大規模 SMP 運用 WG
成果報告書

（活動期間：2004/10～2006/10）

第 2 版

目次

2006/10/31

サイエンティフィックシステム研究会

大規模 SMP 運用 WG

1. はじめに

HPC 分野の計算エンジンとなるスーパーコンピュータは、近年、かつての主流であったベクトル型アーキテクチャから、SMP(Symmetric Multi-Processor)を要素とする分散主記憶型の並列計算機に移行・実現されてきている。しかしその運用面においては課題が多く、会員から検討の要望があがっていた。

このような背景から、本 WG では大規模 SMP 並列機の運用方法を議論し、効率的な運用の実現を目指すとともに、必要な運用ツールについて検討・整備を行ってきた。

活動期間は 2004 年 10 月～2006 年 10 月の 2 年間で、年間 4 回程度の会合を開催した。主な活動内容は次の通りである。

(1) 大規模 SMP 並列機の運用における事例調査
(2) 大規模 SMP 並列機の運用における課題と問題点の抽出・整理
(3) 効率的な運用に必要なツールの検討・整備

活動期間中には、本 WG 方針に賛同するメンバーを会員内から募り、議論と作業に参加いただくことで、更なる情報の共有と充実を図った。

これらの活動により、会員の研究用コンピュータ運用環境の向上を実現し、本 WG の活動成果は本報告書にまとめ、会員間で広く共有できるようにした。

なお、本報告書は２年間の活動を通して検討を行った成果を中心にまとめている。各会合で配付された検討資料については、参考資料として SS 研のホームページで公開している。SS 研のホームページ「http://www.ssken.gr.jp/」も併せて参照されたい。

2. ＷＧ活動経緯

2-1. ＷＧメンバー

Sorry…
(SS 研会員限定情報につき、掲載できません)

＊SS 研会員の方は、認証付の版をご参照ください＊

2-2. 活動実績

第1回　2004年11月24日(水) 京都

- 活動方針等の確認
- 大規模SMP運用における事例検討
- 活動の進め方の検討

第2回　2005年 2月16日(水) 東京

- 大規模SMP運用における事例収集(情報交換)と課題の検討
  - OS基本機能(ジョブスケジューリング)のチュートリアル（富士通）
  - WEBインターフェース化の情報提供（JAXA）
  - サイエンスポータルの情報提供（富士通）
  - 運用における工夫点の報告（JAEA、JAXA、京大）

第3回　2005年 6月29日(水) 名古屋

- 名大基盤 HPC2500導入報告
- 富士通 各種課題に対する検討結果報告
- 今後の活動についての検討、調整
  - JAXAで提供可能なツールについて
- 名大基盤 HPC2500システム見学

第4回　2005年10月 7日(金) 東京

- HPCポータル導入結果(導入における感想など)
- 大規模SMP運用における課題の整理
- 今後の活動についての検討、調整（成果報告書の整理について）

第5回　2006年 2月17日(金) 東京

- 情報交換、話題提供
  - I/O、オープンソースの利用状況報告（JAXA）
- 大規模SMP運用における課題とそれに対する富士通の対応の整理
- 成果報告書の検討

第6回　2006年 5月30日(火) 東京

- 前回からの継続検討事項の検討
- WG成果報告書の内容の調整
- WG成果発表 実施の検討

第7(最終)回　2006年 9月20日(水) 東京

- WG成果報告書の検討
- 成果発表の確認
- 後継WG、新設WGへの意見抽出

(成果発表)　2006年10月31日(火) 京都

- WG成果発表（科学技術計算分科会にて）

3．WG 活動で出された、課題・回答・コメントのまとめ

本章では、2 年間の WG 活動を通し、大規模 SMP（PRIMEPOWER HPC2500）に対して出された課題・要望とその対応等について整理した。
これら課題・要望には富士通からの回答（或いはコメント）だけでなく、他の機関の参考になるよう、各委員それぞれの評価・分析結果や回避策も記載した。
また、すべての情報には、それを挙げた委員の機関名と日付を記載し、５つのカテゴリー別に分類している（①CPU 時間、②資源管理、③使い勝手と性能、④I／O、⑤その他）。
なお、スペース上ここで説明しきれないものには添付資料をつけた。添付資料はカラーでないと分かりづらいものもあるため、SS 研サイトでの参照を推奨する。

SS 研サイト　http://www.ssken.gr.jp/

CPU 時間

(1) CPU 時間の見せ方

【指摘】

- 複数 CPU を使用する並列ジョブの課金を、最長の CPU 使用時間（複数 CPU の中で最も CPU 使用時間の長いもの）にすることにより、アプリケーションの並列化チューニングが促進できるのではないか。（各 CPU を満遍なく使用できれば得だと思う）
- またこの場合、ジョブの打ち切り時間も課金に合わせて、プロセス単位でなく、最長の CPU 使用時間で指定したい。

<京大学情：2004/10/27(科学技術計算分科会) >

- NQS ジョブだけでなく、会話型処理の場合も同様にしてほしい（今後、ラージページを使えるような会話型の重要性が増大すると予想）

<京大学情センター：2005/10/07 第 4 回会合>

【回答】

- Parallelnavi 2.4.1 (2005 年 8 月出荷済)で提供済です。バッチジョブおよび、会話型ジョブの両方において、ジョブ打ち切り時間として CPU 単位、プロセス単位の何れかを選択可能としました。この選択は、ノード単位に設定可能ですが、全ノード共通とすることを推奨します。
  課金・統計情報については、Parallelnavi 初版から CPU 単位をサポート済です。

<富士通：2006/2/17(第５回会合) >

(2) 資源競合によりジョブの実行終了時間が予測できない。

【指摘】

- 実行時間で制限する場合は致命的な問題だ。
  実際にどの程度のバラツキなのかデータを公表してほしい。

<JAEA：2005/2/16 第 2 回会合>

【回答】

- 宇宙航空研究開発機構様、日本原子力研究開発機構様での実測・分析結果を添付資料に示します。

  添付資料 2-1,2-2 を参照

<富士通：2006/2/17(第 5 回会合) >

資源管理

(3) CPU 数のみでの並列度指定

【指摘 1】

- 現在の Parallelnavi では、スレッド数とプロセス数の両方をエンドユーザが指定しなければならない（ハイブリッド並列前提)。また、アプリケーションの特性（スレッド並列かプロセス並列か、並列数は幾つかなど）に応じてバッチキューを複数用意することを推奨している。しかし、プログラミングモデルに依存せず､ジョブ全体の CPU 数を並列度(?)として指定することが望ましい｡また、バッチキューも同様に一つのキューで各プログラムモデルを実行可能とすべきだ。

<京大学情：2004/10/27 科学技術計算分科会>

- NQS ではプロセス数とスレッド数を別々にしか制限できない。
  運用側がやりたいのはトータル CPU 数での制限だ。OS 側で実現して欲しい。

<京大学情：2005/6/29 第 3 回会合>

【指摘 1 への回答】

- ハイブリッド並列を基本プログラムモデルと想定しました。
  → スレッド数とプロセス数の両方を指定することが自然と考えました。
  → ジョブ全体の CPU 数のみを指定したいという要件は理解できますが、スレッド並列・プロセス並列・ハイブリッド並列に共通した資源量指定の良い方式が見出せていません。
- なお、NQS-JM の標準値・最大値設定機能により、通常はプロセス数（スレッド数は 1）のみ指定しスレッド並列ジョブとハイブリッド並列ジョブはプロセス数とスレッド数の両方を指定してもらうなど、エンドユーザの利便性を向上させています。

