Empowered by Innovation NEC

大規模サーバへのLinux適用
エンタープライズ領域での2.6カーネル

NEC Linux推進センター

2004年10月15日

NEC Linux

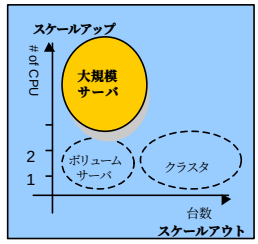
Empowered by Innovation NEC

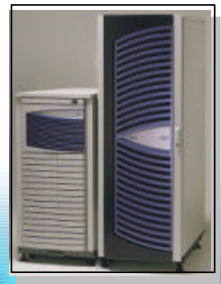
目次

Empowered by Innovation NEC

はじめに

- 大規模サーバとは？
  - 多数のCPU: 8ウェイ～32ウェイのCPU数
  - 大量のメモリ: 32GB～512GBのメモリ
  - 多数、大量の記憶装置: 10TB～100TBのストレージ
  - 多数のデバイス: HBA、イーサネット
- 適用分野
  - 共有メモリ型のワークセット（クラスタには不向き）
  - サーバ集約的なメリット（コンソリデーション）
- 技術要素
  - ハードウェアアーキテクチャ:SMP、NUMA
  - 「構成要素の数の多さ」へのソフト的な対応
    ⇒スケーラビリティ





