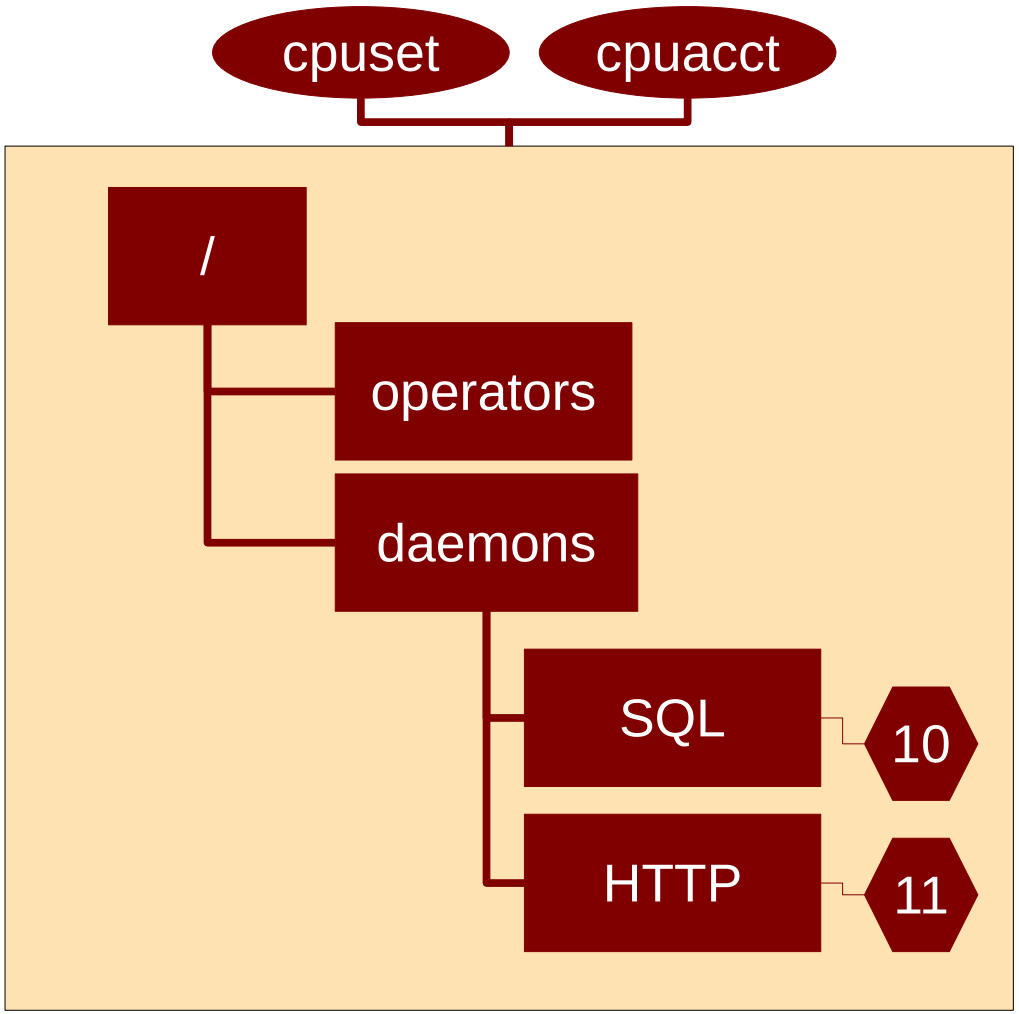
  2010 年 11 月　Red Hat Enterprise Linux 6.0 がリリースされた。(Control Groups 搭載)

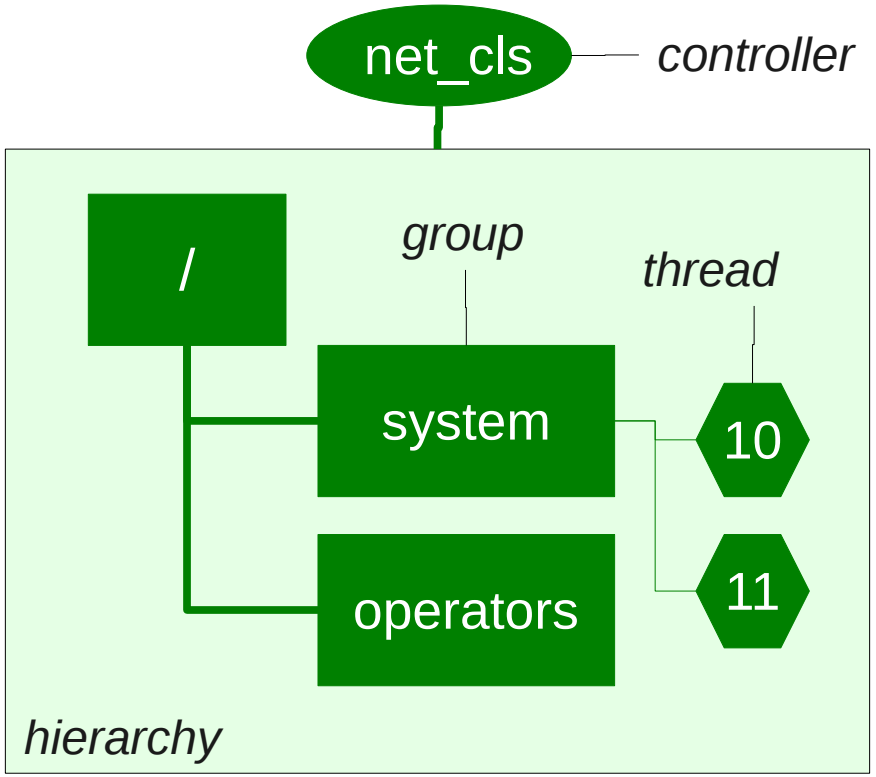
  2011 年 1 月　Linux 2.6.37　の　Block I/O controller に I/O throttling が追加された。

