Linux 2.6ボトルネック

2005年6月23日
山幡 為佐久(yamahata@valinux.co.jp)
VA Linux Systems Japan株式会社 技術本部

発表概要

- 2.6での改善点
- プロファイリング
- チューニングのポイント
- キャッシュメモリ
- チューニング例紹介
- 今後のVA Linuxの取り組み

2.6での改善点

2.4 -> 2.6での改善項目

- 2.4で主だったボトルネックの解消
- スケーラビリティーの改善

O(1)スケジューラ

- O(1)スケジューラ
  - プロセス数に対するスケーラビリティー向上
  - スケジューリング時の全プロセス検索を無くしたことによるキャッシュミス改善
    - カーネルスタックのカラーリングによる改善も存在した

ページキャッシュの改善

- ページキャッシュ/バッファキャッシュの統合
- inode毎にradix treeが導入され、ページキャッシュアクセスの並列度が向上
- ページの書出しもinode単位での書き出しになり効率改善
- rmapが導入され、ページ解放処理が効率化
- buddy allocaterにhot/cold pageの導入
  - pageの確保/解放時にhot/coldを指定して、cpuキャッシュヒット率の向上させる

遅延処理の改善

- workqueueの導入
  - 各cpu上で並列に動作し、スケーラビリティー向上
- taskqueueのtaskletへの全面移行
- Bhハンドラをソフト割り込みで全面的に置き換え
  - 各CPUで並列に動作するようになった。
  - 特にSCSI割り込み処理が並列動作するのは大きい
- タイマー処理もソフト割り込みを使用してCPU毎に並列動作するようになった。

ロックの細粒度化

- 一層のロックの細粒度化(BKLの削減)
  - vfsからのbklの削除
  - block deviceのブロックデバイス毎のロック
  - まだ残っている部分も
    - file system(nfs clinetなど)
    - rpciod

新しいロックの導入

- RCU(Read Copy Update)の導入
  - キャッシュに負荷をかけないアルゴリズム
  - CPU数に対するスケーラビリティーの向上
    - e.g. ネットワークプロトコル管理、ルーティング処理などに使用される。
- seqlockの導入
  - e.g. タイマ変数の更新(xtime, xtime_lock)にしようされ、gettimeofdayの性能向上

ブロックI/O関連の改善

- bioの導入
  - 旧buffer headインターフェースはbioでエミュレーション。
  - 各ファイルシステムの対応が必要
- 各種I/Oスケジューラの導入
  - anticipatory
  - deadline
  - CFQ

AIOの導入

- 2.4ではライブラリによるスレッドを使ったエミュレーションだった
- work queueを利用した実装
- VFSがaioを意識したインターフェースに変更
  - file_operationsの変更
    - aio_read()
    - aio_write()
    - aio_fsync()
  - 現状direct I/Oにしか意味がない

その他

- SMPとキャッシュラインを意識した変数の導入(percpu)
  - スケジューラrun queueやソフト割り込みなどに使用
- NAPI(New API)の導入
- Zero Copy NFS
  - NFSサーバのzero copy化で性能向上
- NUMA対応

プロファイリング

2.6のプロファイリング

- 2.6ではチューニングがなされて、軽くなっているはずだが
- 実際にプロファイリングをしてみる

プロファイリング

- Oprofileを使用して、処理に時間が掛っていそうなところを特定する。
- nfsサーバとして負荷をかけて、oprofileでプロファイリング
- XFSを使用