NX7700i

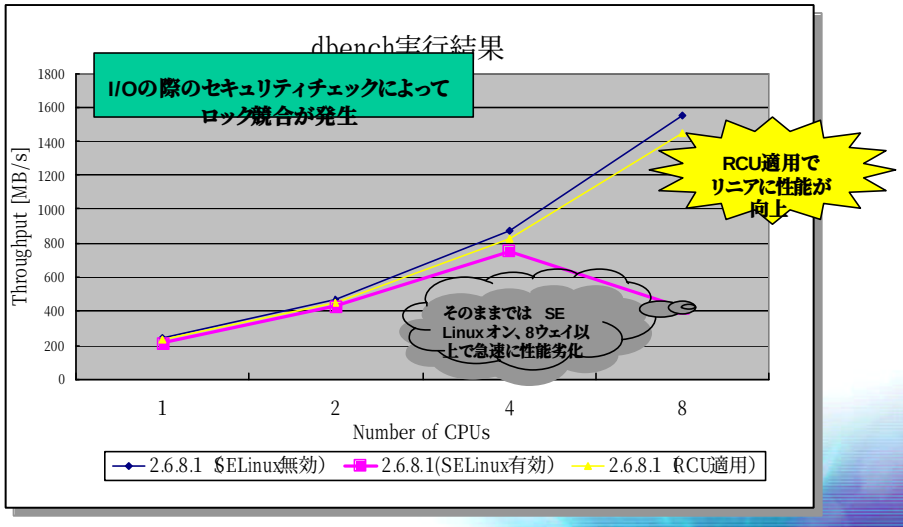
Linuxにおいても対応が必要

2.4から2.6カーネルにかけての原動力

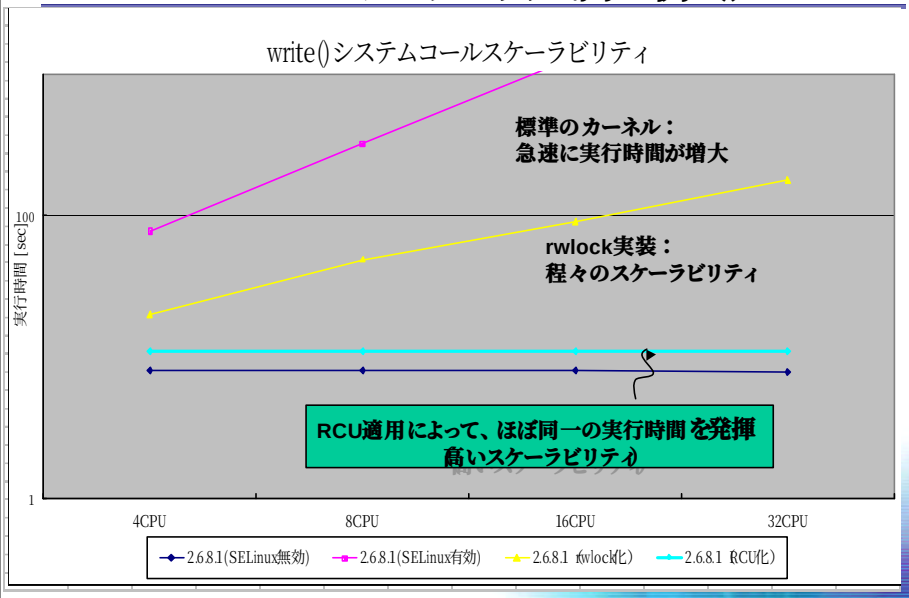
大規模マルチプロセッサ(SMP)対応: ロック競合

- 大規模マルチプロセッサ
  - 4CPU～32CPU、あるいはそれ以上の構成
- 2.4での課題
  - SMP特有の問題:スケーラビリティ
    - ロック競合問題
      - バッファのLRU管理(lru_list_lock)、ページテーブルのロック(mm->page_table_lock)、ページキャッシュ操作のロック(page_cache_lock)、ページキャッシュのLRUのロック(pagemap_lru_lock)、I/O操作のロック(io_request_lock)等々
- 2.6では
  - ロックの細分化、RCU、seqlock等のロックプリミティブの導入
    - でも、新しいコンポーネントには注意が必要
      - SELinuxのスケーラビリティ問題⇔RCU化による解消を提案中

2.6でのスケーラビリティ向上例(1)



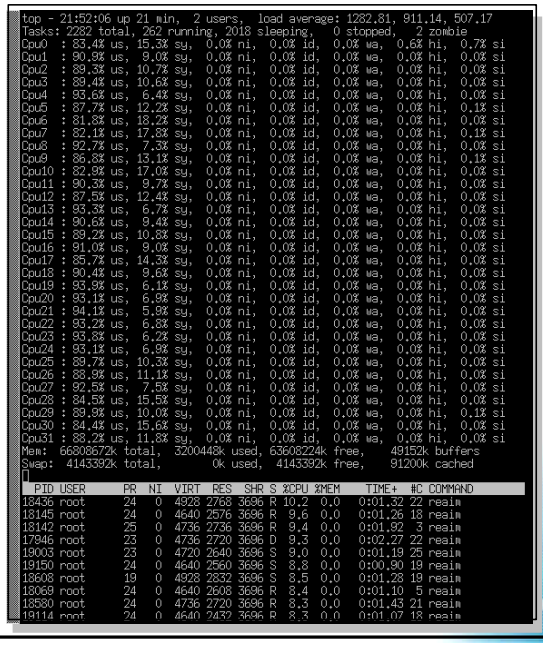
2.6でのスケーラビリティ向上例（2）



大規模マルチプロセッサ(SMP)対応(2)

- 2.6に向けての話題
  - CPUホットプラグ
    - CPUの動的追加、削除をしたい
      - 1CPUの障害でシステム全体を止めたくない
  - CPU表現
    - 64bit (long) では不足
  - topの行があふれる
    - 情報収集の際に大問題 :-)