- 現状では、確かに使いにくいと考えていますが、例えばハイブリッド並列ジョブを複数ノードに跨って実行する場合、各ノードから何 CPU(=何スレッド)分の CPU を割当てないといけないのかが、ジョブ全体の CPU 数のみの指定では分からないという問題等があり、スレッド数・プロセス数を指定する仕様となっています。良い方式が見出せていない状況であり、ご指導いただきたい。

<富士通：2004/11/24 第 1 回会合>

【指摘 1 への回答へのコメント】

- JAXA では NSJS によりトータル要求 CPU 数による投入制限を実現している。
- ジョブ全体の CPU 数の制限であれば、NQS 出口により実現可能と思われる。

<JAXA：2004/11/24 第 1 回会合>

【指摘 2】

- 東大のバルク制運用が簡単に実施できるか。（ユーザ毎に使用 CPU 数を定める）

<京大学情：2005/10/07 第 4 回会合>

【指摘 2 への回答】

- Parallelnavi は、JOB が実行し始めてから資源不足で待ちに入ることを避けるため、資源を事前予約した上で JOB を走行させる仕組みを採用しています。このため、ノード間に跨いだ CPU 割り当てが可能か否かを判断するために、プロスセ数とスレッド数を指定させています。
  ただし、1JOB が利用できる最大 CPU 数を新しく設けることで、最大プロセス数×最大スレッド数分の CPU 数を用意しないでよくなるが、有効でしょうか？
  有効ならば、NQS 出口でチェックする機構を提供します。
- 東大のバルク制運用では、以下の指定で簡単な指定を実現していると思います。
  ■プログラミングモデル：プロセス並列かスレッド並列（ハイブリッド並列含む）か
  ■使用する CPU 数の合計（ただし、スレッド並列は 8CPU 固定）
  NQS-JM の標準値設定機能によりスレッド数の標準値を 8 とし、通常はプロセス数のみを指定させることにより、東大センター殿のバルク運用と同様な運用が可能と考えます。

<富士通：2006/2/17 第 5 回会合>

(4) ジョブの資源量指定が複雑で分かりにくい

【指摘】

- CPU 数/メモリ容量はジョブ種別を意識しないで指定したい。
- 誤った指定ができてしまう。(32 スレッド×32 プロセス=1024CPU)

<JAEA：2005/2/16 第 2 回会合>

【回答】

項番 3 に同じ。

なお、日本原子力研究開発機構様においては、qsub 実行時、-lp×-lP の値が指定 CPU 数未満であるかをチェックするシェルスクリプトを作成し、qsub コマンドを置き換えることで投入制限しております。

<富士通：2006/2/17 第 5 回会合>

(5) 1 ジョブをできる限り同じノードへ割り当てる

【指摘】

- ノード間並列ジョブを想定すると、跨るノード数が少ないほど、ノード異常によるジョブへの影響が確率的に少なくなる。

<京大学情：2004/10/27 科学技術計算分科会>

【回答】

- Parallelnavi の NQS-JS 機能の「PACK 割り当て方式」により、ノードの跨りが少なくなるノード割り当てをサポート済です。詳細は、「添付資料 5-1 1 ジョブを同一ノードに割り当てる方法」を参照願います。

添付資料 5-1 を参照

<富士通：2004/11/24 第 1 回会合>

(6) CPU 利用台数によるジョブの優先度制御

【指摘】

- デバッグは少 CPU で通常のバッチジョブよりも優先度を高くして実施したい。

<京大学情：2004/10/27 科学技術計算分科会>

【回答】

- 以下の二つの運用方法にて実現可能です。ご利用をご検討ください。
  (1) 少 CPU 使用ジョブ用に専用キュー(会話型キューや少 CPU キュー)を設け、当該キューの優先度を他のキューよりも高く設定する。
  (2) NQS-JS の既存機能である、資源の重み評価値機能を使用する。

  「添付資料 6-1 CPU 利用台数によるジョブの優先度制御」に、上記二方式の具体例を示しています。

添付資料 6-1 を参照

<富士通：2006/2/17 第 5 回会合>

(7) JOB スワップ時の異常 CPU の切離し

【指摘】

- JAEA では、ジョブスワップ機能をジョブ間の優先度制御の他に、実行中ジョブを異常 CPU (ジョブ実行は可能であるが性能低下するような異常) から、正常な CPU へ移動させる目的にも使えると考えていた。

しかし、スワップアウト中に異常 CPU を割当抑止（DISABLE）状態としても、スワップイン時に異常 CPU がそのまま割当てられ実行された。

<JAEA：2004/11/24 第 1 回会合>

【回答】

- Parallelnavi の割当抑止機能は、新規に走行を開始するジョブへの資源割当を抑止することを目的に開発していたため、ご指摘のように割当抑止設定した異常 CPU へのスワップインが行われました。しかし、改善が望ましいため、ハード故障全体への対応を踏まえた機能強化を検討させていただきます。
  なお、当面は、日本原子力研究開発機構様にて実施いただいている方法により、代替をお願いします。
  《代替手段》
  日本原子力研究開発機構様では、CPU 異常発生時、当該 CPU がジョブ用 CPU であれば disable に設定し、CPU のグループ定義(sg.conf ファイル)も書き換えることで、新規ジョブやスワップジョブを割り当て対象外に設定するシェルツールを作成して対応。

<富士通：2006/2/17 第 5 回会合>

(8) CPU・メモリの使用量を考慮した資源割り当て＝資源の効率的利用

(例：多 CPU 少メモリジョブと少 CPU 多メモリジョブを同一ノードへ割当)

【指摘】

- 計算資源を効率的に利用し、システム全体の稼動率を向上させるためのジョブスケジューリングが必要。

<京大学情：2004/10/27 科学技術計算分科会>

【回答】

- PRIMEPOWER HPC2500 ではジョブへ割当てる資源種別と割当上の条件が複雑であり、最適資源探索処理のオーバヘッドが非常に大きくなります。このため、オーバヘッドの削減を最優先に考慮し、資源間のバランスを考慮した高度割当はサポートせず、比較的単純な資源探索ロジック（ノード→CPU→ノード内バリア→ノード間バリア→DTU の順に空き資源を探索し割当て）を採用しています。
- オーバヘッドの削減を優先するか、資源間のバランスの考慮を優先するかはセンターによって異なる筈なので、今後はカスタマイズ性に力をいれたいと考えています。

<富士通：2004/11/24 第 1 回会合>

(9) ジョブスケジューリング

【指摘】

- ジョブ投入の際、Parallelnavi ではプロセス単位に空き資源をチェックし割当てているが、京大のように電気代を節約したいサイトでは、逆に空きノードを作りたい。ユーザ側で、そのような定義ができると良い。

  <京大学情：2005/2/16 第 2 回会合>

- JAEA はノード数が少ないので、大きなジョブが投入されることを想定し、通常は京大と同様に、可能な限り空きノードを作っておきたい。

  <JAEA：2005/2/16 第 2 回会合>

【回答】

- 「ノード集中割り当て機能」を 2006 年 12E 目標でご提供します。
  詳細な仕様は、「添付資料 9-1 ジョブスケジューリングの定義(空きノードを作るスケジューリング)」をご参照ください。