プロファイリング結果

samples	%	app name	symbol name
1274047	24.7291	vmlinux	default_idle
747545	14.5098	vmlinux	xfs_iget_core
220450	4.2789	vmlinux	xfs_dir2_leaf_getdents
167999	3.2608	vmlinux	linvfs_readdir
161204	3.1290	vmlinux	xfs_dir2_put_dirent64_direct
114369	2.2199	vmlinux	_spin_lock
69383	1.3467	vmlinux	schedule
42445	0.8239	vmlinux	__d_lookup
40718	0.7903	vmlinux	_spin_lock_irqsave
40001	0.7764	vmlinux	csum_partial
39220	0.7613	vmlinux	_spin_lock_bh
36098	0.7007	oprofiled	odb_insert
35341	0.6860	vmlinux	xfs_iextract
32502	0.6309	vmlinux	nfsd_read
30993	0.6016	vmlinux	filldir_one
29199	0.5667	vmlinux	csum_partial_copy_generic

プロファイリングの結果

- 特に目立って悪いものはない
  - 2.4->2.6でチューニングされた結果か
  - 2.4では目立っていたところが無くなっている。
    - スケジューラなど
- CPU使用率は全体的に悪くなっている

writeのコードパス(ext3fs)

- 2.4

```
sys_write()
 +generic_file_write()
  +do_generic_file_write();
   +__grab_cache_page();
   +ext3_prepare_write()
    +block_prepare_write();
     +__block_prepare_write();
   +__copy_from_user();
   +ext3_commit_write()
    +generic_commit_write();
     +__block_commit_write();
```

- 2.6

AIO導入のために
パスが長くなっている

```
sys_write()
 +vfs_write();
  +do_sync_write()
   +ext3_file_write()
    +generic_file_aio_write();
    +__generic_file_aio_write_nolock();
      +generic_file_buffered_write();
       +__grab_cache_page();
      +ext3_prepare_write()
       +block_prepare_write();
       +__block_prepare_write();
      +filemap_copy_from_user();
      +ext3_ordered_commit_write()
       +generic_commit_write();
       +__block_commit_write();
```

ページキャッシュ追加処理例

2.4

```
__grab_cache_page()
 +__find_lock_page();
  +__find_lock_page_helper();
   +__find_page_nolock();
 +page_cache_alloc();
 +add_to_page_cache_unique()
  +__find_page_nolock();
  +__add_to_page_cache();
   +add_page_to_inode_queue();
   +add_page_to_hash_queue();
    +inc_nr_cache_pages();
```

radix tree導入のために
パスが長くなっている

2.6

```
__grab_cache_page()
 +find_lock_page();
  +radix_tree_lookup();
 +page_cache_alloc();
 +add_to_page_cache();
  +radix_tree_preload();
   +kmem_cache_alloc();
  +radix_tree_insert();
   +radix_tree_extend();
    +radix_tree_node_alloc()
  +radix_tree_preload_end();
 +pagevec_add()
 +__pagevec_lru_add();
```

- 実行されるコードが増えている
  - aioの導入
  - radix treeの導入
  - bioの導入
    - 旧buffer headインターフェースはエミュレーション
    - 各fsが対応するしかない。
- この様な事の積み重ねで、全体的に遅くなっているのか。
  - スケーラビリティー/CPU並行性をあげる為の修正の結果か
- 地道な改善を積み重ねていくしかない。

チューニングポイント

- 今後重要となるであろう傾向について述べる。

CPU

- デュアルコア化、マルチコア化
  - 並列処理の重要性がます
  - CPU間でのキャッシュメモリ共有など非対象性が増え、NUMA的要素の発生
- メモリとの処理速度差が増大

メモリ

- メモリ量の増大
  - Gb単位のメモリ量は当たり前
  - より効率の良いページ管理が必要

ネットワーク

- ネットワーク帯域は増大
  - CPU処理速度に比べて速くなっている
    - 1Gbpsは普及してる
    - 10Gbpsも目前
  - CPU並列性をより活かす処理が必要

