Vol.2013-OS-124 No.14
2013/3/1

高速な復帰が可能なLinuxのスリープ機構の実現

羽田野 大理[1,a)]　岩崎 英哉[1,b)]　鵜川 始陽[1,c)]

概要：近年データセンタでは，サーバの消費電力の増大が問題となっている．サーバの消費電力を抑えるには，例えばWebサーバではリクエストを処理し終えたら次のリクエストが到着するまでサーバをスリープさせるという手法が考えられる．しかし，現状のLinuxのスリープ機構はACPIを利用しており，スリープからの復帰に多くの時間がかかるという問題点がある．本研究では，ACPIを利用しないスリープ機構を作成した．このスリープ機構は，システム全体はスリープさせず，一部のデバイスのみをスリープさせ，CPUを簡易低電力状態にする．この方法で，Linuxのスリープを用いた場合と比較して，スリープからの復帰時間を大幅に削減できる事を確認した．

1. はじめに

近年，データセンタにおける電力使用量の増大が問題となっている．アメリカのデータセンタでは，年間1兆kWhの電力が使われている [1]．しかしながら，サーバではアイドル状態の時にも最大使用電力の60%もの電力が消費されている [2]．このような状況を解決するため，いままで様々な電力削減手法が提案されてきた．

DavidらはPowerNap [3] というサーバ向け電力削減手法を提案した．PowerNapでは，サーバはリクエストを待っている時はスリープ状態を維持する．そしてリクエストが到着したら，スリープ状態から復帰し，サービスを行う．そして，レスポンスを返したらスリープ状態へ戻り，リクエストを待つ．この場合，スリープ状態からの復帰にかかる時間がサーバの性能に大きく影響する．よって，その時間を高速化することが必要である．

しかし，現状のLinuxのスリープ機構はACPIを利用しており，ネットワークインタフェースカード（NIC）のスリープ状態からの復帰に7秒程度と、非常に長い時間がかかる事がわかった．そして，それを解決するにはACPIに頼らないスリープ機構が必要である．

本研究の目的は，PowerNapで利用する高速な復帰が可能なスリープ状態の実現を図る手法を調査し提案する事である．それにより，PowerNapの有効性や課題などを示すことができる．

[1] 電気通信大学
[a)] hiro@ipl.cs.uec.ac.jp
[b)] iwasaki@cs.uec.ac.jp
[c)] ugawa@cs.uec.ac.jp

2. 既存の電力削減手法

本章では既存の電力削減手法について，Dynamic Voltage and Frequency Scaling（DVFS）及び，動的なサーバ統合の2つの手法を紹介する．

2.1 Dynamic Voltage and Frequency Scaling

DVFSは，サーバの使用率が低い状態において，主にCPUの電力を削減する電力削減手法である．

近年のサーバ向けCPUには，CPUの動作周波数及び動作電圧を動作中に変更できる機構が取り入れられている．例えば，Intelの Speedstep [4] や AMDの cool'n'Quiet [5] である．オペレーティングシステム（OS）がこの機構を利用し，サーバのCPU使用率が低い時に，CPUの周波数を下げる事で，サーバの電力を削減する．

しかし，近年のサーバにおいて，サーバ全体に対してCPUが占める電力割合は低くなってきている．図1は，IBM p670 [6] 及び Sun UltraSparc T2000 [7]，また Googleの汎用サーバ [8] のサーバの各コンポーネントが占める電力割合を示している．これによると，CPUが占める電力割合は概ね25%，高々40%程度である．仮にDVFSによりアイドル時の電力を完全に削減できたとしても，サーバ全体の高々40%程度しか電力を削減できず，大きな効果は得られないと考えられる．