添付資料 9-1 を参照

  <富士通：2006/2/17 第 5 回会合>

(10) ラージページとノーマルページの二重構造

【指摘】

- ラージページをエンドユーザが意識しなければならない。
  コンパイル時にラージページ指定忘れなどによりノーマルページを枯渇させるとシステムハングに陥る。

  <京大学情：2004/10/27 科学技術計算分科会>

【回答】

- ラージページの利用を促進するために、以下の機能をサポート済みです。
  - ・ センター毎にラージページをコンパイル時のデフォルトとする機能
  - ・ ノーマルページとラージページの制限を別々に定義できる機能

  <富士通：2006/2/17 第 5 回会合>

(11) ジョブ実行に必要なメモリ量を如何に見積もるか

【指摘】

- エンドユーザがメモリ（ラージページ）使用量を指定しなければならないが、面倒なので大きく指定してしまう。このため、実際には使用していないメモリもリザーブしてしまうため、後続ジョブがメモリ資源不足で実行できず、システムの稼動率が低下する。正しいメモリ使用量を指定させるための機能の充実が必要。

  <京大学情：2004/10/27 科学技術計算分科会>

【回答】

- 自らプログラムする場合、ある程度想定可能であるが、スタック域など見積りが難しい、プログラムしないエンドユーザには見積もりができないなどの課題があります。
- 現在は、エンドユーザがプログラムを一度実行して使用量を把握し、次回からの実行で使用量を指定していただくよう願いたい。

<富士通：2004/11/24 第 1 回会合>

- なお、宇宙航空研究開発機構様では、ラージページメモリ使用量（text 域、data 域、bss 域）を計算する概算するコマンド「lsize」を作成し、ユーザ指定を簡便化しています。
  仕様については、第 2 回会合での運用ツールで説明されていますので、参照ください。

添付資料 11-1(P1 No1)を参照

<富士通：2005/2/16 第 2 回会合>

【回答へのコメント】

- エンドユーザのプログラムは多種多様なので、ジョブスケジュールの導入が必要不可欠と思う。

<JAXA：2004/11/24 第 1 回会合>

(12) 動的なメモリ解放機構の実現

【指摘(第一次)】

- 要求メモリと実際にジョブで使用されるメモリ量との乖離が激しい。

<京大学情：2004/10/27 科学技術計算分科会>

- ユーザはアプリを開発途上であり、固定的な指定ができないため、大きく指定する傾向がある。動的なメモリ開放機構をぜひ実現して欲しい。期待する仕様は、凡そ以下の通りである。
  ○センタデフォルトとして以下の選択
  1)動的メモリ解放機構を ON/OFF するか選択
  2)ジョブ開始から開放するまでの Interval 時間
  ○ユーザ指定(qsub オプション)で変更可能
  ○マルチブロックジョブを意識したヘテロなメモリ資源開放？

<京大学情：2005/6/29 第 3 回会合>

- 誤って必要以上に確保された資源を解放して他ジョブで利用したい。

<JAEA：2005/2/16 第 2 回会合>

【回答(第一次)】

- 非ラージページ：ページング（後続ジョブ実行の阻害要因にはならない）、ラージページ：事前割当てし、ジョブ実行中の動的変更なし。
  指摘のようなジョブ動作途中でのメモリ開放を行った場合、メモリ開放以降に

発生したメモリ獲得要求を如何に処理すべきかという課題があります。ジョブアベンドでよいのか？他ジョブをスワップアウトして空きメモリを作るのか？（本当に稼働率は向上するのか？）
この機能を追加した場合、メモリ開放以降に発生したメモリ獲得要求を如何に処理すべきか、ジョブアベンドでよいのか、他ジョブをスワップアウトして空きメモリを作るのか。どのように考えればよいかご指導願いたい。

<富士通：2004/11/24 第 1 回会合>

【回答(第一次)へのコメント】

- オプションにより、エンドユーザに選択させてもよい。ただし、一定時間後に開放することを選択したエンドユーザがなんらかの形で有利となるような運用上の工夫が必要である。

<JAXA：2004/11/24 第 1 回会合>

- なお、JAXA では、ラージページを使用しないユーザが多い。それらのジョブはセンター管理者が強制終了し、ロードモジュールを正しく作成(コンパイル)するよう指導している。エンドユーザをこのような形で指導(教育)していくことも重要。

<JAXA：2004/11/24 第 1 回会合>

【回答(第二次)】

- ご要望に対する検討結果を回答します。現在、 以下のような機能仕様で実現を考えています。
  - 動的にメモリ解放機構を使用するか否かを qsub オプション/キューで設定
  - メモリ解放するタイミングをジョブ実行開始から自動解放までのインターバルで指定
  - インターバル経過後、未使用ラージページをセグメント単位（ディフォルト 64MB）に解放。以降のメモリ要求はエラー終了（エラー終了した場合、自動解放したジョブである旨、実行結果にレポート）

《確認事項》
本機能によりメモリ資源を有効活用する運用を実現するには、エンドユーザが自動解放を積極的に利用するための推進が必要と考えます。例えば、メモリ課金により、メモリチューニングを考慮したエンドユーザは、安価にシステムが利用できるような運用ルール等（デフォルトで動的メモリ解放を運用するならば別ですが)。
このような運用ルールの変更も踏まえ、本機能の必要性について再度のご検討をお願いいたします。

<富士通：2006/2/17 第 5 回会合>

《必要性の検討結果》
各センターのメモリ利用状況を確認した結果、本機能の緊急性は高くないとの結論を得た。ただし、センター運用上は有効な機能と考えられるので、今後のエンハンス項目の参考とする。

(13) ノード別に分散された空き資源を集約することができない

【指摘】

- 各ノードに分散された空き資源をまとめて、新しいジョブを実行したい（ジョブ実行状況により空き資源の分散が必ず発生、実行中ジョブを他ノードへ移動できれば対応可能）。 <JAEA：2005/2/16 第 2 回会合>

【回答】

- ジョブマイグレーションにより実現可能だと認識しています。
  現在、次期システムでのジョブマイグレーション/ジョブ凍結/定期的チェックポイントの実現に向け、技術的な目処がつき、プロトタイプ版の開発に着手しました。

<富士通：2006/2/17 第 5 回会合>

(14) 実行中ジョブの制限値が変更できない

【指摘】

- 実行時間などの制限値を一時的に変更したい。

<JAEA：2005/2/16 第 2 回会合>

【回答】

- 実行中ジョブの制限値（CPU 時間や経過時間制限値）を変更できるように改善します。
  変更機能として、ジョブ投入者、システム管理者向けに qmodify コマンドを新規に提供予定（2006 年 12E 目標）です。詳細は、「添付資料 14-1 実行中ジョブの制限値変更について」をご参照願います。

  > 添付資料 14-1 を参照

<富士通：2006/2/17 第 5 回会合>

(15) UDTU 活用が難しい

【指摘】

- ノード内が UDTU 未使用ジョブのみとなった場合に UDTU が無駄となる。
- UDTU の使用状況が把握しにくい。Shared-UDTU は遅い。(プログラム依存)
- 測定結果を例示してほしい。