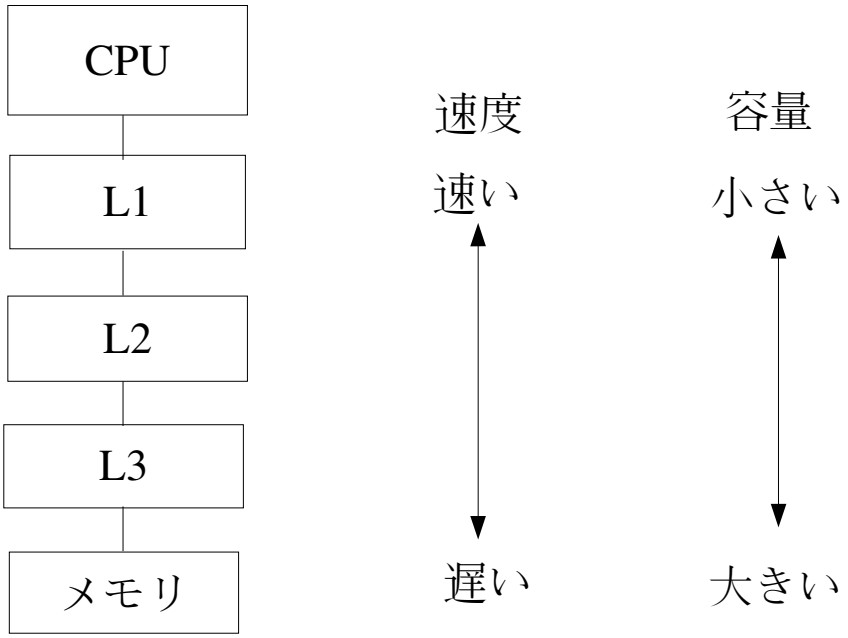
資源分配

- 多くの資源(CPU、メモリ、ネットワーク帯域)が使えるようになってきた
  - より効率的な分配が必要
  - 一方、資源の保証も出来るようにしたい。
    - CKRM(Class Based Resource Management)
  - 仮想化(xen, qemu, vmware等)

キャッシュメモリ

キャッシュメモリとは

- CPUとメモリの速度差を隠蔽。



セット連結キャッシュ

アドレス
物理アドレスか仮想アドレス(アーキテクチャによる)

セット連結キャッシュ(Cont.)

- キャッシュラインサイズ分まとめてキャッシュ
- アドレスをタグ部分とライン部分に分割
- 同ラインのメモリは同じラインに格納
- タグが一致すればキャッシュヒット
- 他CPUがwriteする場合は無効化

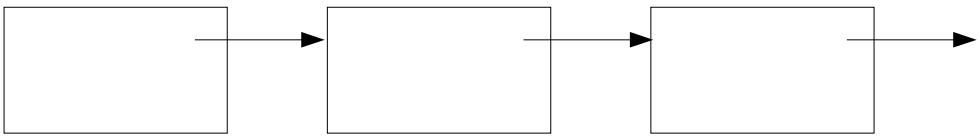
キャッシュに関するチューニング

- なるべく同じメモリを使用／同じCPUから使用する
  - キャッシュヒット率向上
  - 割り込み処理とソフト割り込みを同じCPUで動作させる。（関連する処理を同じCPUで動作）

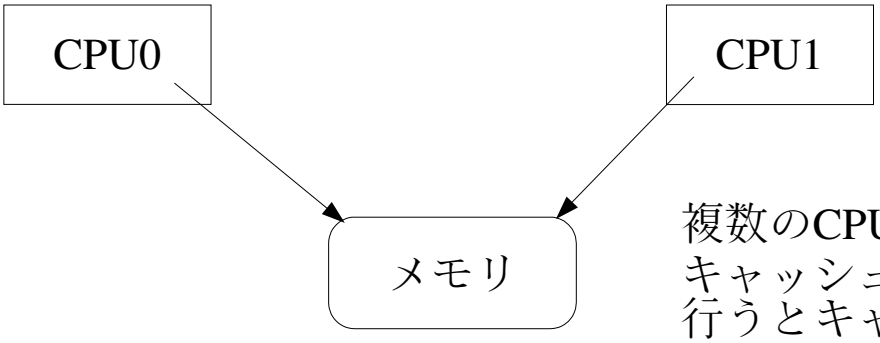
キャッシュに関するチューニング (cont.)