2.2 動的なサーバ統合

動的なサーバ統合 [8] は，仮想マシンモニタを有効活用し，動作中のサーバ数を調整することで，サーバの使用率を向上させ，電力を削減する手法である．

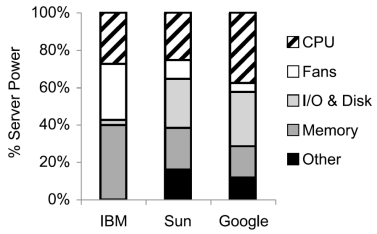


図 1 サーバの電力消費内訳 （ [3] より引用）

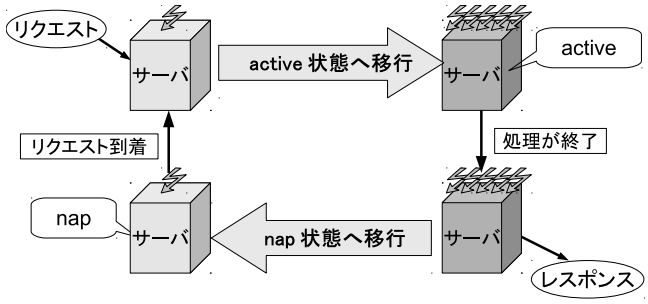


図 2 PowerNap の動作概要

サーバアプリケーションを，直接サーバ上で動いている OS ではなく，仮想マシン内の OS 上で動作させ，仮想マシン上からサービスを提供する．そして，サーバ使用率が低くなってきたら，仮想マシンモニタの機能である仮想マシンのライブマイグレーション [9] を行い，多数のサーバで動いていた仮想マシン群を少数のサーバに集約する．ここで，仮想マシンのライブマイグレーションとは，実行中の仮想マシンを止めることなく，別マシンの仮想マシンモニタへ仮想マシンを移動する機能である．これによりサービスを中断せずに物理サーバ間を移動することができる．

ライブマイグレーションにより仮想マシンが存在しなくなったサーバをシャットダウンする．これにより，必要最小限のサーバのみを動作させる事ができ，不必要な電力を削減する事ができる．

しかし，この動的なサーバ統合にはいくつかの問題点がある．まず，構成の変化に数十分程度時間がかかってしまう事が挙げられる [3]．それに対してサーバに対する負荷は突発的に大きく変化する事があり，この手法では負荷の変化に対応しきれない．

また別の問題点として，サーバアプリケーションによってはサーバ数を減らす事が適さない場合がある．例えば，Web Search [10] や memcached [11] 等の Web 検索サーバアプリケーションでは，高速化のために仕事をサーバ全体に分散させている．この場合には，サーバ数を減らしてしまうと，遅延時間が大きくなり，許容遅延時間を超えてしまう可能性がある．

また他に，サーバを集約する事による信頼性の低下などの問題も考えられる．よって現状のサーバにこの電力削減手法を適用するには解決すべき問題点が多い．

3. PowerNap

前章で述べた電力削減手法は，現状のサーバに適用するには問題点が多い手法であった．本章では，より現状のサーバに適合する手法である，PowerNap [3] について紹介する．

3.1 概要

PowerNap はサーバ向けの電力削減手法であり，低電力状態と完全稼働状態の 2 状態を高速に切り替える事で動作する．PowerNap では，高速な移行及び復帰ができる低電力状態を nap モードと呼び，完全稼働状態を active モードと呼ぶ．nap モードでは，CPU やデバイスも省電力状態となり，一般的なシステムのスリープ状態と同様に，サーバとしての処理を行うことはできない．active モードは，システムの全てのコンポーネントが高速で完全に稼働していて，最も早く仕事を処理できる状態である．

図 2 に PowerNap の動作の概要を示す．サーバが何らかの仕事を待っている間，サーバは nap モードとなっている．リクエストがサーバへ到着したら，サーバは nap モードから復帰してこのリクエストを処理する．処理が終わりレスポンスを返したら，また nap モードへ移行し次のリクエストを待つ．これが PowerNap におけるサイクルとなる．そのため，nap モード時に多くの電力を削減し，かつ nap モードと active モード間の遷移時間を極力短くする事が重要となる．

nap モードからの復帰には，WakeOnLan（WOL）を用いる．WOL とは，コンピュータがスリープ状態にあったり，電源が切れていても，あるパケットがネットワークインタフェースカード（NIC）に来たらシステムに割込みを入れ，スリープ状態から復帰したり電源を ON にしたりする機能である．NIC がどのパケットに反応するようにするかは設定することができ，マジックパケットと呼ばれる WOL 専用のパケットや，その NIC へのユニキャストパケット，ARP パケット，そして NIC に到着する全てのパケットなどを選べる．PowerNap では，全てのパケットや，ユニキャストパケットで WOL 機能が動作するように設定する事で，リクエストが来たら active モードへ復帰しレスポンスを返す．

3.2 モデルによる検証

David らは，PowerNap による電力削減手法を数式によりモデル化し，DVFS と比較し電力削減及びレスポンス時間のオーバヘッドの効果を検証した．そのグラフを図 3 に示す．

図 3 (a) では，nap モード時の電力が通常時の 5%，DVFS による CPU の動作下限周波数を通常時の 20%とした時の

最大電力消費量に対する同量を表したグラフである．そして，グラフ中 F_{cpu} はサーバ全体に対して CPU が占める電力割合を示し，T_t は，nap モード及び active モード間の遷移時間を示している．David らは，nap モードへの移行及びそれからの復帰各々にかかる時間をおよそ 10 ms より短くすれば，DVFS より多く電力削減が行える，と述べている．

図 3（b）は，レスポンスのオーバヘッドを示している．これによると，10 ms で許容範囲内，そして 1 ms までモード遷移にかかる時間を減らすことが出来れば，PowerNap によるオーバヘッドはほぼ無視できるが，100 ms またはそれ以上遅くなると，パフォーマンスに大きな影響がでる，と述べている．