32CPUでのtopの例

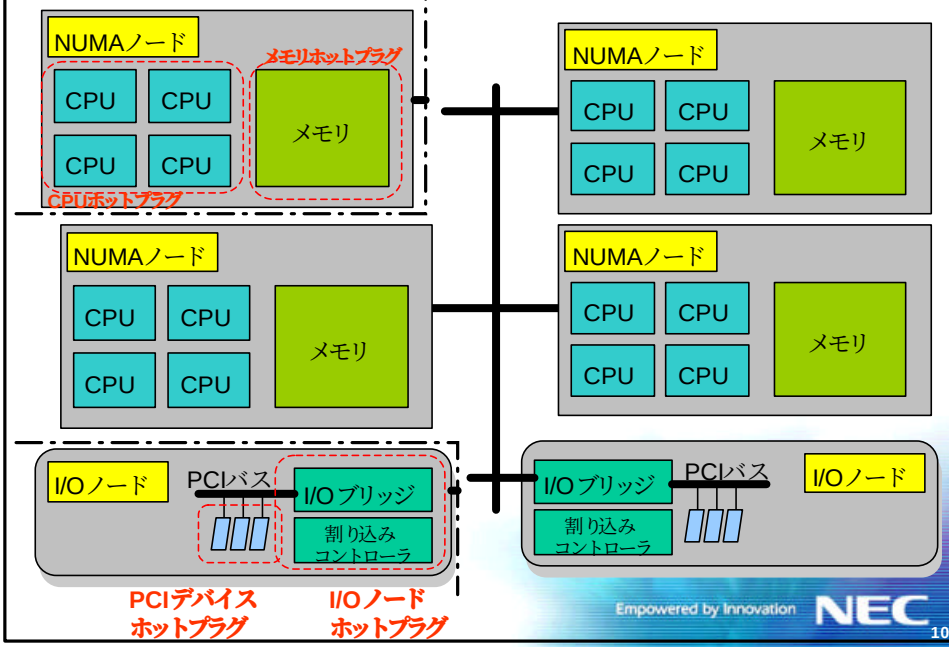


32並列ジョブの実行例
(reaim:OSDLのBM)

NUMA対応とSMT, CMP

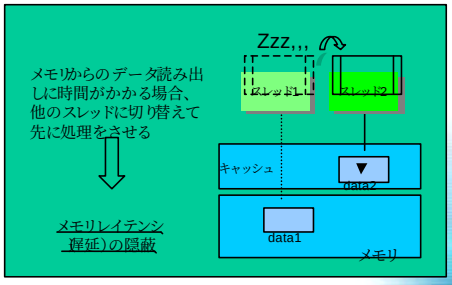
- NUMAとは
  - 大規模SMPシステム向けハードウェア技術
  - システムを小さなモジュール（ノード）に分割
    - ノードとは？「CPU、メモリ、I/Oサブシステム）の組」
    - 全体が一枚岩のシステムより扱いやすい
  - CPUとメモリの「遠近」が生じる
    - ソフト側の対応要
    - プロセスをノード内に閉じて配置したほうがよいか、それともノードにまたがるように配置したほうがよいか？
      （ハードウェア構成 -ノードのバンド幅やレイテンシー に依存）
- 2.4カーネルの時代にソフト側の対応を開始
- 2.6カーネルでは、SMT(ハイパースレッディング）などへの対応と併せた形で、一般的なインフラストラクチャとして取り込まれつつある

ノードも含めた、システム全体のホットプラグイメージ



NUMA対応とSMT, CMP(2)

- SMT(Simultaneous Multi Threading)
  - チップ内で複数スレッド（コンテキスト）を同時実行
    - メモリ待ち中に別スレッドを実行
      - メモリレイテンシ（待ち時間）の隠蔽
    - 空きリソースで別スレッドを実行
      - CPU内リソースの並列利用
    - キャッシュは共有される
  - 同一CPUに複数プロセスを配置すべきか？
    それとも別CPUに配置すべきか？



NUMA対応とSMT，CMP(3)

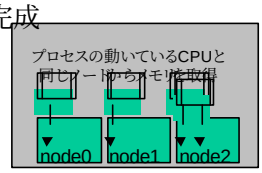
- CMP(Chip Multi Processing)
  - チップ内に複数セットのコアを持たせる
    - キャッシュは共有、非共有いずれもありうる
  - プロセスを同一CPU内コアに配置すべきか、それとも、別のCPUのコアに配置すべきか

プロセスやメモリの配置:
バスバンド幅、プロセスの種別（スレッドで空間を共有するのか、マルチプロセスで別空間をアクセスするのか、共有データは多いか少ないか）に依存

⇒ **SMTやCMPでもNUMAと同様の配慮が必要**
- **スケジューリングの際のCPU選択**
- **ロードバランスのタイミング**
- **SMT、CMP、NUMAを統合する枠組みが必要**

NUMA対応とSMT，CMP(4)