Control Groups(cgroups) とは

- Control Groups は、集約およびタスクセットの分離の仕組みを提供します。派生した子プロセスも対象にすることができ、階層されたグループに属するプロセスに対して、特定の動きを定義することができます。
- 定義 :
  - **cgroup** 1 つもしくは複数の subsystem に対するパラメーターのセットをタスクと一緒に関連付けます。
  - **subsystem** は、特定のタスクを行うグループを制御するために、 cgroups によって提供されるタスクグルーピングを利用するモジュールです。 subsystem は ” controller” と言われます。これはリソースをスケジューリングしたり、 cgroup ごとのリソースの制限値を定義します。どのようなプロセスでも、プロセスグループに従って動かすことができます。例えば、 virtualization subsystem など。
  - **hierarchy** はツリー構造で表現される cgroup の設定の集まりです。システム上の各タスクごとに、 cgroups hierarchy と subsystem のセットがあります。各 subsystem は、 hierarchy に含まれる各 cgroup に紐付いたシステム固有の状態を保持しています。各 hierarchy はそれに関連づけられた cgroup 仮想ファイルシステムのインスタンスを持ちます。

階層構造の例





RHEL6 搭載の cgroups

- Control Group (cgroup)
  - SLA を定義することにより、リソース競合の削減することができ、アプリケーションのパフォーマンス増加予測に対して対応できます。次の項目に対してリソース管理ができます。
    - CPU/CPUSET (cpu/cpuacct/cpuset)
    - メモリー（memory)
    - ネットワーク（net_cls)
    - ディスク I/O (blkio)

カーネルリソース管理

- cgroup – Control group
  - Control group は” resource controller” を集約およびタスクセットの分離の仕組みを提供します。派生した子プロセスも対象にすることができ、階層されたグループに属するプロセスに対して、特定の動きを定義することができます。プロセススケジューリング、メモリー割り当て、ネットワーク帯域に対して有効です。
    - システムリソースの使用率をモニタリングするために追跡できます。
    - システム管理者はツールによってグループごとにリソースの利用許可 / 不許可を決めることができます。
- Memory resource controller
  - cgroup はタスクグループのメモリーの動作を分離します。 cgroup を使えば、システムで使っていない部分のメモリー (ページング含む) に対して、以下のことが実現できます。
    - アプリケーションもしくはアプリケーショングループを同士を分離します。
    - メモリー使用量に上限を設定した cgroup を作成できます。
- cgroup スケジューラー
  - CFS – 階層型で比例した公平なスケジューラー（SCHED_OTHER)
  - 一定の帯域幅の制限で固定優先度のスケジューラー（SCHED_FIFO)

カーネルリソース管理 (続き)

- I/O controller
  - I/O 帯域幅を指定します。(ディスクコントローラーのキューの深さを基本とする)
- Network controller
  - 一般的なネットワーク層と NIC の間のクラスとキューを定義します。異なる優先度のクラス識別子のタグが付いたパケットや、トラフィックシェーピングのために異なる送信キューに送り込んだりします。
- libcgroup
  - cgroup の作成、削除、移動、設定の管理を行います。
  - ルールベースの自動タスク配置のルールや、 PAM モジュール、デーモン、 uid/gid に基づく、ベースのルールを定義することが出来ます。
- **実例となる cgroup の使い方**
  - たとえばデータベースの負荷を 90% まで、後ろで動くバックアップの負荷を 10% に割り振ったりすることができます。
  - 仮想ホスティングプロバイダーの場合、価格に応じた QoS(quality of service) を設定したりすることができます。

RHEL6 搭載の cgroups

- cgroup controllers
  - **memory**: Memory controller
    - RAM とスワップの使用量に上限を設定したり、グループに属するすべてのプロセスのメモリー使用状況を確認できるようにします。
  - **cpuset**: CPUSET controller
    - グループに属するプロセスの CPU 割り当てを指定したり、 CPU 間での切り替えの制御します。
  - **cpuacct**: CPU accounting controller
    - グループに属するプロセスの CPU 使用率の情報を収集します。
  - **cpu**: CPU schedule controller
    - グループに属するプロセスの優先度を管理します。 nice コマンドによる従来型の管理よりも詳細に指定できます。
  - **devices**: Device controller
    - キャラクターデバイスおよびブロックデバイスに対するアクセスコントロールリストを定義します。