David らは，PowerNap を提案し，モデルやシミュレーションを用いて実験を行ったが，実装は行なっていない．

3.3 Linux 向け PowerNap

Linux には Dustin らによる PowerNap というプログラム [12] が存在する．以降，Dustin らによる PowerNap を PowerNap デーモンと呼ぶ．これは Python で書かれたスクリプトであり，ユーザ空間で監視の役割を果たすプログラムである．

PowerNap デーモンは，様々なアクティビティの監視を行い，指定された期間以上監視対象の利用がなかった場合には，任意のコマンドやアクションを実行することができる．アクティビティにはたとえば，コンソールの利用，プロセスの存在有無，CPU 使用率，ネットワークコネクションなどを指定できる．サーバ向けの電力削減機構として利用する場合は，たとえば HTTP サーバの場合，サーバの TCP ポート 80 番のコネクションを監視させ，HTTP コネクションがない場合にシステムをスリープ状態へ移行させる，といった設定にする．

3.4 PowerNap デーモンを用いた予備実験

PowerNap デーモンは Linux 既存のスリープ機構を用いている．そこで PowerNap デーモンを動作させる事によるパフォーマンスへの影響を測定する実験を行った．実験環境は，表 1 の通りである．

この実験では，簡易 Web サーバに PowerNap デーモンを導入した場合としていない場合において，クライアントで 10 秒おきに `wget` コマンドを実行して Web ページを取得するのにかかった時間を 100 回測定し，平均値を求めた．サーバはユニキャストパケットにより WOL でスリープ状態から復帰する設定にした．また，5 秒間コネクションがなければスリープ状態に移行する設定としたため，リクエスト時にはサーバはスリープ状態となっており，WOL によりスリープ状態から復帰する．よってレスポンス時間には，スリープの復帰にかかる時間が含まれる．

結果を表 2 に示す．結果において，PowerNap を導入した方は 7 秒程度遅くなっている．7 秒という時間は David らが推奨する 10ms と比べると非常に遅い．

次に，この 7 秒という時間は何に費やされているのか調査した．`tcpdump` コマンドによりネットワーク上のパケットを監視したところ，HTTP リクエストにおいて 1 回目のクライアントからのパケットは無視され，数秒後の 2 回目のクライアントからのパケットに対し，サーバからのレスポンスが返されていた．この事より，NIC のスリープ状態からの復帰中のパケットは捨てられており，クライアントのリクエスト再送によりレスポンスを返していると言う事がわかる．

また，リクエストのタイミングによっては，ずっとレスポンスを得られない事があった．これは，サーバがスリープ状態から復帰してもパケットは捨てられており，コネクションを確立できていないため，サーバは現在リクエストがないと認識し，クライアントのリクエスト再送までの間に再度スリープ状態へ移行してしまうためである．クライアントは通信に失敗するとリクエストの再送時間が長く設定されるため，この様な事が発生すると，リクエストに対するレスポンスが得られなくなるという状況が発生する．

ここでは予備実験のため，PowerNap デーモンの監視間隔時間の設定を 5 秒と長めにしたが，実際のサーバでは短くする方が電力削減量を多くできるため，本来はより短く設定されるべきものである．そのため，この時間を短くすれば，上のようなレスポンスを得られなくなるという状況が発生しやすくなると考えられる．

3.5 問題の解決案

前説で述べた問題を回避するために，幾つかの手法が考えられる．

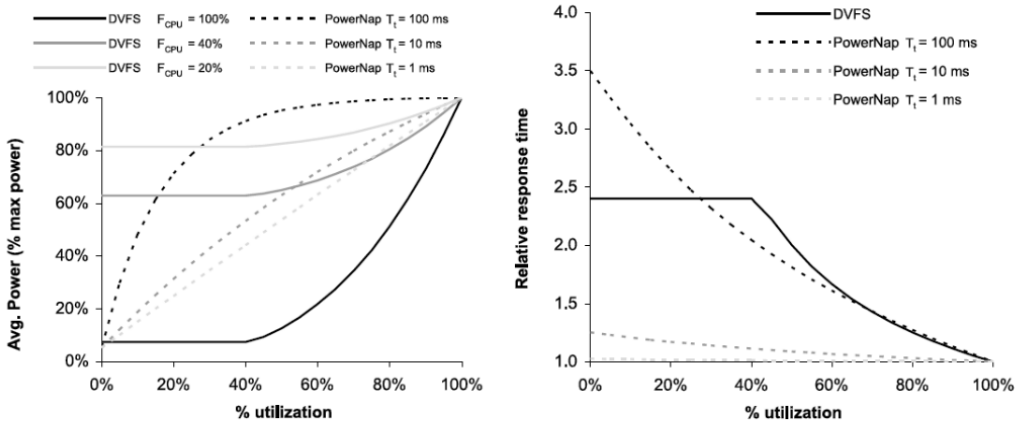
第一に，NIC ハードウェアが WOL 時のパケットを保存しておく，という手法である．これが実現できれば，NIC がスリープ状態から復帰後，パケットを失わずシステムに

