計算機アーキテクチャ　147－9
ハイパフォーマンス　89－9
コンピューティング
（2002．3．7）

HPC 向け大規模クラスタシステムにおける省電力機能の実装

鈴木 和宏　勝野 昭　木村 康則†

HPC 向けクラスタシステム上で動作するアプリケーションは、通常最大性能を引き出すためにクラスタの最大ノード数で実行することが多いと考えられる。しかしながらアプリケーションによっては、必ずしもノード数を増やせば並列最高性能が出るとは限らない。この場合にはノード数を制限して実行することになる。大規模なクラスタシステムにおいて、アプリケーションを実行していないアイドル状態のノードが多数存在するのは消費電力から見て無駄が多い。そこでアイドル状態のノードは積極的に停止させて、必要なときに起動するような省電力クラスタシステムを実装した。

The Implementation of Power-Saving Faculty for Large Scale HPC Cluster System

Kazuhiro SUZUKI　Akira KATSUNO　Yasunori KIMURA†

This paper describes the power-saving faculty for large scale cluster system. In a large scale cluster system, we cannot keep using all nodes simultaneously because of the application characteristic. In this situation, we waste electric power. Therefore, we have been developed and implemented the power-saving faculty on SCore which had been developed by RWCP.

1　はじめに

従来のスーパーコンピュータに代って複数の安価な計算機 (ノード) をつないだクラスタシステムが開発されている。クラスタシステムによって単体では限界であった処理能力や信頼性を向上させることができる。複数のノードを一つのクラスタとして動作させるためのソフトウェアがクラスタシステムソフトウェアである。一般にクラスタシステムソフトウェアには以下のようなものがある。

- フェイルオーバ型
  2 台またはそれ以上のノードを動作させ、何らかの原因で動作不能になった場合にバックアップとして待機させていた他のノードがその処理を引き継ぐことによって HA(High Availability) を向上させる。フェイルオーバー型クラスタには SafeCLUSTER [1]、ClusterPerfect [2]、CLUS-

†(株) 富士通研究所
FUJITSU LABORATORIES LTD.

TERPRO [3] などの製品がある。

- ロードバランシング型
  WWW や FTP サーバなどのサーバを多重化して、スケーラビリティを実現するものである。ロードバランシング型クラスタには TurboLinux Cluster Server [4] や RedHat High Availability Server [5] 等がある。

- HPC 型
  複数のノードが協調動作することによって並列処理アプリケーションを高速に実行する。HPC 型クラスタには SCore [6] や Beowulf [7] 等が挙げられる。

SCore のような HPC 向けクラスタシステム上で動作するアプリケーションは、クラスタが持っている最大ノード数で実行することが多いと考えられる。しかしながらアプリケーションによっては最高性能が出るノード数が全ノード数よりも小さい場合もある。またクラスタ内のノードをいくつかのサブクラスタに分けて、それぞれのサブクラスタで複数のアプリケーションを起動するような場合は、アプリケーションの終了時間の違いからアイドル状態のノードができてしまう可能性がある。クラスタシステムが大規模になるにつれてアイドル状態のノードが多数存在するのは消費電力から見て無駄が多くなる。

本稿では省電力クラスタシステムを実現するための機能について説明する。さらに本機能を RWCP で開発された HPC 向け大規模クラスタシステムソフトウェアである SCore 上に実装し、ベンチマークプログラムを動作させて測定した消費電力について述べる。

2 省電力機能

2.1 動作イメージ

図 1に省電力機能の動作イメージを示す。基本的な動作はアイドル状態のノードは積極的に停止または省電力状態に移行 (サスペンド) させて、必要な時に起動 (リジューム) することによってクラスタシステム全体の消費電力を低減する。

図 1: 省電力クラスタの動作イメージ

2.2 サスペンド

一般的な PC には APM (Advanced Power Management) と呼ばれる電力管理機能が備わっている。APM は Microsoft 社と Intel 社が共同で規格化した電源管理に関する規約である。 APM によって OS 側から電源を切ったり省電力状態に移行したりすることができる。元々はノート PC 等においてバッテリの消費量を押さえるために規格化された機能であったが、最近のデスクトップ機やサーバ機等でもサポートされている。