- キャッシュミスの削減
  - カラーリング
    - 特定のキャッシュラインにアクセスが片寄らないようにアドレスをずらす。
    - 仮想アドレスでキャッシュするアーキテクチャでは割り当てる仮想アドレスのカラーリングも効果が大きい。
  - 構造体のレイアウトの見直し
    - 同時に使用するメンバが同じキャッシュラインに載るようにする。
    - キャッシュラインを跨がないように配置する。
      - __cache_line_aglined等
  - プリフェッチ

キャッシュに関するチューニング (cont.)



リスト検索は飛び飛びのアドレスを参照するため
あまり良くない.



複数のCPUから同じ
キャッシュラインへの修正を
行うとキャッシュミスが頻発
特にspin lockは重たい

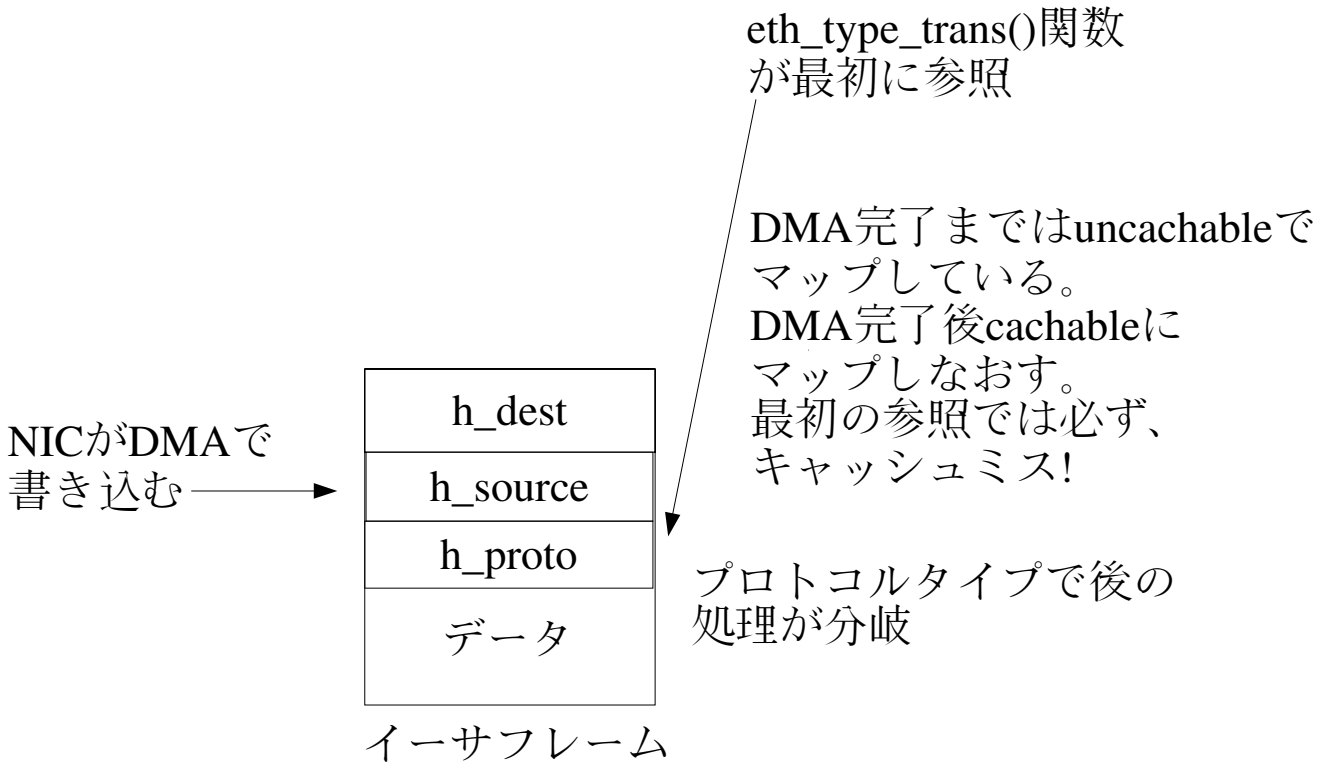
キャッシュに関するチューニング (マルチプロセッサ)