表 1 予備実験環境

	Server	Client
CPU	4Core4Thread 2.4GHz	4Core4Thread 3.1GHz
Memory	3.2GB	3.7GB
Chipset	Intel G965	Intel H61
Storage	SSD 120GB	HDD 500GB
OS	Ubuntu 12.04LTS	Ubuntu 12.04LTS
Kernel	Linux 3.5.3	Linux 3.2.0-35
Network	1000Base-T	1000Base-T

表 2 レスポンス時間測定予備実験

PowerNap の有無	時間（ms）
有り	7013
無し	3



(a) Power Scaling (b) Response Time Scaling

図 3 PowerNap と DVFS の電力削減量比較 [3] より引用

配送することができる．しかしながら，PCI デバイスにおいては，NIC がスリープ状態から復旧する時，NIC のリセットが行われ，レジスタが初期化される．そのため，この手法の実現は難しい．

第二に，NIC がスリープ状態から復帰するまでリクエストをロードバランサ等でプールさせておくという方法も考えられる．まず，ロードバランサがサーバに対するリクエストを受け取ったら，サーバに対し適当なパケットを送信し，WOL を発生させ，スリープ状態から復帰させる．そして，NIC がスリープ状態から復帰したら目的のパケットを送信サーバに送信する，という手順である．しかし，サーバの NIC がスリープ状態から復帰した瞬間を捉えるのは難しく，またスリープ状態から復帰するまで別のサーバを用意し，リクエストパケットをプールする必要があり，それにかかる電力等を考えると現実的ではない．

第三に，NIC をスリープ状態へ移行させない，といった方針が考えられる．しかし，Linux ではスリープ状態への移行処理の最後で，ACPI のシステム全体をスリープ状態へ移行させる機能を利用する．すると，各デバイスの主電源は遮断され，NIC 等一部のデバイスに対して補助電源が供給されるのみとなる．よって，ACPI を利用しては簡単には NIC をスリープ状態へ移行させずに，システムをスリープ状態へ移行するといった事はできない．

我々は，第三の方法を改良した案を考えた．改良案では，ACPI によるスリープ状態への移行機能を用いず，擬似スリープと呼ぶ手法を提案する．詳細は 5 章で説明する．

4. Linux におけるスリープ機構

本章では，Linux におけるスリープ機構の利用方法や，その処理の流れ及びそれにかかる時間について説明する．

4.1 ユーザ空間とのインタフェース

Linux においては，電力管理情報やデバイス情報などを取得するのに，sysfs ファイルシステムが使われる．sysfs とは，procfs と同じ，Linux カーネルが提供する仮想ファイルシステムであり，この中のファイルを読み書きする事で，カーネル内部の情報にアクセスできる．この sysfs は `/sys/`ディレクトリにマウントされている．

システム全体の電力管理に関するファイルは，`/sys/power/` ディレクトリに配置されており，`/sys/power/state` ファイルに特定の文字列を書き込むことで，スリープ状態に移行することができる．“`mem`” とファイルに書込めばスリープ（Suspend to RAM）が実行され，“`disk`” と書き込めばハイバネーション（Suspend to Disk）が実行される．

Linux で認識されているデバイスに関しても，デバイス毎に情報を取得できるディレクトリ（デバイスディレクトリ）がある．sysfs 直下には bus や，module 等が配置されているディレクトリがあり，bus や module から辿っていくことで各デバイスディレクトリにアクセスできる．

4.2 スリープ状態への移行処理

ここでは，`/sys/power/state` ファイルに `mem` と書き込んだあと，システムがスリープ状態に移行するまでの過程を示す．

(1) `sync` システムコールの呼び出し
(2) プロセスの停止
(3) デバイスをスリープ状態へ移行
(4) 起動用 CPU 以外の CPU の停止
(5) 割込みの無効化
(6) 割込みコントローラ等のシステム中核デバイスのスリープ状態への移行
(7) ACPI による全システムのスリープ

また，スリープ状態から復帰する時の処理は以下のようになる．

(1) ACPI によるシステムの復帰

(2) システム中核デバイスをスリープ状態から復帰
(3) 割込みの有効化
(4) 全ての CPU の有効化
(5) デバイスのスリープ状態からの復帰
(6) プロセスの再開