APM によってスタンバイ状態とサスペンド状態に移行することができる。スタンバイ状態は画面を止めることによって消費電力を下げることができる状態である。またサスペンド状態ではメモリ上に実行状態を保持して CPU やハードディスクを止めるため、スタンバイ状態よりも消費電力を低くすることができる。

さらにハイバネーション (SaveToDisk) という状態を備えているものもあるが、これは実行状態を含めたメモリイメージをハードディスク上に書き出して完全に電源を切ることができるものである。ハイバネーション状態は主にノート PC に採用されている機能である。ハイバネーション状態からリジュームする時はハードディスクからメモリイメージを読込むために、サスペンド状態よりもリジューム時間が長くなる。

本稿では消費電力の削減量とリジューム時間を考慮してサスペンド状態をサポートした。他の状態も同様に実装可能である。

2.3 リジューム

リジュームとはスタンバイ、サスペンド、ハイバネーション状態から復帰することである。リジュー

ムする際のイベントは、 PC に内蔵された周辺機器や BIOS によって異なる。主なイベントを以下に示す。

- 電源 (サスペンド) スイッチの押下。
- タイマ
  あらかじめ定義された時間でサスペンド・リジュームする
- モデムの着信
  モデムカードを内蔵した PC でモデムに着信があった時
- WOL(Wake On LAN)
  PCI バスに挿された LAN カードにマジックパケット[1]と呼ばれる特別なパケットが到達した時

ユーザレベルのソフトウェアからリモート PC をリジュームさせるためには WOL が最も容易に実現できる方式である。 WOL を利用するには WOL 対応の LAN カードが搭載されている必要がある。このような LAN カードに対してマジックパケットを送ると、 LAN カードがマザーボードに電源 ON の命令信号を伝えることによって PC の電源を入れる / リジュームすることができる。

3 実装

SCore クラスタシステムソフトウェアでは SCore-D と呼ばれるユーザレベルのジョブスケジューラがノード数やメモリ容量などのリソースを管理し、適切なノードにジョブをスケジューリングする。ここでは SCore-D に実装した省電力機能について説明する。

3.1 ノード内の動作

ノード内の動作の概要を図2に示す。ノードをサスペンド状態に移行する場合は Linux の apm コマンドを呼び出す。 apm コマンドは APM BIOS に対する BIOS コールで、サスペンド状態への移行を指示する。 APM BIOS は実行状態をメモリに保持したまま、 CPU やハードディスクを止めサスペンド状態に移行する。

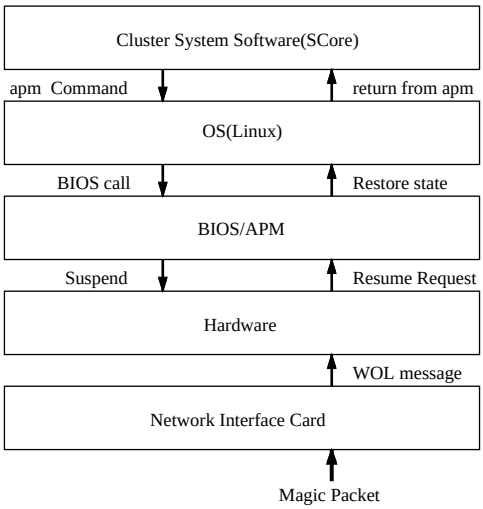


図 2: ノード内の動作

反対にノードがリジュームする場合は、マジックパケットを受け取った LAN カードがハードウェアに対して WOL メッセージを送る。 WOL メッセージを受けたハードウェアは APM BIOS に対してリジュームコマンドを発行し、 APM BIOS が実行状態を復帰させた後で Linux に制御が戻される。

3.2 サスペンドタイミング

SCore-D は全てのノードで動作しており、それぞれが協調してユーザジョブを並列実行している。ユーザのジョブが無いアイドル状態の時、 SCore-D はリスト 1に示すような Idle ループを実行しながらユーザのジョブの到着を待っている。

```
void ult_idle_hook( void ) {
    ・・・・・・
   timeout.tv_sec = 0;
   timeout.tv_usec = 500*1000;
   errno = 0;
      n = select( fd_max+1, &fdsc, (fd_set*) NULL,
(fd_set*) NULL, &timeout );
   if( n == 0 )
      return;                          // <--- Idle 状態
   ・・・・・・
}
```