- spin lockの削減
  - RCU化
  - per cpu変数の導入
    - 各CPU毎にキャッシュラインを意識したアドレス
- atomic operationの削除
  - これもキャッシュラインを汚す。
- コピー処理の削減

チューニング例紹介

例1(eth_type_trans)

- NICが受信したイーサフレームの解析を行う関数



prefetchで改善

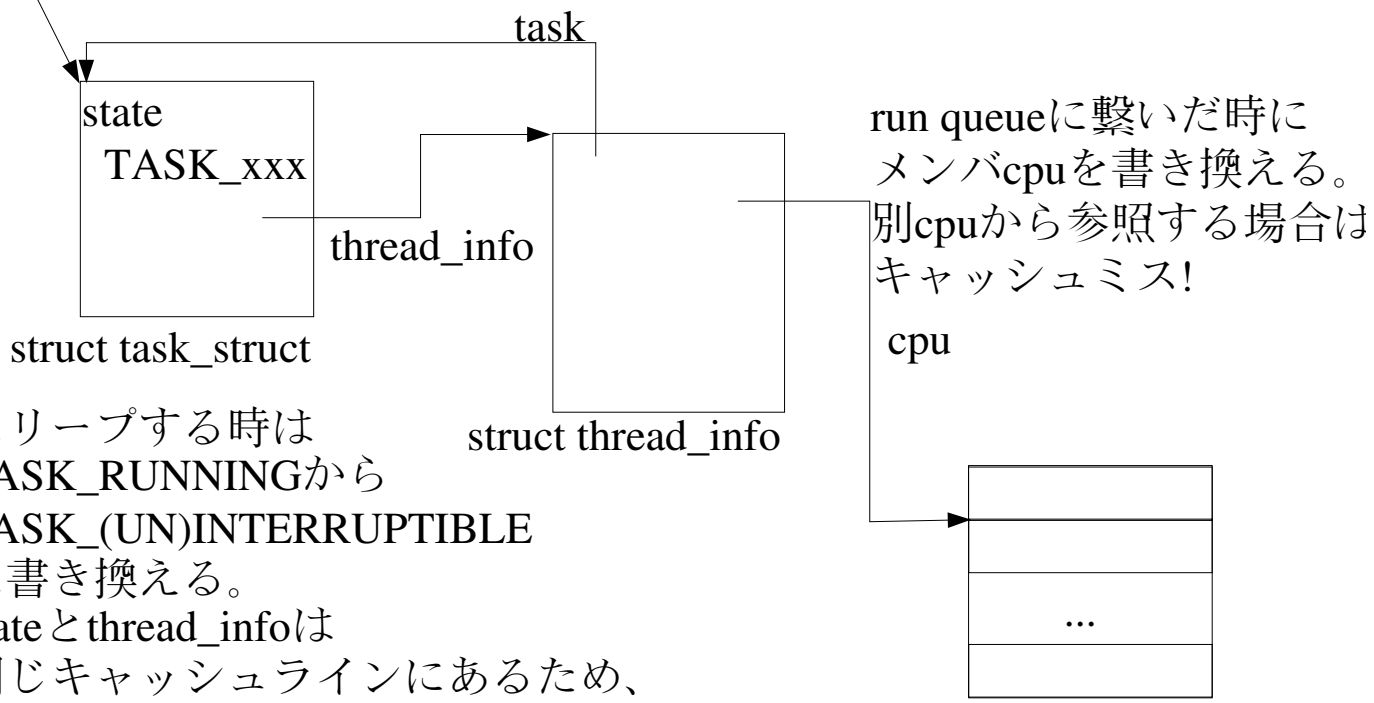
- キャッシュミスはprefetchにて改善

vma	samples	%	symbol
c03b0d40	780	0.7439	eth_type_trans
		↓	
c03b0da0	694	0.6619	eth_type_trans