これらの各処理にかかる時間を，printk 関数を用いカーネルログを出力し、それに埋め込まれるタイムスタンプを用いて調査した．カーネルログは、タイムスタンプとメッセージの組で出力され、dmesg コマンドを通じて取得できる。このカーネルログにはタイムスタンプが小数点以下第 6 位まで記録されている．x86 マシンにおいては，タイムスタンプに CPU の持つタイムスタンプカウンタ（TSC）を用いられており，精度が高い．また、このカーネルログはメモリ上に保存されているため、殆どのデバイスが停止した状態でも利用可能である。

スリープ状態への移行時にかかった時間の調査の結果を表 3 に，スリープ状態からの復帰時の調査結果を表 4 に示す．中でも最も時間がかかっていたのがデバイスをスリープ状態から復帰させる処理である．

5. 擬似スリープ機構の設計と実装

3 章では，PowerNap を実現するには NIC をスリープ状態へ移行させる事に問題がある，という事を述べた．本章では，NIC をスリープ状態へ移行する事なく，システム全体をスリープに近い状態に移行させる機構についての，設計及び実装を述べる．以後，この機構を擬似スリープ機構と呼ぶ．

擬似スリープ機構は，以下の 3 つの要素からなる．これを，既存の Linux カーネルのソースコードを変更することで実装した．

まず 1 つ目の要素は，ユーザが指定したデバイスだけスリープ状態へ移行させる機構である．これを選択的デバイススリープと呼ぶ．Linux のスリープ機構において，デバイスのスリープ状態への移行処理および NIC のコネクション復帰に多くの時間がかかっていた．よって，NIC など時間のかかるデバイスはスリープ状態へ移行させないようにするなど，デバイス毎にスリープ状態へ移行させるデバイスを選択できるようにする事で，ユーザに遷移時間と電力削減量のトレードオフを選べるようにする．

表 3 スリープ状態へ移行時の詳細

処理	時間（ms）
sync システムコールの呼び出し	3
タスクの停止	35
デバイスをスリープ状態へ移行	307
起動用 CPU 以外の CPU の停止	311
割込みの無効化	0
システム中核デバイスをスリープ状態へ移行	0

表 4 スリープ状態から復帰時の詳細

処理	時間（ms）
システム中核デバイスをスリープ状態から復帰	0
割込みを有効化	0
全ての CPU を有効化	48
デバイスをスリープ状態から復帰	469
タスクの再開	0

2 つ目は，ACPI によるシステムスリープへの遷移機能を使わずに，CPU をスリープ状態へ移行させる事で，擬似的なシステムスリープ状態を実現する．これを CPU HALT 状態と呼ぶ．CPU HALT 状態は，CPU を低電力状態にする命令を実行することで実現している．

3 つ目に，ユーザが設定したデバイスからの割込みによりスリープ状態から復帰する機構である．NIC からの IRQ を捕捉し，擬似スリープ状態から復帰する事で，WOL を有効にしたスリープ状態と，同様の効果を得る事ができる．但し，IRQ は複数のデバイスから共有される事があるため，必ずしも特定のデバイスを一意に識別できるとは限らず，他のデバイスからの割込みによってスリープ状態から復帰してしまう可能性がある．しかし近年の PCI デバイスでは Message Signaled Interrupts（MSI）という仕組みがあり，多くのデバイスを別々の IRQ へ割振ることができるようになったため，共有される事は稀である．Linux 既存のスリープ機構や WOL を用いた場合，リクエストを取り逃してしまう事があった．よって，割込みの状態を本機構で厳しく管理する事で，リクエストの捕捉漏れがないように設計する．

5.1 選択的デバイススリープ

各デバイスに対応する device 構造体に，そのデバイスをスリープ状態へ移行させるかどうかのフラグを追加した．そして，擬似スリープ移行に際しては，このフラグを確認し，指定されたデバイスのコールバック関数のみを呼出すようにし実現する．

スリープ状態に移行させるデバイスは，4.1 節にて述べたデバイスディレクトリに新しく配置したファイルに対して書込みを行うことにより選択する．

5.2 CPU HALT 状態

x86CPU には，割込みがあるまで命令実行を止め，CPU コアを低電力状態へ移行する HLT 命令がある．この命令を利用し，CPU HALT 状態としている．

HLT 命令は，低電力状態への移行及び低電力状態からの復帰が早く，キャッシュやレジスタなどのデータも一切失われない．予備実験（表 3，表 4）での，Linux 既存のスリープ状態への移行処理において，CPU の停止にはスリープ移行時に 307 ms，レジューム時に 48 ms かかっていた．