<JAEA：2005/2/16 第 2 回会合>

【回答】

- 宇宙航空研究開発機構様及び日本原子力研究機構様での評価結果を添付資料 15-1 及び 15-2 に示します。

  > 添付資料 15-1,15-2 を参照

<富士通：2006/2/17 第 5 回会合>

(16) NQS のバッチ share 実行

【指摘】

- 期待通りの share 実行とはならない。

<JAXA：2005/2/16 第 2 回会合>

【回答】

- Share モードに対する見解については、第 3 回会合（2005/6/29）で説明した添付資料 16-1「Share 運用での課題に対する回答」を参照されたい。

  添付資料 16-1 を参照

<JAXA：2005/6/29 第 3 回会合＞

(17) Checkpoint/Restart（長時間ジョブ向け対策）

【指摘】

- CPU 異常でプロセスが KILL されてしまった場合、長時間ジョブを先頭から再実行では非常に無駄。異常発生直前のチェックポイントから再開できると無駄が少なくて済む。

<京大学情：2004/10/27 科学技術計算分科会>

- Checkpoint/Restart（長時間ジョブ向け対策）は、将来的にもぜひ要望したい。

<JAXA：2004/11/24 第 1 回会合>

【回答】

- 項番 13 の回答を参照ください。

<富士通：2006/2/17 第 5 回会合>

(18) キャッシュ縮退時に性能劣化の考慮

【指摘】

- キャッシュが縮退すると性能にブレが発生し、ジョブ実行性能の再現性が保証できなくなる。当該 CPU を新規ジョブには割当てない考慮が必要。

<京大学情：2004/10/27 科学技術計算分科会>

【回答】

- キャッシュ縮退が発生した場合、syslog メッセージ(注)が出力されますので、以下の対応をお願いします。
  - swatch(フリーソフト)や Systemwalker などのメッセージ監視機能を利用し、キャッシュ縮退メッセージを監視します。
  - メッセージ出力時に当該 CPU を Parallelnavi の割当抑止機能(pwadmin コメンド)により無効化し、新規ジョブへの割当を抑止します。

注）syslog メッセージ

WARNING: [AFT1] WAY Degradation Event detected by CPUX, errID XX.XX AFSR XX.XX AFAR D1 XX.XX AFAR U2 XX.XX

・"WAY Degradation" ： キャッシュ縮退が発生したことを示します。

・"CPUX" ： キャッシュ縮退が発生した CPU を示します。

X の個所に当該 CPU の CPU-ID が出力されます。

<富士通：2006/2/17 第 5 回会合>

(19-1) 一つの CPU 故障だけでノードを停止させない（ﾕｰｻﾞﾌﾟﾛｾｽ走行時）

【指摘】

- CPU 故障が発生した場合、現状、プロセス KILL 後リブートであるが、リブートまでしなくても良いのでは。

<京大学情：2004/10/27 科学技術計算分科会>

【回答】

- CPU 故障発生時、プロセスの動作を保証することができないため、当該プロセスを kill しています。さらに、kill されたプロセスが運用上必須なプロセスである可能性を考慮し、システムをリブートしています。
- Solaris10 で以下のようなノードダウンしない仕組みを実現しました。
  ・ユーザプロセス走行時の CPU 異常 ：実行中のプロセスを kill した後、重要なプロセスが kill された場合のリスタート方法を定義できる機能(Solaris Fault Manager：SMF）により、CPU 縮退した状態で運用継続が可能になります。

<富士通：2006/2/17 第 5 回会合>

(19-2) 一つの CPU 故障だけでノードを停止させない（OS 走行時）

【指摘】

- OS 走行時に異常発生しても簡単にパニックさせないでほしい。
  現状、OS での CPU 異常はパニック。一つの CPU 異常で、ノードダウンしないような考慮が必要。

<京大学情：2004/10/27 科学技術計算分科会>

- 一部の障害でノード全体がダウンしてしまう。
  ジョブ用システムボードで障害が発生してノード全体がダウンしてしまう。

<JAEA：2005/2/16 第 2 回会合>

【回答】

- CPU 異常発生時は、データの保証ができないためパニックとせざるを得ません。

<富士通：2006/2/17 第 5 回会合>

(20) ジョブ凍結

【指摘】

- 保守作業にノード停止が必要な場合、当該ノード上で実行中の長時間ジョブの実行継続性を保証する必要がある。その手段として、実行中ジョブの凍結→システム停止→保守→システム起動→ジョブ解凍が考えられる。

<京大学情：2004/10/27 科学技術計算分科会>

【回答】

- 定期的チェックポイント(ダウン対策)とジョブ凍結（システム保守向け）の実現については、項番 13 の回答を参照ください。
  現システムでは、システム凍結機能をご利用いただきたい。システム凍結を利用できる範囲は以下のとおりです。
  - システムメモリ全体を凍結・解凍
  - システム凍結中の保守：CPU，メモリ，SB，DDCON のハード交換可
  - SYSVOL ミラー：Solstice Disk Suite を利用（GDS による SYSVOL ミラーは不可）
  - テープ装置がなど DR(Dynamic Reconfiguration)不可なハードが未接続であるシステムのみ <富士通：2006/2/17 第 5 回会合>

(21,22) ジョブマイグレーション

【指摘】

- 上記に加え、他ノードへマイグレーション可能であれば、ジョブの一時中断が短時間で済み、運用性が向上する。

<京大学情：2004/10/27 科学技術計算分科会>

- ジョブが走っていると、ノードの電源が切れない。

<名大基盤：2005/6/29 第 3 回会合>

【回答(第一次)】

- 重要性は認識しているものの、Solaris OS での実現には多くの時間が必要なため、非サポートとしています。

<富士通：2004/11/24 第 1 回会合>

【回答(第二次)】

- 次期システムでのジョブマイグレーションの実現に向け、技術的な目処がつき、プロトタイプ版の開発に着手しました。

<富士通：2006/2/17 第 5 回会合>

(23) 障害ノードが運用中に活性保守できない

【指摘】

- 障害復旧にはシステム停止による部品交換が必要である。
  NQS ジョブが実行中で停止できない。(実行中ジョブを他ノードへ移動できれば対応可能であるが)

<JAEA：2005/2/16 第 2 回会合>

【回答】

- PRIMEPOWER HPC2500 では、拡張インターリーブをサポートしているため以下の範囲のみ活性保守が可能となっています（SB の活性保守が制限)。
  ・SCF(システム制御ボード) (ノード/HS)
  ・フロントエンド電源(AC-DC 変換ユニット) (ノード/HS)
  ・クロスバ用電源ユニット(DC-DC コンバータ)
  ・HS 用電源ユニット
  ・冷却ファン (ノード/HS)
  ・内蔵ハードディスク(ミラーなど要)
  ・HS 側光モジュール(但し該当ポートに接続されるノードは切り離し)