RHEL6 に搭載の cgroups

(続き)

- **freezer**: Freezer controller
  - グループ内のプロセスの実行を一時停止、再開する仕組みを用意します。グループ全体を SIGSTOP するものだと思ってください。
- **net_cls**: Control group classifier network controller
  - 'tc' ネットワーククラスとプロセスを関連付けることにより、ネットワークの使用率を制御できます。
- 実装例としては、 libvirt LXC ドライバー (コンテナーベースの仮想化) は、 net_cls と cpuset の集合を除いて、これらすべてのコントローラーを使用しています。
- libvirt QEMU ドライバーは cpu と devices のコントローラーのみを使用します。

Subsystems - cpuacct

- 計算に必要な CPU サイクルをグループのメンバーによって指定することができます。
- パラメーター：

  `cpuacct.usage` – グループで利用した CPU サイクル数

  `cpuacct.usage_percpu` – グループで利用した CPU サイクル数（CPU ごと）
- 例：
  - 'daemons' のメンバーが 1000 万 CPU サイクルのみ使えるように指定したい
  - そして、 100 万 CPU サイクルは SQL のみのために使いたい。

Subsystems - cpu

- スケジューラー優先度が指定できます。
- パラメーター：

  `cpu.shares` – グループ内のスレッドの優先度、他のグループとの比率

- 例：
  - SQL は HTTP よりも 2 倍の CPU サイクルを消費します。

Subsystems - memory

- グループ内に属するプロセスのメモリー量の上限を設定します。
- パラメーター（memory.txt を参照):

  memory.limit_in_bytes – グループ内に属するプロセスに対して許可されたメモリー割り当ての最大値

  memory.max_usage_in_bytes – 対象のプロセスで使用された最大のメモリー量

  memory.stat – 現在のメモリー統計情報（RSS, swap, ...）
- 例 :
  - HTTP にはメモリーを 3GB まで使用させる。

Subsystems - devices

- グループ内に属するタスクの利用できるデバイスに制限を与えます。
- パラメーター：

  devices.allow

  devices.deny
- 例：
  - DB サービスを /dev/null にアクセスさせない。

Subsystems - freezer

- CPU のグループからのすべてのメンバーを保持します。
- パラメーター：

  freezer.state – グループのステータス

  - **FROZEN** — cgroups でサスペンドされたタスクを意味します。
  - **FREEZING** — cgroups でサスペンドを行う処理中のタスクを意味します。
  - **THAWED** — cgroups でレジュームがされているタスクを意味します。

  ※freezer.state に対して書き込むことができる値は FROZEN と THAWED です。 FREEZING は書き込むことができません。読み取った場合に返される読み取り専用の値です。