HLT 命令を使えば，この時間を大幅に短縮する事ができる．

ただし，この HLT 命令は論理コアを HALT 状態へ移行させるが，電力を削減する場合は物理コアに属する全ての論理コアで HLT 命令を実行しなければならない [13]．よって，たとえば Intel のハイパースレッディング機能を搭載した CPU でハイパースレッディングをオンにしている場合は，1 コアを HALT 状態にしただけでは電力削減効果がない．よって全ての論理コアで HLT 命令を実行させるため，任意の関数を全ての CPU に強制的に実行させる Linux の機能を用いる事により実現した．この機能は CPU 間割込み（IPI）を用い実現されており，カーネル空間でのみ使うことができる．

しかし，IPI により関数を実行させると，割込みが禁止される．HLT 命令は割込みがあるまで命令実行を停止するため，そのまま呼び出すとそのコアは停止した状態から復帰できなくなる．そこで，まず IPI により呼び出した関数の最初で STI 命令 [13] を実行させた．これは，CPU のフラグの 1 つである，割込み許可フラグをセットする x86 CPU の命令である．これにより，禁止された割込みを有効化し，これを解決した．

HLT 命令が解除されるのタイミングは割込みの発生である．しかし，デバイスから配送される割込みは，どれか 1 つの論理コアに対してだけであり，1 つの論理コアしか HLT 命令から復帰しない．CPU HALT 状態から全てのコアを復帰させるために，プロセスリスケジューリングを促すフラグをセットするだけの IPI を用い，全コアの復帰を実現した．

また単に HLT 命令を実行しただけでは，デバイスからの割込みにより CPU が HALT 状態から復帰してしまう．これは，次節で述べる，指定した割込みによるスリープ復帰を用いる事で，解決した．

5.3 指定した IRQ 番号によるスリープ復帰

HLT 命令を繰り返し実行する事で，NIC 以外のデバイスからの割込みにるスリープからの復帰を防ぐようにした．NIC からの割込み検知は，どの IRQ 番号から割込みが来たかを記録し，ユーザから指定のあった IRQ から割込みであったら HLT 命令の繰り返しを抜けるようにした．これによる CPU HALT 状態への移行処理は以下のようになる．

(1) システムスリープ状態への移行指示が出される
(2) IRQ 記録用変数の初期化
(3) スリープ移行処理
(4) ユーザから指定された IRQ が発行されている場合は（10）の処理に移る
(5) HLT 命令の実行
(6) デバイスが CPU に対して IRQ を発行
(7) IRQ 番号を記録
(8) 当該の IRQ ハンドラの実行
(9) （4）の処理に戻る
(10) スリープ状態からの復帰処理

IRQ 記録用変数の初期化は上記の通り，スリープ状態への移行処理の冒頭で行う．これにより，スリープ状態への移行処理中にリクエストが来ても，HLT 命令は実行されずにスリープ状態からの復帰処理が行われ，レスポンスを返すことができる．

IRQ 番号の記録は，IRQ が発行された時の統計情報更新処理の中で行なっている．

ユーザには，IRQ 番号を指定してもらう必要がある．このインタフェースとして，sysfs 内に新しくファイルを追加し，そのファイルに IRQ 番号を書き込むことで指定できるようにした．IRQ 番号は，/proc/interrupts に情報があり，eth1 等の NIC のエイリアス名から目的の NIC の IRQ 番号を知ることができる．

上のようにして，NIC のパケットの捕捉もれを殆どなくす事ができるが，まだ完全ではない．スリープ状態への移行処理の開始時の詳細は，以下の様な動作になる．

(1) サーバが処理を終え，クライアントへレスポンス送信
(2) PowerNap デーモンが，コネクションがない事を確認
(3) PowerNap デーモンが，/sys/power/state へ書込み
(4) sysfs ファイルシステムが，スリープ関数の呼び出し
(5) IRQ 記録用変数を初期化
(6) スリープ状態への移行処理

上記動作のスリープ状態への移行処理の 2 から 5 の間に NIC に処理が到着したら，捕捉漏れが起きてしまう可能性がある．これは，PowerNap デーモンがスリープ状態への移行指示を出してから，IRQ 記録用変数の初期化を行なっている事が原因である．

そこで，IRQ 記録用変数の初期化処理を PowerNap デーモンに行わせるよう，IRQ 記録用変数の初期化するインタフェースの作成及び，PowerNap デーモンの処理を書き換える方法が考えられる．その場合の処理は以下のようになる．

(1) サーバが処理を終え，クライアントへレスポンス
(2) **PowerNap デーモンが，IRQ 記録用変数の初期化**
(3) PowerNap デーモンが，コネクションがない事を確認
(4) PowerNap デーモンが，/sys/power/state へ書込み
(5) sysfs ファイルシステムが，スリープ関数の呼び出し
(6) スリープ状態への移行処理

このようにする事で，完全にリクエストの捕捉漏れを防ぐことができる．

ただし、この NIC の割込みによるスリープ復帰では，ブロードキャストパケットや他ホストへのパケット等全てのネットワーク通信によってスリープ状態から復帰してしまう問題がある．