<富士通：2006/2/17 第 5 回会合>

(24) ホットスタンバイ時の省電力

【指摘】

- VPP に比べ消費電力が多く、エンドユーザへサービスしていない間の消費電力を極力抑えたい。

<京大学情：2004/10/27 科学技術計算分科会>

【回答】

- SMP の UNIX サーバは他社も同様に、CPU・メモリ・ディスクともアイドル時を含め稼動状態のままです。CPU はアーキテクチャ上動作したままです。

<富士通：2004/11/24 第 1 回会合>

(25) 起動・停止と縮退運転：高速 IPL

【指摘】

- エンドユーザへサービスしていない間の消費電力削減。

<京大学情：2004/10/27 科学技術計算分科会>

【回答】

- 電源投入時に多くの時間が必要であったハードウェア診断(POST)については、メモリチェックを全 CPU で行うことや CPU の性能向上により、概ね 40 分(1.3GHz CPU)を 25 分(2.08GHz CPU)に改善しております。なお、複数のお客様から POST

の簡易化を要請されておりますが、リブート時に構成変更されている可能性があり、診断を削除することはできません。

<富士通：2006/2/17 第 5 回会合>

(26) 電気使用量

【指摘】

- VPP5000 と PRIMEPOWER HPC2500 の電気使用量を比較した際、HPC2500 が高い。(2 倍弱)

<名大基盤：2005/6/29 第 3 回会合>

- 電気代節約、障害時保守の効率化のため、JOB マイグレーションを可能にしてほしい。たとえ実現に時間がかかっても構わない。

<京大学情：2005/6/29 第 3 回会合>

【回答】

- 項番 9 に示した「ノード集中割り当て機能」によるノード資源の効率利用をご検討ください。

<富士通：2006/2/17 第 5 回会合>

(27) 起動・停止と縮退運転：ノード起動・停止の空調機連動

【指摘】

- ジョブ混雑具合により、ノードを起動・停止し省電力を図る → ノード起動・停止と連動し空調機も起動・停止

<京大学情：2004/10/27 科学技術計算分科会>

【回答】

- 空調機のインタフェースなどセンター毎に対応内容が異なるため、個別に対応させていただいております。
  宇宙航空研究開発機構様では、各ノードと空調機の高度な自動連動機構を開発されています。

<富士通：2006/2/17 第 5 回会合>

(28) 起動・停止と縮退運転：ノード起動・停止の NQS 連携

【指摘】

- ジョブの混雑状況に応じて、ノード起動・停止を自動化。停止時は、当該ノードにジョブが新規に実行されないなどのジョブスケジューリング上の工夫が必要。

<京大学情：2004/10/27 科学技術計算分科会>

【回答】

- ジョブ混雑の考え方、起動・停止タイミングなどお客様毎に要件が異なるため、富士通の製品では制御するための部品(電源制御コマンド、ジョブ状態監視コマンドなど)を提供しています。京都大学様、宇宙航空研究開発機構様での利用方法は以下のとおり。

□京都大学様

ノード起動：ジョブのキューイング状況をチェックし、待ちジョブの alloc CPU 数に応じて起動ノード数を決定。128CPU 以内で 1 ノード、129～256CPU なら 2 ノードといったルールを決めており、なるべく空調機と連動しているノードのペアで起動させている。現在は 20 分単位で cron で実行。

ノード停止：単位時間(例えば 1 時間)内でノードにジョブがいなければノード停止。
ただし、猶予時間を与えそれ以内にジョブが投入された場合は停止処理を中断する。現在は 30 分単位で cron で実行。

□宇宙航空研究開発機構殿様

NSJS(独自のスケジューラ)と設備の連動機構を開発しています。
例えば、NSJS はカレンダを持っており、停止対象ノードのジョブ割当を抑止し、安全に停止する機構がある。

<富士通：2004/11/24 第 1 回会合>

(29) 起動・停止と縮退運転：ジョブ凍結・マイグレーション

【指摘】

- ノード停止時に実行中ジョブを他ノードへマイグレーション（もしくはジョブ凍結）できれば、より一層効率的な省電力運用が可能。

<京大学情：2004/10/27 科学技術計算分科会>

【回答(第一次)】

- 重要性は認識しているものの、Solaris OS での実現には多くの時間が必要なため、非サポートとしています。

<富士通：2004/11/24 第 1 回会合>

【回答(第二次)】

- 次期システムでのジョブマイグレーションの実現に向け、技術的な目処がつき、プロトタイプ版の開発に着手しました。

<富士通：2006/2/17 第 5 回会合>

⊞ 使い勝手と性能

(30) qsub のシェルでは融通性がない

【指摘】

- エンドユーザからみた使い勝手の向上を図りたい。
  例えば以前の MSP のように、キーパラメータだけでプログラム名を指定し変更できるとよい。また、やがては WEB インタフェース化したい。

<京大学情：2004/10/27 科学技術計算分科会>

【回答】

- エンドユーザの使い勝手向上は、Parallelnavi Workbench にてカバーする考えですが、現状の使い易さをどう把握しているのかや WEB インタフェースのサポート等について、開発部門に状況と方針等を確認し、検討させていただきたい。

<富士通：2004/11/24 第 1 回会合>

【コメント】

- メーカから提供されている Workbench を使用することは、エンドユーザにとっては難しい。

<京大学情：2004/11/24 第 1 回会合>

- JAXA では、プラットフォームが変わっても共通のエンドユーザ向けインタフェースを提供するために NS コマンドというスクリプトレベルの簡易インタフェースを提供している。Workbench は高機能である反面、簡単に使用したいエンドユーザには向いていない。

<JAXA：2004/11/24 第 1 回会合>

- メーカが Workbench を製品化するとエンドユーザにとって難しすぎるものになるとの印象がある。もっと軽く簡単なものでよい。

<京大学情：2004/11/24 第 1 回会合>

JAXA 開発のツール、HPC ポータル等を導入し検討する。<全委員>

(31)特定のジョブの実行が他のジョブの経過時間に影響を与える問題

【指摘】

- 特定のジョブの実行により、転送回数が多い他のジョブの経過時間に大きく影響を与えてしまう問題が発生していた。この原因は、DTU であった。転送要求の多いジョブはプライオリティを高めるなど、ユーザごとに DTU を調節する機能が欲しい。
- また、この原因を解明するのに 2 年半もかかった。このような問題点は、開発者側から予め教えてほしかった。

<JAXA：2005/2/16 第 2 回会合>

【回答】

- DTU 競合による他のジョブの経過時間に影響を与える原因の説明を添付資料31-1 に示します。
  また、宇宙航空研究開発機構様での経過時間のブレ分析結果を添付資料 31-2 に示します。

添付資料 31-1,31-2 を参照

<富士通：2006/2/17 第 5 回会合>

(32) ジョブ実行状況の確認が遅い

【指摘】

- rscsetstat コマンドのレスポンスが遅く、負荷も高い。
  ※多重実行でメッセージ

<JAEA：2005/2/16 第 2 回会合>

【回答】

- Parallelnavi 2.4.1 にて、ノード内の情報収集処理の性能改善を実施済み。
  例：128 プロセス並列/1 ノードの情報収集処理：20 秒→1 秒に減少

  《回避策》(日本原子力研究開発機構様、宇宙航空研究開発機構様での回避策)
  cron で定期的に rscsetstat コマンドを実行しファイルへ保存、エンドユーザはこのファイル内容を表示するスクリプトを利用している。

<富士通：2006/2/17 第 5 回会合>

(33) メモリアクセス性能改善

【指摘】

- キャッシュミス/TLB ミスのプログラム改善対策はユーザ負担が大きい。
  既存プログラムの高速化チューニングが容易ではない。

<JAEA：2005/2/16 第 2 回会合>

【回答】

- 宇宙航空研究開発機構様では、性能チューニングのために、性能チェックシートを作成し、ユーザ支援する仕組みを実施しています。
  宇宙航空研究開発機構様で利用しています「性能阻害箇所の検出ツール」及び「性能チェックシート」を添付資料 33-1 及び 33-2 に示します。