事例(try_to_wake_up)

- task_structにおけるキャッシュミス
- 2.4ではスケジューラが全task_struct構造体を線形検索していた為、キャッシュミスが頻発していた。
  - カラーリングパッチが存在
  - O(1)スケジューラで改善
  - task_struct構造体がtask_struct構造体とthread_info構造体に分割された。

try_to_wake_up()　　task->thread_info->cpuの二重の参照を行う。



スリープする時は
TASK_RUNNINGから
TASK_(UN)INTERRUPTIBLE
に書き換える。
stateとthread_infoは
同じキャッシュラインにあるため、
他CPUからthead_infoを
参照するとキャッシュミス!

runqueue_t runqueues[NR_CPUS]

try_to_wake_up

- prefetchなし

```
vma              % symbol
c01148e0  0.5957 try_to_wake_up
c01148e0  0.1872
c01148e1  0.8941
c01148e4  0.2654
c01148e5  0.2878
c01148e6  0.1285
c01148e9  0.0168
c01148f0  0.1704
c01148f1  1.2043
c01148f4  0.4750
c01148f5  0.9696
c01148f8  0.7796
c0114900 10.6262 <<< p->thread_info
c0114903 12.5933 <<< p->thread_info->cpu
c011490a  1.5172
c011490c  0.0615
c011490f  0.0531
c0114914 22.6131 <<< task_rq_lock(p, &flags);
                     spin lock
```

- prefetchあり

```
vma              % symbol
c01148e0  0.5271 __try_to_wake_up
c01148e0  0.5221
c01148e1  0.5047
c01148e4  0.2262
c01148e5  0.1740
c01148e6  0.0522
c01148f0  0.1044
c01148f1  1.3227
c01148f4  0.4525
c01148f5  1.0790
c01148f8  1.2008
c0114900  0.2436 <<< p->thread_info
c0114903 15.7153 <<< p->thread_info->cpu
c011490a  1.5837
c011490c  0.0522
c011490f  0.1218
c0114914 28.6808 <<< task_rq_lock(p, &flags);
                     spin lock
```

try_to_wake_up

- CPU間をまたがったタスクのwake upはrun queueのスピンロックを伴うのでどうしても重い。
  - 少しでも処理を軽くしたい。
- プリフェッチでキャッシュミスは改善
- でもまだキャッシュミスは残る。
  - ２段目のキャッシュミスはプリフェッチでは救えない。

事例(資源管理)

- ページキャッシュ管理は2.6でもLRU管理のまま
- シーケンシャルリードを行ってしまうと有用なデータまで追い出されてしまう。
  - バックアップソフト
  - Video再生等
- もっと高度な管理が必要
- あるいは使用資源を制限するような機構が必要

資源管理

- CPUset
  - CPU及びNUMAノードを分割して、プロセス(郡)にバインド
- CKRM(Class Based Resource Management)
  - クラスに分類して利用資源を制限
    - 使用ページ数を制限できる。
      - 現状あまり良い実装ではなく、洗練が必要

今後のVA Linuxの取り組み

今後のVA Linuxの取り組み

- より一層の性能改善
  - SMP時の性能改善
  - デュアルコア/マルチコアなどNUMA的な要素のある場合の改善
  - 平行処理性改善
  - NFSサーバの改善

今後のVA Linuxの取り組み(cont.)

- 高負荷時の安定化
  - ページキャッシュ管理の改善
- 資源管理機能(CKRM)の強化
  - ページ利用制限機能の強化

質疑応答

ご静聴ありがとうございました