- 例：
  - 何か悪いことが起きている時に、システムを調べるために、すべての子プロセスをフリーズさせたい場合に有効です。

Subsystems - net_cls

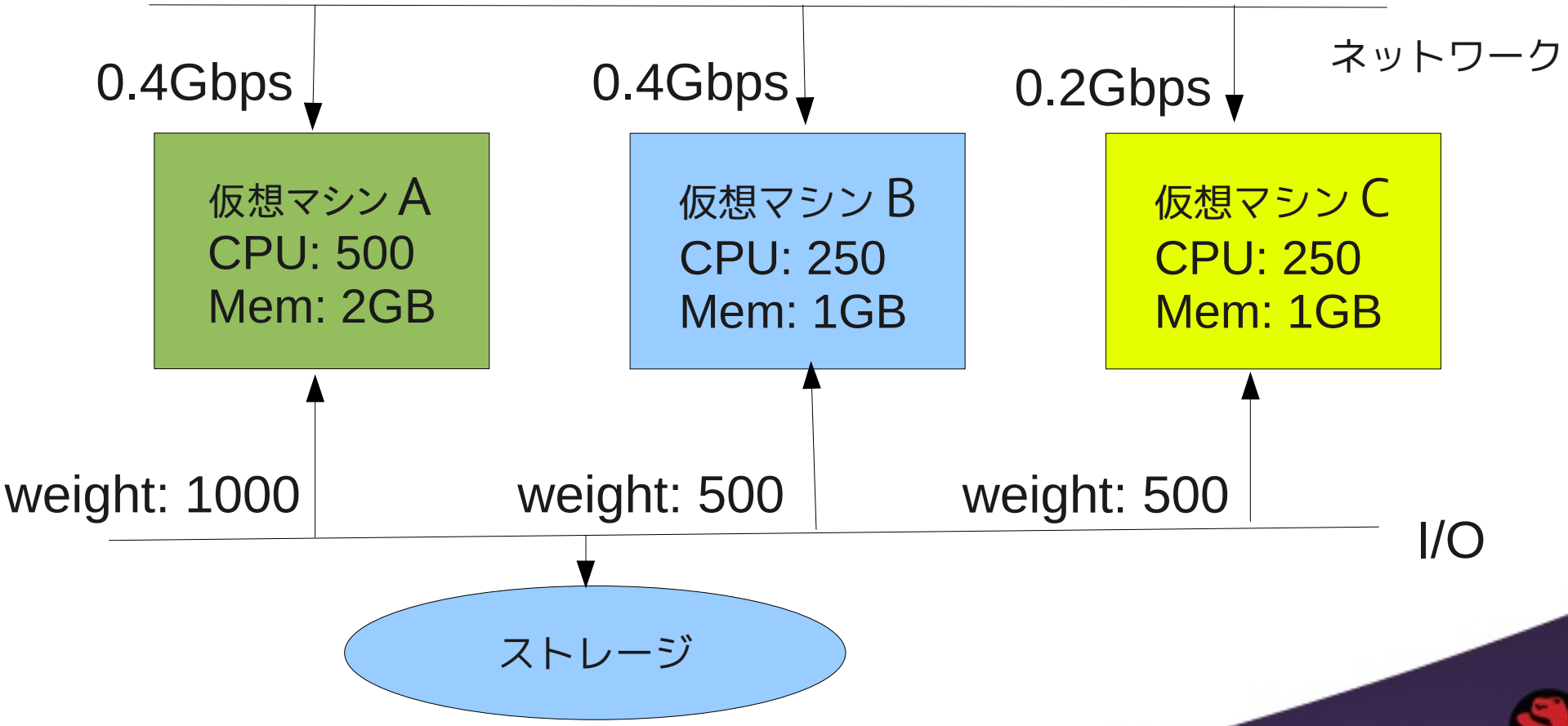
- 送信するパケットにマークをします。
- パラメーター：

  net_cls.classid – 送信するパケットに指定する class ID (トラフィックシェイパーに認識される ID).

  ※iproute2 の GIT snapshot が必要です。
- 例：
  - SQL に対して 800Mbps の帯域を予約する

カーネルリソース管理

- cgroup の実際の使い方
  - データベースの負荷が 90% 、バックアップユーティリティーの負荷を 10% で動かしたい。
  - 仮想化ホスティングプロバイダーで QoS を定義したい場合（※下の図を参照）



RHEL6 搭載の cgroups

- cgroups はシステム全体のリソースです。推定してはいけません。 cgroup controller がどのように取り付けられるか、また、 controller がどのような hierarchy で配置されるかを規定することができます。
- cgroups の設定は、完全にシステムの構成を判断した管理者がマウントポイントとディレクトリーのセットアップを任せなければなりません。
- その記述は、 /etc/fstab に対してマウントポイントを追加するものではありません。
- 管理者はディレクトリー階層と、どのぐらいのプロセスがその cgroup の中で動くのかを決める必要があります。
- 幸いなことに、 libcg プロジェクトによって SysVinit 準拠のサービスと、ホスト構成を支援するためのツールセットが提供されています。 RHEL6 の上では libcgroup の RPM パッケージの中に収録しています。

libcgroup

- シンプルなライブラリといくつかツールの集まり
- 現在も開発中
- Fedora12 よりも古い Fedora を使っている場合は、次のように実行することでインストールできます。

```
# yum install libcgroup
```

※1 Fedora 8 から libcgroup パッケージが存在します。
※2 Fedora 10 から cgconfig サービスや設定ファイルの概念が実装されました。

libcgroup - cgred

- このサービスは指定されたユーザー ID やグループ ID に応じて、

  プロセスを配布します。
- 設定ファイルは **/etc/cgrules.conf** で記述方法は次のようなスタイルです。

```
john        cpu     operators
@operator   cpu     operators
apache      cpu     daemons/http
```

libcgroup - tools

- cgexec
  - 特定のグループに所属するプロセスを開始する。
- cgclassify
  - 指定のプロセスを特定のグループに移動する。

cgroups を始めてみよう

- cgroups で重要なファイル
- cgroups に対する設定ファイルで一番重要な /etc/cgconfig.conf
- この設定では、２つの興味深い点があります。
  - ステップ１として、 controller がどこにマウントされているか宣言しています。次のような表記で複数の controller を１つのマウントポイントにマウントすることができます。

```
mount {
    cpu     = /dev/cgroups;
    cpuacct = /dev/cgroups;
    memory  = /dev/cgroups;
    devices = /dev/cgroups;
}
```

※Red Hat および Fedora のデフォルトとは異なります。

  - この設定内容で /dev/cgroups をルートとして、複数の cgroups の hierarchy を形成します。

cgroups を始めてみよう

- cgroups に対する設定ファイル /etc/cgconfig.conf の続き
  - ステップ 2 として、もし、あなたがさらに詳細な設定を行いたい場合、例えば、 1 つの hierarchy を CPU スケジュール管理だけにしたい場合、そしてデバイスの ACL や、メモリー管理を分けたい場合など。そのような場合には次のように設定ファイルを記述することができます。

```
mount {
    cpu     = /dev/cgroups/cpu;
    cpuacct = /dev/cgroups/cpuacct;
    memory  = /dev/cgroups/memory;
    devices = /dev/cgroups/devices;
}
```