添付資料 33-1,33-2 を参照

<富士通：2006/2/17 第 5 回会合>

(34) ポートスキャンで NQS トラブル

【指摘】

- ポートスキャンで NQS バッチリクエスト起動デーモン(nqsexecd)異常終了
- 新規ジョブが実行できなくなったため、改善要望し、該当デーモンの異常終了を監視するようにした。
  トラブルシューティングとして運用に必要なサービス(デーモン)が正常に起動されているか確認する手段が必要と思われる。

<JAEA：2005/2/16 第 2 回会合>

【回答】

- NQS デーモンの異常終了は、Parallelnavi 2.4.1 で対応済です。
  Parallelnavi で提供するそれ以外のデーモンで同様な問題がないことは確認済みです。
  《回避策》(日本原子力研究開発機構様での回避策)
  ps コマンドを定期的に実行し、nqsexecd が起動されていない場合にメール通知するような運用ツールを作成しで対処しています。

<富士通：2006/2/17 第 5 回会合>

(35) 適切な稼働率の計算方法

【指摘】

- 稼働率が低いノードからジョブを詰めていくため、ジョブが分散し稼働率が低下してしまう。(稼働率の定義を含め、評価方法を検討中である)

<名大基盤：2005/6/29 第 3 回会合>

【回答】

- 項番 9 に示した「ノード集中割り当て機能」により、稼働率の高いノードからジョブをつめていく機能を予定しています。

<富士通：2006/2/17 第 5 回会合>

(36) 不足資源調査コマンド

【指摘】

- 現在の仕様では、Reason 欄に、どの資源が不足しているかが文字列で表示される。(例：CPU,MEM,UDTU)
  より分かりやすくなるように、グラフィカル表示してもらいたい。

<名大基盤：2005/6/29 第 3 回会合>

【回答】

- 名古屋大学様：JAXA 殿で稼動中の CMS(システムの利用状況を Web ブラウザによりグラフィカル表示するツール)を流用することにより対応予定である。

<富士通：2006/2/17 第 5 回会合>

I／O

(37) PRIMECLUSTER　GFS によるディスク共用は遅い

【指摘】

- 小規模ファイルを多数アクセスすると遅い。

<JAEA：2005/2/16 第 2 回会合>

【回答】

- PRIMECLUSTER GFS において以下のような操作におけるレスポンスが UFS と比較して悪いとのご指摘と認識しております。
  ・ 多数（千ファイル程度）をアーカイブした tar ファイルの展開
  (= 多数・小容量ファイルの一斉新規書き込み)

  PRIMECLUSTER GFS は、複数ノード間での共用ファイルシステムであり、ノード間での整合性維持のためファイルのメタ情報（ディレクトリ情報など）を MDS (Meta Data Server) というサーバ・プロセスにて管理します。tar ファイルの展開のように一度に多数のファイルを新規に作成するような操作では、MDS との通信が多発するため UFS と比較すると高いレスポンスが得られません。ノード間共用ファイルという機能の性格上、短期的に大きなレスポンス改善につながる対応は技術的に困難な状況であり、以下の対応につきご検討いただきますようお願いいたします。

  tar ファイルのような多数かつ小容量のファイルを扱うのは、プログラムのコンパイル（make） 時など一時的な操作と想定しております。

  make のような操作は、各ノードのローカルなファイルシステム（UFS）上で実施し、make したバイナリを GFS 上へ複写し複数ノードで共用するような運用をご検討願いたい。

<富士通：2006/2/17 第 5 回会合>

その他

(38) Parallelnavi のエンハンス状況を知りたい

【要望】

- 本 WG にて出された課題が契機となってエンハンスする機能を含めた Parallelnavi 全体のエンハンス内容（実施済み、計画）について知りたい。

<京大学情：2006/2/17 第 5 回会合>

【回答】

- Parallelnavi エンハンス一覧を、添付資料 38 に示します。

添付資料 38-1 を参照

<富士通：2006/2/17 第 5 回会合>

(39) 大規模 SMP クラスタ運用のユーザ事例を、普及すべき

【要望】

- 大規模 SMP クラスタにおける運用に工夫を重ねるユーザの事例は、既に大規模 SMP クラスタを導入されている機関は勿論のこと、これから導入予定の機関にも参考になるのではないか。ユーザ事例を広げるべきではないか。

〈WG 総意：2006/5/30 第 6 回会合〉

【回答】

- JAXA の大規模 SMP クラスタにおける運用の課題と工夫を、添付資料 39 に示します。

添付資料 39-1 を参照

〈JAXA：2006/9/15 第 7 回会合〉

(40) HPC Portal への要望

【要望】

- SSL に対応した際、ファイルのアップロード遅延が発生。OS の違い（Windows2000 と WindowsXP）によりアップロード時間が大幅に違った。（10 倍程度）
- パスワード変更機能を追加してほしい。

〈以上 2 点、JAEA：2005/10/07 第 4 回会合〉

- 例えばジョブ投入の際にマニュアルを参照したい場合、一旦メインに戻らないとならないのは不便。マニュアルのトップページを、別ウィンドウで表示する等の手段は?
- 「ファイル操作」で日本語のディレクトリ名が作成できてしまうのはよろしくない。
- 自分の好きなエディタを使うことができれば便利。（例：Windows 版 emacs）

〈以上 3 点、名大基盤：2005/10/07 第 4 回会合〉

【回答】

- 添付資料 40-1 に示します。

添付資料 40-1 を参照

〈富士通：2006/10/27〉

(41) 富士通の WWSite 構築サービスの商品体系

【要望】

- HPC Portal を含む、富士通製 R&D サーバ WWSite 構築サービスの商品体系を教えてほしい。

〈京大メディア：2006/5/30 第 6 回会合〉

【回答】

- 添付資料 41-1 に示します。

  添付資料 41-1 を参照

〈富士通：2006/10/27〉

以上

第5回SS研大規模SMP運用WG資料

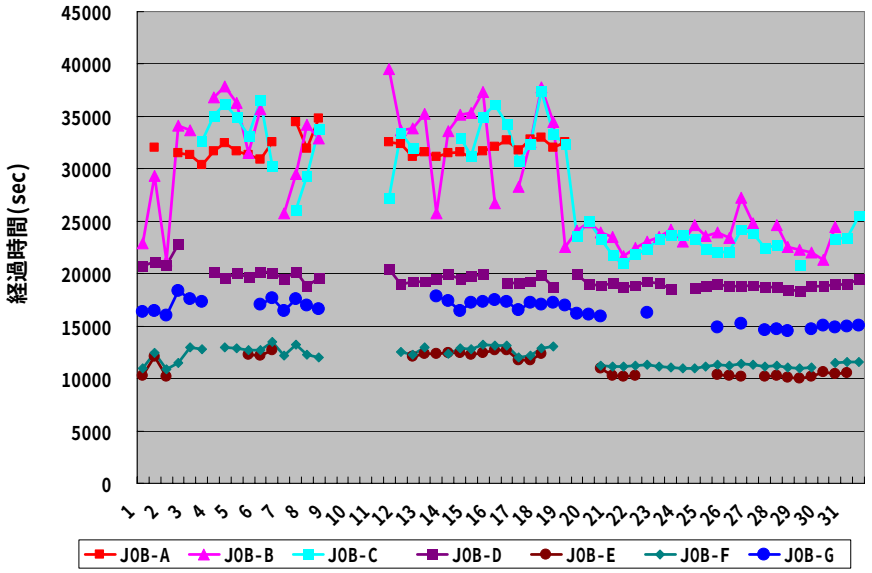
添付資料2-1
実行時間のブレ問題の分析結果

2006年2月17日

宇宙航空研究開発機構

ブレの傾向 (特定ジョブの影響)