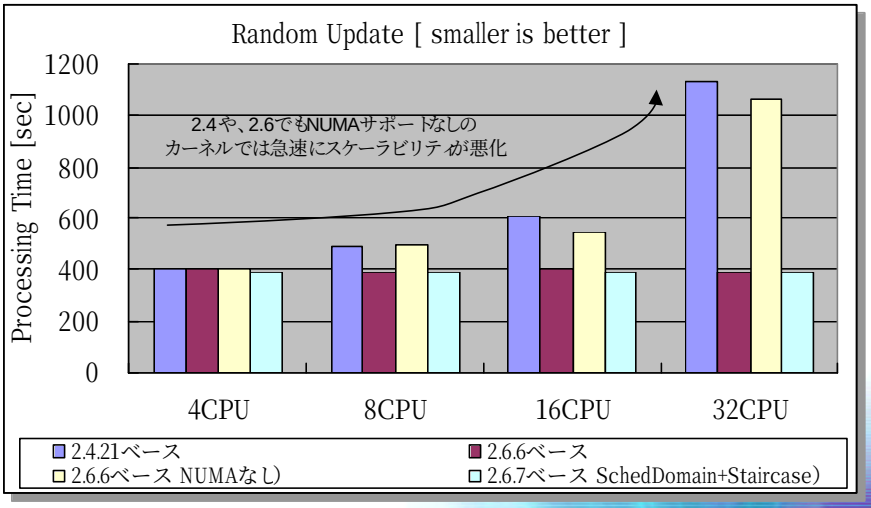
- NUMA向けのメモリ割り当ての対応
  - 2.4カーネルで、ある程度のインフラストラクチャが完成
    - DiscontigMem
      - 各ノードごとにページアロケータを持つイメージ



- スケジューラの対応
  - 2.4カーネル時代
    - O(1)スケジューラで共通的な問題は解決
      - CPU毎のランキュー
      - ランキュー間でのロードバランス（負荷を見てプロセスを移動させる）
      - デフォルトではノードを意識しない（ノード内もノード跨りも気にしない）
  - 2.6での改善
    - NUMAスケジューラ
      - ノードを意識したスケジューラ（ノード間ではロードバランスの頻度を下げる）
    - SchedDomainスケジューラ
      - 複数段構成のNUMA向け
      - より一般化された枠組みでSMTにも同一の枠組みで対応する
    - NUMA API
      - メモリやCPUのプロセス、空間への割り当てを制御するAPI
      - 「どのメモリをアクセスしても同じ性能が発揮できる」という仮定はもうできない

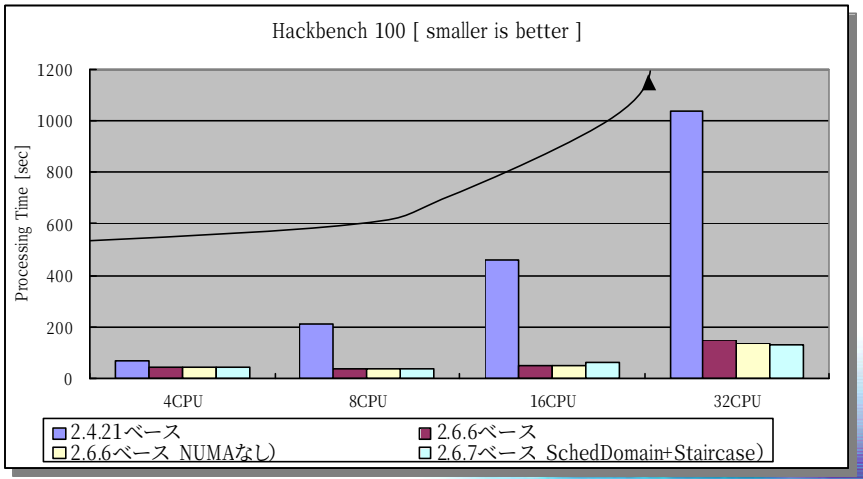
2.6でのNUMA性能向上例（1）

ランダムなメモリ更新のテスト（2.6系ではほぼ同一実行時間）



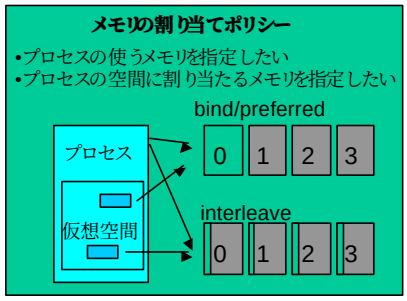
2.6でのNUMA性能向上例（2）

ランダムな通信テスト（2.4系列では幾何級数的に実行時間が伸びる）



NUMA対応とSMT, CMP(5)