※Red Hat および Fedora のデフォルトとは異なります。

  - この２つ目のサンプルは cgroups を個別に分けてマウントするれいです。
  - Red Hat および Fedora では /dev にマウントせず、 /cgroup にマウントします。

cgroups を始めてみよう

- /etc/cgconfig.conf の他の例として次のようなものがあります。
  - グループとリソースアライメントを作成しています。

```
mount {
    cpu =     /cgroup/cpu;
    cpuacct = /cgroup/cpu;
}

group daemons/www {
    perm {
        task {
            uid = root;
            gid = webmaster;
        }
        admin {
            uid = root;
            gid = root;
        }
    }
    cpu {
        cpu.shares = 1000;
    }
}
```

```
group daemons/ftp {
    perm {
        task {
            uid = root;
            gid = ftpmaster;
        }
        admin {
            uid = root;
            gid = root;
        }
    }
    cpu {
        cpu.shares = 500;
    }
}
```

cgroups を始めてみよう

- 前のページの例は次のようなコマンドを実行しています。
  - cpu と cpuacct の 2 つの subsystem の hierarchy を作成して、 cpu.shares パラメーターを指定します。

```
mkdir /cgroup/cpu
mount -t cgroup -o cpu,cpuacct cpu /cgroup/cpu

mkdir -p /cgroup/cpu/daemons/www
chown root:root /cgroup/cpu/daemons/www/*
chown root:webmaster /cgroup/cpu/daemons/www/tasks
echo 1000 > /cgroup/cpu/daemons/www/cpu.shares

mkdir -p /cgroup/cpu/daemons/ftp
chown root:root /cgroup/cpu/daemons/ftp/*
chown root:ftpmaster /cgroup/cpu/daemons/ftp/tasks
echo 500 > /cgroup/cpu/daemons/ftp/cpu.shares
```

- daemons というグループは最初のサブグループが作成された時に、自動的に作成されます。
- デフォルトではパラメーターは root ユーザーのみがアクセスできます。
- 両方の subsystem (cpuacct/cpu) は同じディレクトリーにマウントされています。
  すべてのグループは暗黙の cpu/cpuacct によって管理されます。 明示的な cpuacct セクションがある時も同様です。

cgroups の基本的な操作

- その他の設定ファイルは /etc/cgrules.conf です。
  - このファイルはユーザーやプロセスに対してリソースコントローラーを自動アサインするルールを定義することができます。
- cgroups の設定が行われており、サービスが有効であることを確認して下さい。

```
[jmh@oddjob cgroups]$ chkconfig --list cgconfig
cgconfig        0:off  1:off  2:on   3:on   4:on   5:on   6:off
```

- お使いのシステムの cgroups を開始します。
  (/etc/cgconfig.conf に基本的な設定が行われた事を確認した後で)

```
[jmh@oddjob cgroups]$ service cgconfig start
```

- /cgroup ディレクトリーの下に cgroup hierarchy を作成されます。

```
[jmh@oddjob cgroups]$ ls /cgroup/
cpu  cpuacct  cpuset  devices  freezer  memory  net_cls
```

cgroups の基本的な操作

- cgroups を恒久的に有効化するには、 chkconfig コマンドで cgconfig サービスを自動起動にするだけで構いません。

```
[jmh@oddjob cgroups]$ chkconfig --list cgconfig
cgconfig        0:off 1:off 2:on  3:on  4:on  5:on  6:off
```

cgroups の基本的な操作

- プロセスをグループに追加する場合

  [root@oddjob]# echo 1 > /cgroup/cpuset/daemons/tasks

  - サブグループのパラメータ設定は、上位のグループから継承します。
  - 再起動後、一時的な設定は失われます。

- グループからプロセスを削除します。(間接的にしか行えません)

  [root@oddjob]# echo 1 > /cgroup/cpusets/tasks

  ※ トップレベルのグループにプロセス ID を追加というスタイルになります。

- 特定のプロセスがどのグループに含まれるのか確認する場合

  [root@oddjob]# cat /proc/1/cgroup

- グループに含まれるプロセスの一覧

  [root@oddjob]# cat /cgroup/cpusets/daemons/tasks

- パラメーターから値を取得する方法

  [root@oddjob]# cat /cgroup/cpusets/daemons/cpuacct.cycles

- パラメーターに値を設定する方法

  [root@oddjob]# echo 0 > /cgroup/cpusets/daemons/cpuacct.cycles

cgroups を始めてみよう

- この例では、“ cpuset” controller の test1 グループに
  プロセスを配備する手順を示したものです。

```
[root@oddjob ]# cd /cgroup/cpuset
[root@oddjob ]# mkdir test1 ( グループ名を意味します。 )
```

- パラメーターの値を変更するファイルは、任意のディレクトリー
  の中に自動的に作成されます。これらのファイルは cgroup の
  リソースとして管理されています。

```
[root@oddjob test1]# ls
cgroup.procs  cpuset.mem_exclusive cpuset.memory_pressure
  cpuset.mems  notify_on_release cpuset.cpu_exclusive
  cpuset.mem_hardwall  cpuset.memory_spread_page
  cpuset.sched_load_balance  tasks cpuset.cpus
  cpuset.memory_migrate  cpuset.memory_spread_slab
  cpuset.sched_relax_domain_level
```

cgroups を始めてみよう

- デフォルトではファイル群の中身は空ので、値を echo コマンドで設定することにより、以下のように制御することができます。

```
[root@oddjob test1]# echo "0-1" > /cgroup/cpuset/test1/cpuset.cpus
```