- 環境 32CPU・256GBmem/node×55node
- JOB-Aが走ると、JOB-B、JOB-Cが影響を受ける



JOB-B

約1.63倍

JOB-C

約1.40倍

ブレの原因について

大規模SMPで競合するポイントは以下2点

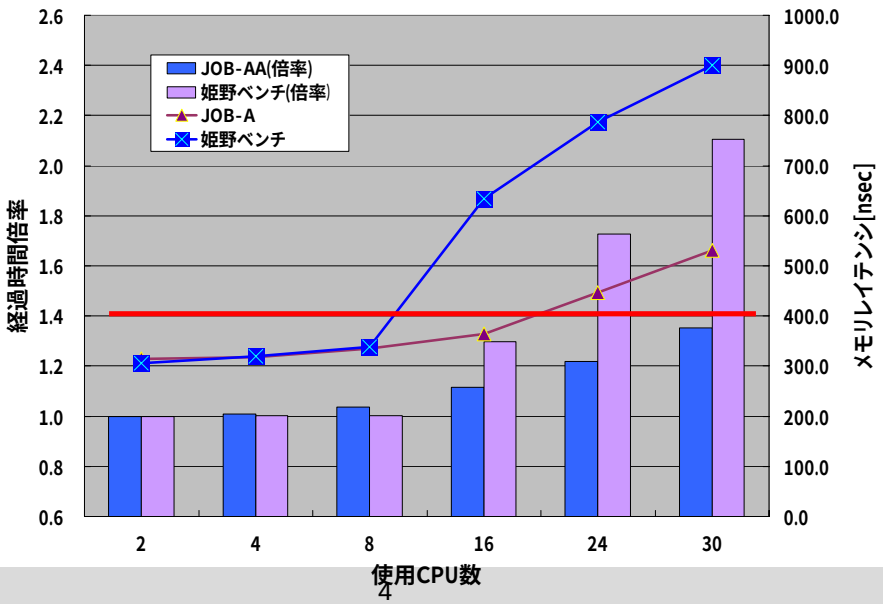
- メモリ性能ネック
- DTU転送性能ネック

※　IOも競合するが、一般にジョブ実行時間に対する時間比率が小さいため、今回は考慮しない。

JAXAでは、当初メモリ負荷の競合で性能がブレていると考えていたが、その後の調査により、DTU転送負荷の影響が支配的であることが判明

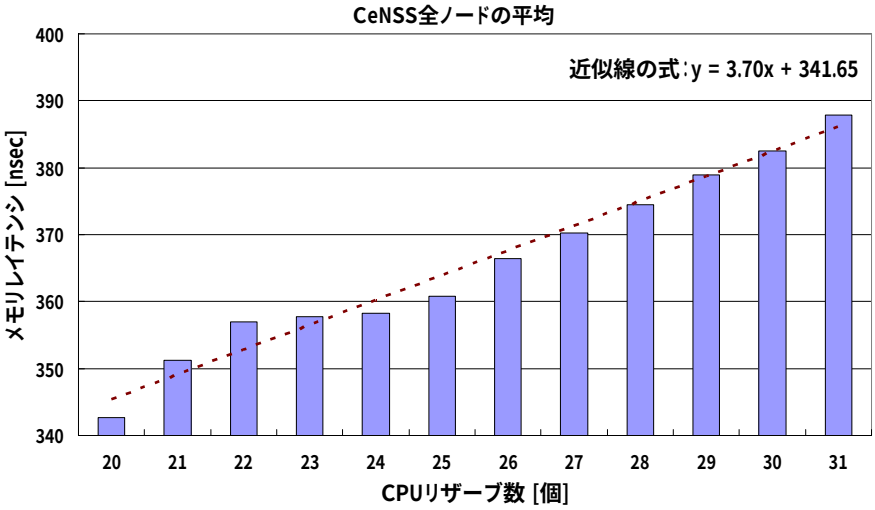
メモリー性能ネックの検証①

- メモリ高負荷ジョブ　JOB-A(JAXAｺｰﾄﾞ)、姫野ベンチで影響度を検証
- 400[nsec]を超えるとブレが顕著に現れる



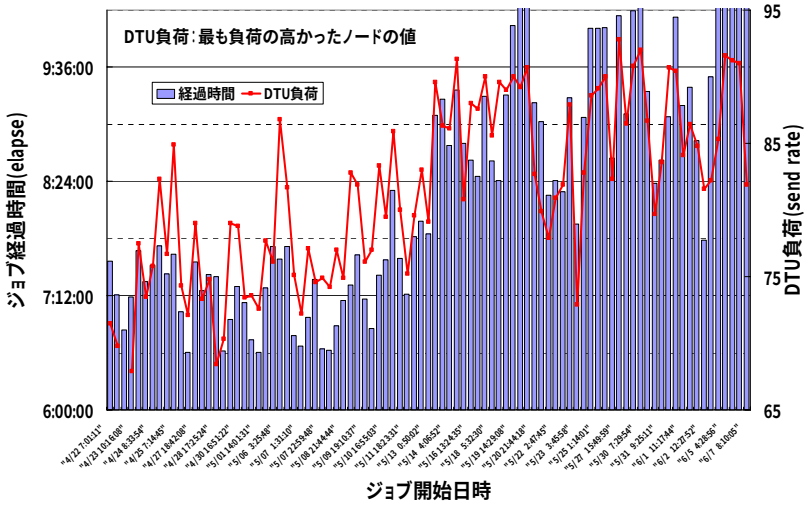
メモリー性能ネックの検証②

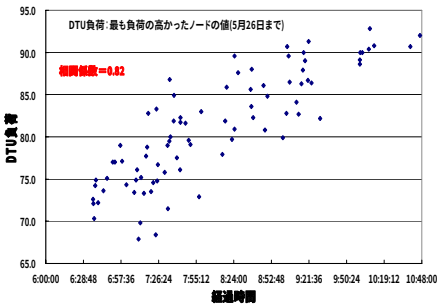
- 実運用では、メモリ負荷は通常390[nsec]以下であり、メモリ負荷の影響は低い
- このままスケールすると64CPUノードでは影響が出てくると思われる



DTU転送ネックの検証

- DTU転送負荷の高いJOB-Aを調査
- DTU負荷と経過時間の伸びには相関関係がある (相関係数0.82)





まとめ

- JAXAでは、メモリ負荷よりDTU転送負荷がブレの主な原因
- DTU転送負荷については、以下を継続中
  - DTU転送負荷をかけているジョブの特定方法の検討（DTU利用度合い）
  - 特定したジョブの分析、対応検討
    - DTU転送サイズの調整