- NUMA APIによる制御
  - 2.6カーネルでは、ピーク性能を発揮させるためのチューニングが可能
  - プロセスのCPUへの固定
    - sched_{set|get}affinity(2)
  - メモリの割り当てポリシ
    - プロセスの使うメモリの指定
      - {set|get}_mempolicy(2)
        - ポリシー: interleave, bind, preferred
    - 仮想空間に割り当てられるメモリの明示的な指定
      - mbind(2)
        - 仮想空間にポリシーを設定
        - ページ割り当ては、ポリシーにしたがって実施される
        - ポリシー: interleave, bind, preferred
  - ライブラリ関数群: libnuma (numa_* API)



大容量メモリへの対応

- 32GB～512GB（もしくはそれ以上）
  - 2.4での課題
    - キャッシュやバッファによるメモリの占有
      - キャッシュ類の占有（ノードキャッシュ等）
        - 大量のメモリノードが生成されることがあり、不用意に舐めたりすると酷い目にあう（一時期のXFSなど）
    - 大プロセスのスワップアウト時の仮想空間スキャン
  - 2.6のスケーラビリティ観点での強化によって軽減
    - (Object based)rmapの導入や、ロックの削減
    - それでも、新しいコンポーネントには注意が必要
      - hugetlbfs+大規模メモリ
    - 実機評価、厳しい状態でどうなるか分析が重要

大規模ストレージ

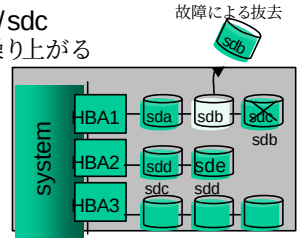
- 2TB超デバイス
  - 2.4での課題: 1TBが限界

1セクタ=512B、512Bx2^31=1TB

    - ブロックデバイスのセクタアドレッシングが31ビット
    - SCSIコマンドが2TBを超えたボリュームに未対応
    - カーネルパッチは存在するが、きちんと実運用を行うためには、コマンド等も含めてシステムワイドな対応が必要
  - 2.6での拡張、改善
    - セクタアドレッシングは64ビット化された
    - SCSIコマンドのサポートも強化された(16バイトコマンドサポート)
    - 制限がないわけではない
      - LVM
        - *displayの表示がおかしい（実運用では無視できない）
      - md
        - 構成ボリュームサイズ<2TB
          （今はあまり気にならないが、そのうち修正が必要かもしれない）

一貫したデバイス命名

- 2.4までの課題
  - 伝統的なデバイス命名: /dev/sda, /dev/sdb, /dev/sdc
    - よく知られた問題: sdbが壊れると、sdc が sdb に繰り上がる
    - ラベルマウントで解決（本当？）
      - mount -L HOME /home
      - ラベルに対応したファイルシステムでないと使えない
      - 同じラベルを持ったデバイスを接続すると破綻する
  - 2.4カーネルでの対処としてはdevfs
    - /dev/scsi/host0/bus0/target1/lun0/part1形式
    - シンプルでわかり易い仕組みだったが、udev/sysfsに統一する形で書き直し（dcache周りなどバグも多かった）