リスト 1: Idle ループの動作

[1]マジックパケットは AMD 社が開発した WOL のための特殊なパケットで、 6 つの 0xFF と 16 個の LAN カードの MAC アドレスを並べたものである。このパケットをネットワークに対してブロードキャストすることによって電源投入やリジュームを行うことができる。

select()システムコールは timeout で指定した時間内に、監視しているファイルディスクリプタに変化がない場合には 0 を返す。リスト 1内のselect()システムコールは、処理すべきメッセージの到着を監視しており、 0 を返した場合にはこのノードで実行するべきジョブが無いという事を意味する。そこでselect()システムコールが 0 を返した回数をカウントしておいて、 サスペンドするまでの時間を指定するIDLE_COUNT_MAXの値を上回ったときに APM によってサスペンド状態に移行させる。この時ジョブの割り当てを管理しているサーバノードはサスペンド状態に移行させないようにしなければならない。サスペンド状態から復帰した時にはカウンタの値をリセットして次回のサスペンドに備える。

3.3 リジュームタイミング

SCore で提供されているマルチスレッドテンプレートライブラリ MPC++ [8] ではリスト 2に示すようなテンプレートによってノード番号 n 上で関数 f() を呼び出すことができる。 SCore-D は MPC++ で記述されており、これらのテンプレートによってリモートノードでの関数呼び出しを実現している。

```
invoke(int n, void (*f)(...), ...);
ainvoke(int n, void (*f)(...), ...);
```

リスト 2: invoke テンプレート

invoke()は関数f()の終了を待つ同期型呼び出しで、ainvoke()は終了を待たずに処理を進める非同期型の呼び出しである。ノード番号 n が自ノード番号と異なっている場合は、リモート呼び出しを意味する。リモート呼び出しの場合にはリモート側のノードをリジュームしてから関数呼び出しを行うようにする。こうすることでノードが一台ずつ順番にリジュームされるために全ノードが同時にリジュームすることによる、いわゆる突入電流を抑えることができ、電源装置にかかる負荷を軽くすることができる。

リジュームには対象ノードの MAC アドレスを使ったマジックパケットを送出することによって行う。SCore においてクラスタのノードやネットワークの情報を管理しているデータベースサーバ “scoreboard” は、 LAN カードの情報としてノード番号と MAC アドレスをテーブルとして保持している。マジックパケットを生成する場合には scoreboard に問い合わせることによって対象ノードの MAC アドレスを得ることができる。

全てのリモート呼び出しごとにリジュームを行うと、サスペンドしていないノードに対してもマジックパケットを送ることになって無駄が生じてしまう。そこで各ノードの状態を記憶しておいて、これがサスペンド状態である時にだけマジックパケットを送出することで無駄を省くことができる。サスペンドするノードとリジュームさせるノードが異なることから、各ノードの状態は全ノードで共通に参照できるメモリ空間に記憶しておかなければならない。これは MPC++ のGlobalPtr<T>クラステンプレートで実現することができる。GlobalPtr<T>クラステンプレートは任意の型をパラメータとして受け取って、その型のオブジェクトを指すグローバルポインタを生成する。グローバルポインタによって全てのノードから共通にアクセスすることができるようになる。

3.4 ジョブの終了

ジョブが終了すると SCore-D は全てのノードのハードディスクをシンクするためにsync_all()関数が呼び出される。これは全てのノードに対してainvoke()テンプレートによってsync()システムコールを呼び出すためのものである。これを呼び出すとジョブが割り当てられていなかったサスペンド中のノードにもsync()システムコールを呼び出す時にリジュームしてしまう。これは不必要なリジュームであるためsync_all()関数の中で実行中のノードにだけsync()システムコールを実行するようにした。