※ ここで指定する CPU が提供されている必要があります。

- 指定した cgroup の中で任意のコマンドを実行したい場合、
  cgexec コマンドでグループを指定して動かします。

```
[root@oddjob ]# cgexec -g cpuset:test1 <command>
[root@oddjob ]# cat /cgroup/cpuset/test1/tasks
```

※ このグループに含まれる PID の一覧が表示されることでしょう。

cgroups を始めてみよう

- すでに実行中の処理を他の cgroup ”test2” に移動したい場合は、
  次のように cgclassify コマンドを実行します。
  [root@oddjob test1]# cgclassify -g cpuset:test2 <test1 内のすべての PID>
- 複数のグループを指定して動かしたい場合 (例えば、 cpus と memory) 、
  次のように指定できます。
  [root@oddjob test1]# cgexec -g cpus,memory:test1 <command>
- cgroup 内の CPU 数は動的に変更可能で、メモリー割り当て量も
  動的に増加可能です。メモリー割り当て量の縮小はサポートしていません。

cgroups の利用例（daemon 編）

- 現在の cgroup のタスクを確認したい場合、次のコマンドで確認できます。
  [root@oddjob]# ps -o cgroup
  [root@oddjob]# cat /proc/<pid>/cgroup
- 既存のサービスに特定の control group を指定した上でサービスを開始することができます。
- 多くのサービスでは設定ファイルに起動時に指定したい control group を指定できるようになっています。
- /etc/sysconfig/< サービス名 > といった名前の設定ファイルに次のように 'CGROUP_DAEMON' を指定することにより実現できます。
  CGROUP_DAEMON="cpu:/daemons/foo cpuacct:/daemons/foo"
  ※ この指定方法をサポートするサービスは、 /etc/sysconfig/< サービス名 > の設定ファイルを読み、かつ、プログラムの起動時に /etc/init.d/functions で定義された daemon() 関数を使用して記述してある必要があります。

cgroups の利用例（daemon 編）

- 例えば、 libvirtd を cgroup の管理下 (memory controller) で動かしたい場合、

  cgroup 'virt' を作成し、 libvirtd と派生したプロセスに適用します。

  ※ 例として、メモリーのセル番号を指定したり、上限を設定したい場合など
- メモリー管理のための “ memory” という指定を、 cgroup 'virt' 内に作成します。具体的には /etc/cgconfig.conf の中に次のように追記します。

```
group virt {
    memory {
        memory.limit_in_bytes = 750M;
    }
}
```

- /etc/sysconfig/libvirtd の中に次のような指定を追記 (もしくは変更) します。

CGROUP_DAEMON="memory:/virt”

cgroups の利用例（daemon 編）

- cgconfig と libvirtd サービスを再起動します。

```
[root@oddjob ]# service cgconfig restart
[root@oddjob ]# service libvirtd restart
```

- libvirtd が cgroup の管理下に置かれているかを確認します。

```
[root@oddjob ]# PID=$(pgrep libvirtd)
[root@oddjob ]# cat /proc/${PID}/cgroup
7:net_cls:/
6:freezer:/
5:devices:/
4:memory:/virt
3:cpuacct:/
2:cpu:/
1:cpuset:/
```

- “memory” controller の virt グループのプロセス一覧を確認します。

```
[root@oddjob ]# cat /cgroup/memory/virt/tasks
```

cgroups の利用例（blkio 編）

- 一番簡単なテスト方法は異なる croup に所属した２つの dd スレッドを起動するというものです。
- 次のように実行すると試すことができます。
  - blkio コントローラーがロードされていることを確認（/cgroup/blkio）
  - I/O エレベーターを CFQ スケジューラーにします。

```
[root@oddjob ]# echo cfq > /sys/devices/virtual/block/
  $HOSTDEV/queue/scheduler
```

  - I/O group isolation を有効にします。

```
[root@oddjob ]# echo 1 > /sys/devices/virtual/block/
  $HOSTDEV/queue/iosched/group_isolation
```

  - test1 と test2 という２つの cgroup を作成します。

```
[root@oddjob ]# mkdir -p /cgroup/blkio/test1  /cgroup/blkio/test2
```

  - test1 と test2 グループにウェイトを設定します。

```
[root@oddjob ]# echo 1000 > /cgroup/blkio/test1/blkio.weight
[root@oddjob ]# echo 500 > /cgroup/blkio/test2/blkio.weight
```

cgroups の利用例（blkio 編）

- 同じディスクに同一サイズのファイル largefile1 、 largefile2（ 2GB ぐらい）を作成します。
- ２つの異なる cgroup に所属した dd コマンドをそれぞれ１つずつ起動して、先ほど作成したファイルを同時に読み込みます。

```
[root@oddjob ]# sync
[root@oddjob ]# echo 3 > /proc/sys/vm/drop_caches
[root@oddjob ]# time dd if=/blkiotest/largefile1 of=/dev/null &
[root@oddjob ]# echo $! > /cgroup/blkio/test1/tasks
[root@oddjob ]# cat /cgroup/blkio/test1/tasks
[root@oddjob ]# time dd if=/blkiotest/largefile2 of=/dev/null &
[root@oddjob ]# echo $! > /cgroup/blkio/test2/tasks
[root@oddjob ]# cat /cgroup/test2/tasks
```