これを回避するには，ネットワーク上にサーバに無関係なパケットを極力流さないようにする事が挙げられる．例

えば，直接サービスを提供するサーバはネットワークを隔離する，ARP パケットや DHCP のパケットを利用しないように設定する，などの対策が考えられる．

6. 評価

本章では，レスポンス時間及び，電力削減量の観点から擬似スリープ機構の評価を行う．

測定に用いた実験機の構成は，表 5 のとおりである．そして，ネットワークに余計なノイズが混入しないよう，サーバ及びクライアントに対し，以下の設定を施した．

- ネットワークはサーバとクライアントを直結する
- スタティック IP アドレスを振り，DHCP の利用を禁止する
- arp テーブルにサーバ・クライアント互いの IP アドレスと MAC アドレスを追加しておく
- DNS サービスの利用を禁止する

6.1 レスポンス時間の測定

この実験では，以下の 3 種類の設定を施したサーバに対して，クライアントが `wget` コマンドを用いて Web ページを取得し，そのレスポンスにかかった時間を 100 回測定し平均値を求めた．

通常時 PowerNap デーモンを導入しなかった場合

Linux 既存 PowerNap デーモンを導入し Linux 既存のスリープ機構を用いた場合

擬似スリープ PowerNap デーモンを導入し擬似スリープ機構を導入した場合

PowerNap デーモンは，TCP ポート 80 番に 3 秒間コネクションがなければスリープ状態に移行するようにし，`wget` は 8 秒おきに実行するようにした．よってクライアントが `wget` を行った時には，サーバは必ずスリープ状態へ移行している．Linux 既存のスリープ状態時において，NIC はユニキャストを受け取ると WOL が働くように設定した．擬似スリープ機構に関して，選択的デバイススリープでは NIC 及びシリアルコンソールをスリープ状態へ移行しないよう設定した．また，割込みは NIC 及びシリアルコンソールの IRQ 番号を設定した．但し，シリアルコンソールに関してはデバッグ目的であり，実験中にはシリアルコンソールから割込みが発生しないようにした．

表 5 実験環境

	Server	Client
CPU	4Core4Thread 2.4GHz	4Core4Thread 3.1GHz
Memory	3.2GB	3.7GB
Chipset	Intel G965	Intel H61
Storage	SSD 120GB	HDD 500GB
OS	Ubuntu 12.04LTS	Ubuntu 12.04LTS
Kernel	Linux 3.5.3	Linux 3.2.0-35
Network	1000Base-T	1000Base-T

測定結果を表 6 に示す．擬似スリープ機構においては，433 ms で，通常時の 3 ms と差分を取るとオーバヘッドは 440 ms となり，David らが推薦する 10 ms までは遠い．しかし，Linux 既存のスリープ機構の 7010 ms と比べると，6570 ms 高速化されており，十分に高速化されたと考えられる．

6.2 電力削減量の測定

電力削減量を評価するために前節の 3 通り n おサーバに対して消費電力を測定した．電力の測定には富士通コンポーネント製スマート電源コンセント [14] を用いた．このスマート電源コンセントは USB の口がついた電源タップであり，USB を PC に繋ぎ，専用のアプリケーションを通じて，電力を表示，記録することができる．

ここでは，上と同じ条件のサーバに対して以下の 3 種類について 3 分間電力測定を行い，その平均値を測定した．

結果を表 7 に示す．この結果から，表 5 のサーバでは，電力を殆ど削減できなかった事が分かった．

7. 電力削減に関する考察

実験において，擬似スリープ機構は，電力を削減することはできなかった．本章では，この点について考察する．

7.1 CPU の電力管理

CPU に，`HLT` 命令を実行させる事で電力削減を図ったが，結果的には変化しなかった．これは，Linux にはアイドルプロセスというプロセスがあり，CPU 使用率が低い時には，スケジューラからアイドルプロセスが選出される．そして，このアイドルプロセスは，CPU を低電力状態へ移行させている．