3.5 競合状態の回避

サスペンドするノードとリジュームさせるノードが異なることから図 3に示したような競合状態によってデッドロックが発生する可能性がある。

図 3のようにノード 0 でノード 1 に対してリモート関数呼び出しを行うことを考える。ノード 0 はノード 1 がサスペンド状態ではないことを確認してから関数を呼び出す。この時、ノード 1 の状態確認とリモート

	CPU	メモリ	NIC	ノード数
クラスタ A	Celeron 466 MHz	256MB	Intel 82557	12
クラスタ B	Crusoe 700MHz	128MB	RealTek 8139C	7
	Crusoe 633MHz	128MB	Intel 82557(PCMCIA)	1

表 1: 実験環境

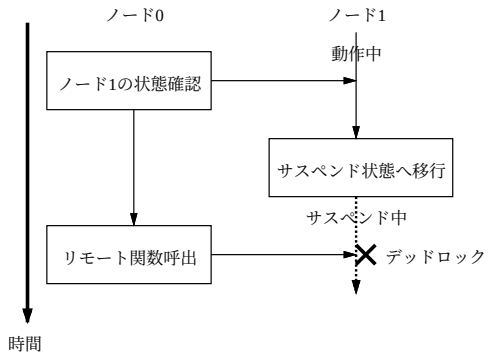


図 3: 競合状態

呼び出しの間にノード 1がサスペンドしてしまった場合、関数呼び出しが行われずにデッドロックが発生することになる。

これを回避するためにリジュームされた場合に Idle ループを回った回数を −1 に設定する。 Idle ループ側ではカウンタが −1 に設定されていた場合にはサスペンド要求を出さないようにする。ジョブが終了した時に呼び出されるsync()システムコールの前でカウンタの値をリセットしてサスペンド可能な状態に戻す。これによってジョブが割り当てられてから終了するまでの間は、そのノードがサスペンドしなくなりデッドロックを起すことを回避できる。

4 実験

4.1 実験環境

実験には 2 種類のクラスタを用いた。それぞれの構成を表 1に示す。これらのクラスタで SCore-4.2.1 上に省電力機能を実装して実験を行った。クラスタ A では SCore-4.2.1 に付属の NIC ドライバで動作したが、クラスタ B の NIC はでは動作しなかった。そこで Donald Becker 氏による “rtl8139” というドライバに WOL 機能を追加して実装した。また本実験では処理性能に対するチューニングは行っていない。

4.2 実験結果

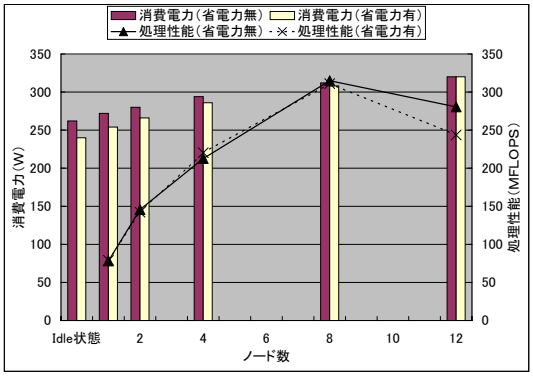
図 4に各クラスタ上で MPI 版の姫野ベンチマーク [9] を実行したときの最大消費電力と処理性能を示す。問題サイズは SMALL (64 × 64 × 128) である。横軸はベンチマークを動作させたノード数を示しており、アイドル状態はベンチマークを動作させていない時の消費電力を測定した結果である。処理性能は姫野ベンチマークが表示した MFLOPS 値でスタートアップ時間などは含まないループのみの値である。

図 4から省電力機能を使わない場合でも使用するノード数が少なくなるに従って消費電力も減少していることがわかる。これはシステムがアイドル状態の時には CPU で halt 命令を実行することによって CPU 自体の消費電力が低下しているためである。省電力機能を有効にした場合はさらに消費電力が減少させることができる。図からクラスタ A ではユーザのジョブが無いアイドル状態では全体で約 10% の消費電力が削減することがわかる。またクラスタ B では約 1/3 まで消費電力を低下させることができる。