※ なお、 Slab とページキャッシュをクリアする次のコマンドは
システムに高負荷を与えるので実機では確認を取った上で実行してください。

```
# echo 3 > /proc/sys/vm/drop_caches
```

cgroups の利用例（blkio 編）

- 解説
  - 'test1' の dd コマンドが終了する前に 'test2' の dd コマンドを実行する必要があります。
  - もっと正確なデータを取得したい場合、スクリプトで test1 と test2 blkio.disk_time と blkio.disk_sectors を取得してください。
    そうするとディスクに何ミリ秒アクセスしたのか、ディスクのどのぐらいのセクター数を転送したのかが確認できます。
  - ディスク割り当て時間を公平に提供しているので、 blkio.time は cgroups の weight に比例する必要があります。

cgroups の利用例（blkio 編）

- blkio.weight
  - cgroup ごとに重み (weight) を指定します。
  - 現在、 weight には 100 ～ 1000 の値を指定することができます。
- blkio.time
  - cgroup でデバイスごとにディスクの割り当て時間をミリ秒単位で指定することができます。
  - 最初のフィールドはブロックデバイスのメジャー番号とマイナー番号を表しています。スペースで区切られた次の値はディスクの割り当て時間をミリ秒単位で表しています。
- blkio.sectors
  - グループに属するプロセスが読み書きしたディスクのセクター数を表しています。
  - 最初のフィールドはブロックデバイスのメジャー番号とマイナー番号を表しています。スペースで区切られた次の値は読み書きしたディスクのセクター数を表しています。

cgroups の利用例（KVM 編）

- お使いのマシンに 2 つの仮想マシンを KVM で用意します
- そして、仮想 CPU 数をお使いのマシンの物理 CPU 数と同じにします。
  - この理由は CPU シェア割り当てをデモするためだけのものです。
- 仮想マシンを起動します。
- スケジューリングのプロパティーを確認します。

```
[root@oddjob ]# virsh schedinfo rhel6vm02
Scheduler      : posix
cpu_shares     : 1000
```

- virsh コマンドでスケジューリングのプロパティーを調整します。

```
[root@oddjob ]# virsh schedinfo --set cpu_shares=500 rhel6vm01
Scheduler      : posix
cpu_shares     : 500
```

- virt-top コマンドを使って CPU 使用率をモニターします。

cgroups の利用例（メモリー編）

- 16GB/8CPU 搭載のシステム上で、
  現在実行中のシェルのプロセス全体に対して、
  cgroup で 1GB/2CPU のサブセットを指定

```
[root@oddjob ]# mkdir -p /cgroup/cpusets/memtest
[root@oddjob ]# cd /cgroup/cpusets/memtest
[root@oddjob ]# echo 0 > cpuset.mems
[root@oddjob ]# echo 0-1 > cpuset.cpus
[root@oddjob ]# echo 1000000000 > memory.limit_in_bytes
[root@oddjob ]# echo $$ > tasks
```