よって，アイドル時と擬似スリープ時はほぼ同じ事をしているため，結果的に CPU の電力を削減する，という結果には至らなかったと考えられる．

7.2 PCI バスの電力管理

現在，多くのデバイスは，PCI Express バスや PCI バス上で動作している．よって，PCI バスには独立した電力管

表 6 レスポンス時間測定

サーバ設定	時間（ms）
通常	3
Linux 既存	7013
擬似スリープ	443

表 7 電力測定

サーバ設定	電力（W）
通常	82.0
Linux 既存	4.0
擬似スリープ	81.3

表 8 PCI D 状態

D 状態	サポート	電源状態
D0	必須	通常
D1	任意	省電力
D2	任意	省電力
D3hot	必須	省電力
D3cold	必須	電源断または補助電源のみ

理機能があり，各々の PCI デバイスを低電力状態へ移行できれば，電力を削減できるはずである．

PCI バスには ACPI と同じように，個々のデバイスの電力状態である D 状態と呼ばれる状態を定めている．D 状態を，表 8 に示す．D0 状態は動作中であり，それ以外の状態は動作中ではない．従って，サーバの動作に復帰するためには，D0 状態に復帰する必要がある．D0 以外の各状態の違いは，D0 状態への復帰時間である．D0 状態へ復帰するまでにかかる時間は，Dx の数値が大きくなる程長くなり，D3cold が一番長くかかる．逆に電力使用量は，D3cold が一番少なく，Dx の数値が小さくなるほど電力使用料は大きくなる．

PCI デバイスにおいて，Linux のデバイスをスリープさせる関数を呼び出すと，多くのデバイスの電源状態は D3hot 状態へ移行している．

従って，擬似スリープ状態においても，NIC 以外のデバイスは D3hot 状態へ移行しているはずである．しかし，電力は減っていなかったので，今回の実験において用いられたコンピュータのデバイスは，D3hot 状態と D0 状態の消費電力の差が小さいものであったと考えられる．

7.3 PCI D3cold 状態

D3hot 状態から，D3cold 状態へ移行させることが出来れば，PCI デバイスの電力を削減することができると思われる．実際 ACPI により，全システムをスリープ状態へ移行する処理において，PCI デバイスは D3hot 状態から D3cold 状態へ移行していると考えられる．

しかしながら，D3cold 状態へ移行するのは難しい．D0 状態から D3hot 状態までは，各デバイスの特定のレジスタに値を書き込めば移行できる．しかし，D3cold 状態にするには PCI バスに対して処理を行わせる必要があり，さらに PCI バスに属している全てのデバイスが D3hot 状態である必要がある．そして，擬似スリープ状態では，NIC をスリープ状態へ移行させずに，D0 状態のままにしておくという方針を採用した．従って，擬似スリープ状態では，D3cold 状態は利用できず，PCI デバイスにおいてこれ以上の電力削減は望めないと考えられる．

8. おわりに

NIC のスリープ状態からの回復には遅延やパケット破棄の問題があり，ソフトウェアの面だけから解決することは難し事を示し，擬似スリープ機構を提案した．擬似スリープ機構では ACPI のスリープ移行機能を利用せず，各デバイス及び CPU をスリープ状態へ移行する事により機能を実装した．本機構を利用する事により電力はまだ一部のデバイスでしか削減できないが，遅延を大幅に削減できる事が確認できた．

今後の課題として，個々のデバイスの電力を削減する有効な手段がないか探っていきたい．また，レスポンスにかかる時間もより短くなるよう改良案を摸索していきたい．

参考文献

[1] Report to congress on server and data center energy efficiency Technical report.
[2] Barroso L. A. and Holzle U. The case for energy-proportional computing. In *Computer 40*
[3] Devid M., Brian T. G., Thomas F. W. PowerNap: Eliminating server idle power. In *ASPLOS '08: Proceedings of the 14th International Conference on Architectural Support for Programming Lanuages and Operating Systems*, 2008
[4] Intel Corporation Enhanced Intel SpeedStep Technology for the Intel Pentium M Processor `http://www.memcached.org`.
[5] AMD Cool'n'Quiet Technology `http://www.amd.com/jp/products/technologies/cool-n-quiet/Pages/cool-n-quiet.aspx`
[6] Lefurgy C., Rajamani K., Rawson F. Felter W., Kistler, M. and Keller T. Energy management for commercial servers. In *IEEE Computer 36, 12*, 2003
[7] Laudon J. UltraSPARC T1: A 32-thrreaded CMP for severs. Invited talk. 2006
[8] Fan X., Wever W. D., and Barroso L. A. Powerprovisioning for a warehouse-sized computer. In *ISCA '07: Proceedings of the 34th Annual International Symposium on Internet Technologies and Systems*, 2007
[9] Christopher C., Keir F., Steven H., Jacob G. H., Eric J., Christian L., lan P., Andrew W. Live Migration of Virtual Machines In *In Proceedings of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation*, 2005
[10] Barroso L. A., Dean J. and Holzle U. Web search for a planet: The google cluster architecture. In *IEEE Micro. 23*, 2003
[11] Memcached - a distributed memory object caching system. `http://www.memcached.org`.
[12] Dustin Kirkland powernap in Launchpad `https://launchpad.net/powernap`.
[13] Intel 64 and IA-32 Architectures Software Developer's Manual Volume 2 `http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html`
[14] 富士通コンポーネント スマート電源コンセント `http://www.fcl.fujitsu.com/services/smart-power-strip/`