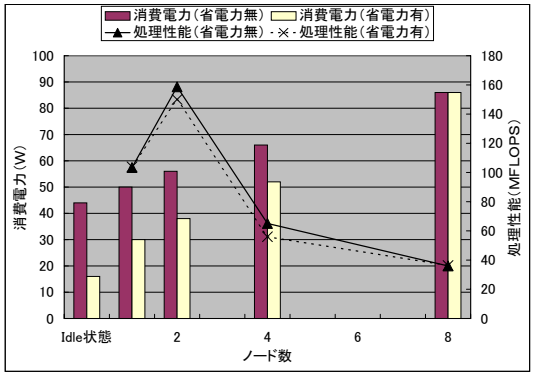
ベンチマークの MFLOPS 値は省電力機能を有効にした場合と無効にした場合でほとんど差がない。これによって本機能を実装したことによる処理性能に対する影響は小さいことが確認できる。

サスペンド状態に移行した時の消費電力の削減量は HDD や冷却ファンなどの周辺機器への給電を止めるかどうかに依存している。一方、一度止めた機器に給電を開始して再起動するにはある程度の時間がかかる。機器が多くなればなるほどリジュームにかかる時間が大きくなってしまう。そのため、低消費電力化とリジューム時間はトレードオフの関係にあると考えることができる。

そこで各クラスタ上でサスペンドすることによる起動時間を測定した。結果を図 5に示す。グラフはサスペンドした場合の全実行時間からサスペンドしない場合の全実行時間を差引いた値をプロットしたものであ



(a) クラスタ A



(b) クラスタ B

図 4: 最大消費電力と処理性能

る。どちらのクラスタもノード数に比例して起動時間が増加していることがわかる。これは今回の実装では 1 ノードずつリジュームしているためで、いくつかのノードをまとめてリジュームすることによって起動時間を短縮することができると考えられる。

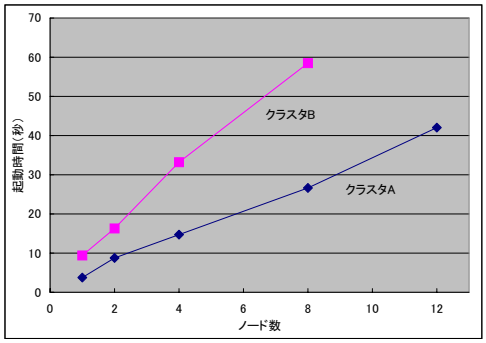


図 5: 起動時間

5 まとめ

本稿では省電力クラスタシステムを SCore 上に実装し、ベンチマーク実行時の最大消費電力を測定することによってその有効性を確認した。

PC の電力管理機能には APM の他に ACPI (Advanced Configuration and Power Inteface) が提案されており、 ACPI をサポートした PC も多く製品化されてきている。 Linux ではカーネル 2.4 から本格的な ACPI のサポートが始まっているが、動作が不安定となるため今回は APM での実装を行った。将来的に ACPI が安定して利用できるようになれば、本方式によって APM よりもより柔軟に電源管理を行うことで省電力化を実現できると考えられる。

今後は積算した消費電力を測定し、ベンチマーク全体の消費電力量の評価を行う予定である。

参考文献

[1] SafeCLUSTER, “http://jp.fujitsu.com/”.

[2] ClusterPerfect, “http://www.toshiba.co.jp/”.

[3] CLUSTERPRO, “http://www.nec.co.jp/”.

[4] 男澤昌哉, 新井リンダ, 鈴木賢剛, 森蔭政幸 : “LINUX HA クラスター”, オライリー・ジャパン (2001).

[5] RedHat, “http://www.redhat.com/”.

[6] Ishikawa, Y., Hori, A., Tezuka, H., Sumimoto, S., Takahashi, T., O’carroll, F. and Harada, H.: “RWC PC Cluster II and SCore Cluster System Software – High Performance Linux Cluster”, in *Proceedings of the 5th Annual Linux Expo*, pp. 55 – 62 (1999).

[7] Beowulf, “http://www.beowulf.org/”.

[8] 栄純明, 石川裕, 松岡聡, 高橋俊行 : MPC++-on-MPI のコモディティクラスタ環境における評価, ハイパフォーマンスシステム, 第 41 巻 (2000).

[9] 姫野ベンチマーク, “http://w3cic.riken.go.jp/HPC/HimenoBMT/”.