cgroups の利用例（メモリー編）

指定されたメモリー量を確保するプログラム" memory" を実行し、vmstat コマンドで動作を観察。

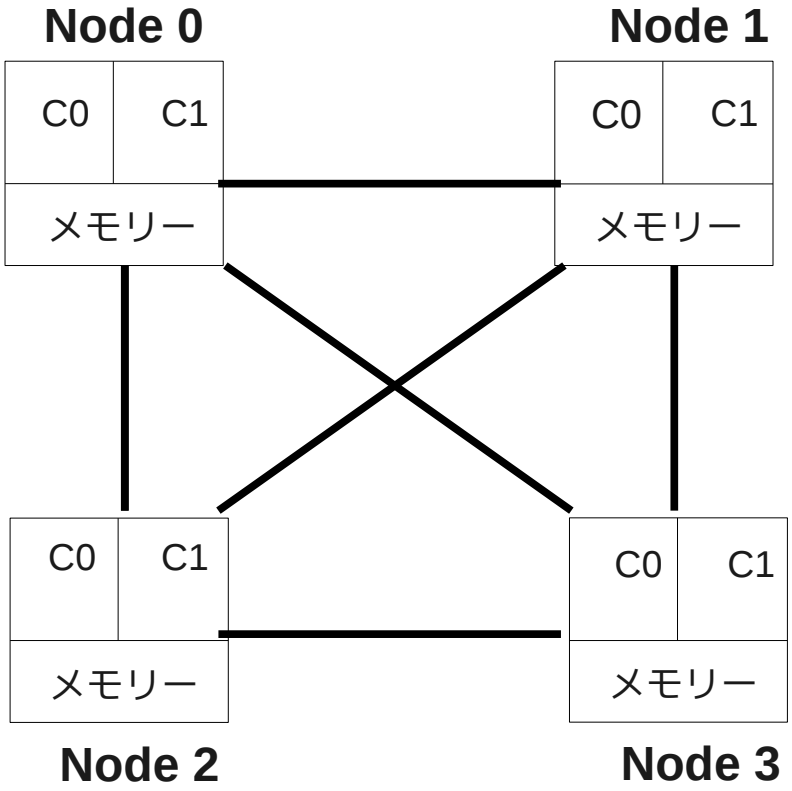
```
[root@oddjob ]#  /usr/local/bin/memory 2GB &
[root@oddjob ]#  vmstat 1
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu------
 r  b   swpd     free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0      0 15465636  33636 459612    5   67    16    68   46   27  1  0 99  0  0
 0  0      0 15465504  33636 459612    0    0     0     0  246  160  0  0 100  0  0
 1  0      0 14598736  33636 459612    0    0     0     0 1648  299  1  5 94  0  0
 1  0 114092 14484980  33636 459528    0 114176    0 114176 2974 1031  0  6 82 12  0
 0  1 264672 14479896  33636 459508    0 150496    0 150496 2630  568  0  2 90  7  0
 0  1 375612 14479524  33636 459612    0 110940    0 110940 2301  322  0  4 76 19  0
 0  1 500064 14477788  33636 459692    0 124452    0 124452 1869  273  0  2 91  7  0
 1  0 609908 14477540  33636 459628    0 109888    0 109888 1960  198  0  8 76 15  0
 0  1 709996 14478476  33636 459400    0 100044    0 100044 2243  260  0  3 91  6  0
 0  1 818924 14478352  33636 459600    0 108928    0 108928 2210  342  0  4 77 18  0
 0  1 932920 14478476  33636 459548    0 113996    0 113996 1951  303  0  2 91  7  0
 1  0 1055352 14476864  33636 459516   0 122560    0 122560 1885  197  0  6 76 17  0
```

プロセッサーの種類とノードの配置

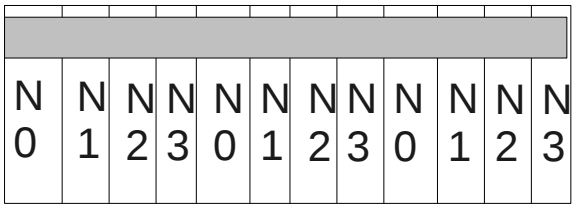
```
[root@oddjob ]# cat /proc/cpuinfo
Processor       : 0    < 論理コア番号 >
physical id     : 0    < ソケット番号 >
Siblings        : 16   < ソケットあたりの論理コア数 >
core id         : 0    < ソケットごとの論理コア番号 >
cpu cores       : 8    < ソケットあたりの物理コア数 >
```

```
[root@oddjob ]# cat /sys/devices/system/node/node*/cpulist
node0: 0-3
node1: 4-7
```

一般的な NUMA のレイアウト

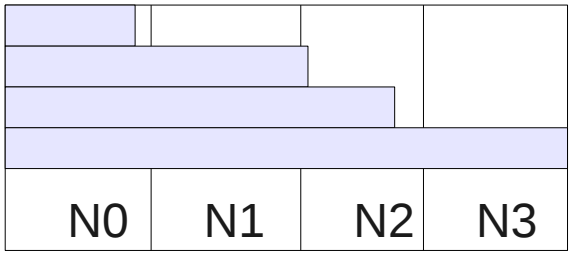


Node0 上 C0 のプロセスメモリー



インターリーブ (非 NUMA 環境)

Node0 上 C0 のメモリー処理



ノン・インターリーブ (NUMA 環境)

Red Hat 系でよく使うツール

CPU Tools

1 – top
2 – vmstat
3 – ps aux
4 – mpstat -P all
5 – sar -u
6 – iostat
7 – oprofile
8 – gnome-system-monitor
9 – KDE-monitor
10 – /proc

Memory Tools

1 – top
2 – vmstat -s
3 – ps aur
4 – ipcs
5 – sar -r -B -W
6 – free
7 – oprofile
8 – gnome-system-monitor
9 – KDE-monitor
10 – /proc

Process Tools

1 – top
2 – ps -o pmem
3 – gprof
4 – strace,ltrace
5 – sar

Disk Tools

1 – iostat -x
2 – vmstat - D
3 – sar -DEV #
4 – nfsstat
5 – NEED MORE!

参考となる情報源

- kernel.org の Documentation/cgroups/cgroups.txt
  http://www.kernel.org/doc/Documentation/cgroups/cgroups.txt

- Libcg - Library for Control Groups
  http://libcg.sourceforge.net/

- Using CGroups with libvirt and LXC/KVM guests in Fedora 12
  http://berrange.com/posts/2009/12/03/using-cgroups-with-libvirt-and-lxckvm-guests-in-fedora-12/
  ※Daniel P. Berrangé は Red Hat のエンジニアです。

- blog: Linux CGroups: Subsystems as Modules
  http://cgrouphacking.blogspot.com/