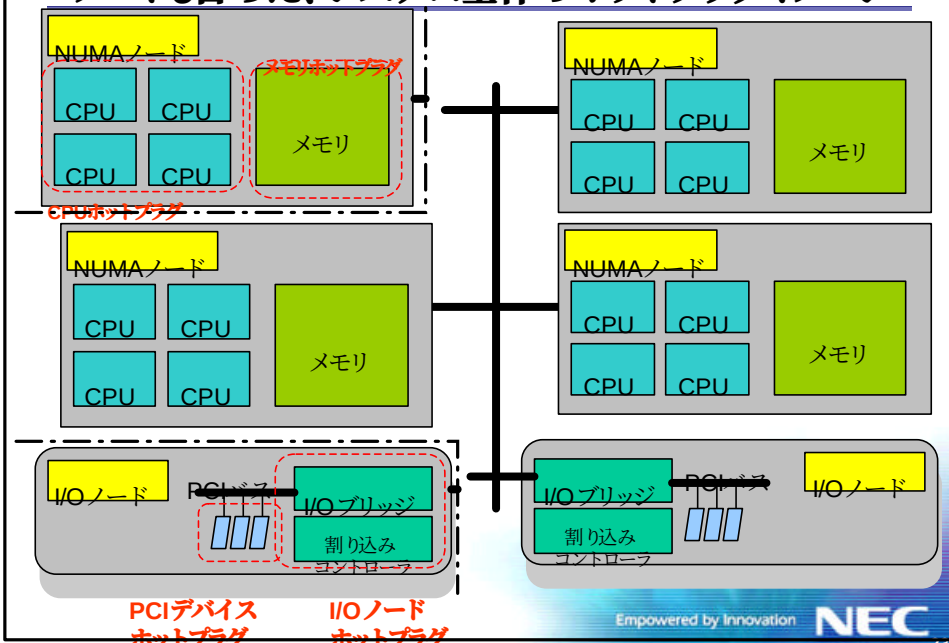


- 2.6カーネルでの改善:udev/sysfs
  - sysfsがシステムの状態、構成をユーザランドに表示
  - udevがそれを見て、適切なデバイスファイルを作成
    - ホットプラグと同一の枠組み
  - 名前付け規約が重要
    - すべての利用法に対応できるような規約はありそうにない

ホットプラグ（ノードホットプラグ）

- ノードの動的追加/動的切り離し: 2.6で現在開発中
  - ノードの構成要素すべてについての活線挿抜が必要
    - CPU、メモリ（NUMAノード）
    - PCIカード、PCIバス、I/Oブリッジ、割り込みコントローラ（I/Oノード）
- 要素技術の集積
  - PCIデバイスのホットプラグ、CPUのホットプラグ、メモリのホットプラグ
  - PCIルートブリッジ、PCI-to-PCIブリッジのホットプラグ
  - ノード上のデバイスの取り扱い(コンテナデバイス)
    - 「抜去するノードに含まれるデバイス」といった概念
  - デバイス間の依存関係の配慮
    - 「PCIデバイスAは、割り込みコントローラBに依存している」
  - ACPI (Advanced Configuration and Power Interface)
    - 接続トポロジ情報・各種デバイスの状態の取得
    - ホットプラグイベントのハンドリング

ノードも含めた、システム全体のホットプラグイメージ



障害対応（クラッシュダンプ）

- LKCD（Linux Kernel Crash Dump）
  - 割り込みや例外を起きないようにしておかないと結構採取に失敗する
    - LKCDは割り込みを空けたままダンプを取る（）
  - 動いていたドライバをそのまま再利用してダンプ（）
    - ロック期間中の障害やI/O中の障害で採取できない
      ⇒一番取りたい時だが、そういう時にかぎって取れない
  - 大規模システムへの対応の観点では:
    - ダンプ採取量、採取時間
      - 32GBメモリ実装のシステムで1時間近くかかったことも。障害データの転送が困難
      - ページキャッシュやユーザページ等を無視する仕組み
      - Linuxカーネルの重要な構造はメモリの先頭には集約されていない
        - 頭だけ取れてもだめなことが多く、全部取れないと苦しい
- 2.6に向けていくつかのプロジェクトが活動中
  - kexecベースのクラッシュダンプ
    - メモリを残したまま別カーネルで起動、そこからメモリダンプを採取
    - PCIバス状態等、リセットなしで本当に回復できるか？
  - netdump
    - ネットワーク経由のダンプ
      ネットワークドライバの方がつくりが単純なので再利用しやすいが...
    - ディスクに拡張したdiskdumpもある

おわりに

- 2.4から2.6に掛けて、Linuxカーネルは大規模サーバ対応という文脈で大きく進化
- 2.6はすでに完成の域に達しているか？
  - ホットプラグや障害解析など、新しい分野での強化、改善が引き続き必要
  - 新しいハードウェアや新しい使い方
    - 規模の拡大や縮小（組み込みハードウェアなど）
    - 新しいコンポーネント、新しいデバイス
  - 細かい不具合や問題意識は次々と生まれてくる
- コミュニティやディストリビュータと連携して、地道に解決していくことも重要