第5回SS研大規模SMP運用WG資料

添付資料2-2
実行時間のブレ問題の分析結果

2006年2月17日

日本原子力研究開発機構
富士通株式会社

測定環境

◆実行測定環境

測定日時　：　2004/10/10 10:00 ～ 2004/10/12 1:00
測定マシン：　HPC2500(128CPU/ノード,占有利用)
測定は1ノードだけを使用した。
プログラムはParanavi2.4で再コンパイルしたモジュールを使用した。

◆実行測定内容

プログラム(a) 8スレッド並列ジョブ 同時多重実行(ユーザ殿提供プログラム)
① 単体測定　(8×1=8CPU使用)
② 2多重実行測定　(8×2=16CPU使用)
③ 4多重実行測定　(8×4=32CPU使用)
④ 8多重実行測定　(8×8=64CPU使用)

プログラム(b) 8スレッド並列ジョブ 同時多重実行(ユーザ殿提供プログラム)
① 単体測定　(8×1=8CPU使用)
② 2多重実行測定　(8×2=16CPU使用)
③ 4多重実行測定　(8×4=32CPU使用)
④ 8多重実行測定　(8×8=64CPU使用)

◆実行測定結果(次頁参照)

採取データ
- Elapsed時間　(各ジョブ単位)
- 実CPU時間　(各ジョブ単位)
- 実効CPU時間　(各ジョブ単位)

実行測定データ

◆プログラム(a) 8スレッド並列ジョブ

	時	分	秒	秒換算	ratio	実CPU:m秒	ratio	実効CPU:m秒	ratio
単体	0	12	53.24	773.24	1.00	4779120.00	1.00	3090900.00	1.00
多重度2	0	13	48.20	828.20	1.07	5255670.00	1.10	3105990.00	1.00
多重度2	0	13	54.48	834.48	1.08	5274160.00	1.10	3107760.00	1.01
多重度4	0	15	8.40	908.40	1.17	5817310.00	1.22	3128020.00	1.01
多重度4	0	15	35.74	935.74	1.21	6005420.00	1.26	3128660.00	1.01
多重度4	0	15	30.23	930.23	1.20	6005180.00	1.26	3129750.00	1.01
多重度4	0	15	8.68	908.68	1.18	5833830.00	1.22	3131280.00	1.01
多重度8	0	15	33.43	933.43	1.21	5892650.00	1.23	3130070.00	1.01
多重度8	0	17	36.02	1056.02	1.37	6860560.00	1.44	3136570.00	1.01
多重度8	0	17	35.54	1055.54	1.37	6859220.00	1.44	3139650.00	1.02
多重度8	0	17	42.94	1062.94	1.37	6915950.00	1.45	3141520.00	1.02
多重度8	0	17	40.58	1060.58	1.37	6916530.00	1.45	3140690.00	1.02
多重度8	0	19	30.24	1170.24	1.51	7800380.00	1.63	3154530.00	1.02
多重度8	0	19	36.55	1176.55	1.52	7816930.00	1.64	3154810.00	1.02
多重度8	0	19	31.18	1171.18	1.51	7820390.00	1.64	3154380.00	1.02

◆プログラム(b) 8スレッド並列ジョブ

	時	分	秒	秒換算	ratio	実CPU:m秒	ratio	実効CPU:m秒	ratio
単体	0	5	31.16	331.16	1.00	639990.00	1.00	433940.00	1.00
多重度2	0	5	31.11	331.11	1.00	639870.00	1.00	433000.00	1.00
多重度2	0	5	27.83	327.83	0.99	638970.00	1.00	431760.00	0.99
多重度4	0	5	32.91	332.91	1.01	649250.00	1.01	432230.00	1.00
多重度4	0	5	32.46	332.46	1.00	649560.00	1.01	431730.00	0.99
多重度4	0	5	32.67	332.67	1.00	654100.00	1.02	432130.00	1.00
多重度4	0	5	31.95	331.95	1.00	651410.00	1.02	432100.00	1.00
多重度8	0	5	36.22	336.22	1.02	660470.00	1.03	432820.00	1.00
多重度8	0	5	34.42	334.42	1.01	662390.00	1.04	432060.00	1.00
多重度8	0	5	36.94	336.94	1.02	667540.00	1.04	432410.00	1.00
多重度8	0	5	34.72	334.72	1.01	661470.00	1.03	432070.00	1.00
多重度8	0	5	35.59	335.59	1.01	652790.00	1.02	432480.00	1.00
多重度8	0	5	38.12	338.12	1.02	666370.00	1.04	433200.00	1.00
多重度8	0	5	36.18	336.18	1.02	663260.00	1.04	432130.00	1.00
多重度8	0	5	35.89	335.89	1.01	655500.00	1.02	432290.00	1.00

Elapsed時間　　実CPU時間　　実効CPU時間

(注意) 実CPU、実効CPU時間は8スレッドの合算値

考察

- 2004年8月測定時と比較してプログラム(a)のバラつき具合は多少小さくなった。

  2004年8月測定結果：　8多重時に単体時と比べ1.67倍(Elapsed)
  2004年10月：　　8多重時に単体時と比べ1.52倍(Elapsed)
  →コンパイラ (Paranavi2.3→2.4への改善効果)

- 実効CPU時間、及び採取したプロファイラ情報(メモリスループット)から、ジョブ実行時間のバラつきの原因はメモリアクセス競合による影響と判断できる。

  実効CPU時間はプログラム(a) 8多重時においても安定した値を示す(最大1.02倍)
  プログラム(a)のL2-miss、メモリスループット値が大きい

第5回SS研大規模SMP運用WG資料

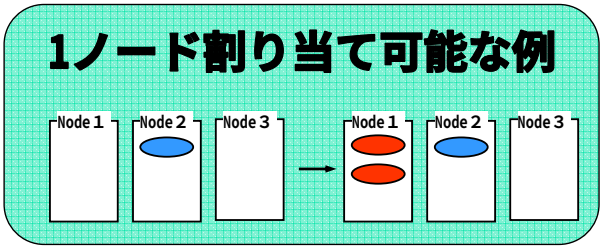
添付資料5-1
１ジョブを同一ノードに割り当てる方法

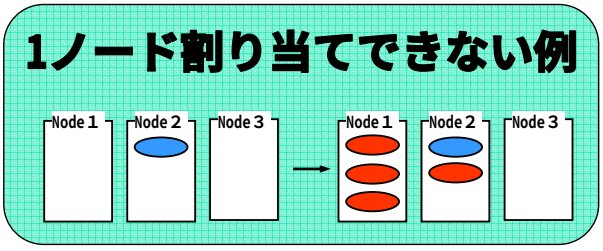
2006年2月17日

富士通株式会社
Linuxソフトウェア開発統括部

１ジョブを同一ノードに割り当てる方法

- 1ジョブの同一ノードへの割り当ては、「PACK割り当て方式」によりサポート済
  - 割り当て可能プロセス数が多いノードを選択(　空きCPUの多い順)
  - できるだけ1ノードへジョブを収める
  - 資源不足時などは複数ノードに跨る





第5回SS研大規模SMP運用WG資料

添付資料6-1
CPU利用台数によるジョブの優先度制御

2006年2月17日

富士通株式会社

Linuxソフトウェア開発統括部

優先度による実行ジョブの決定

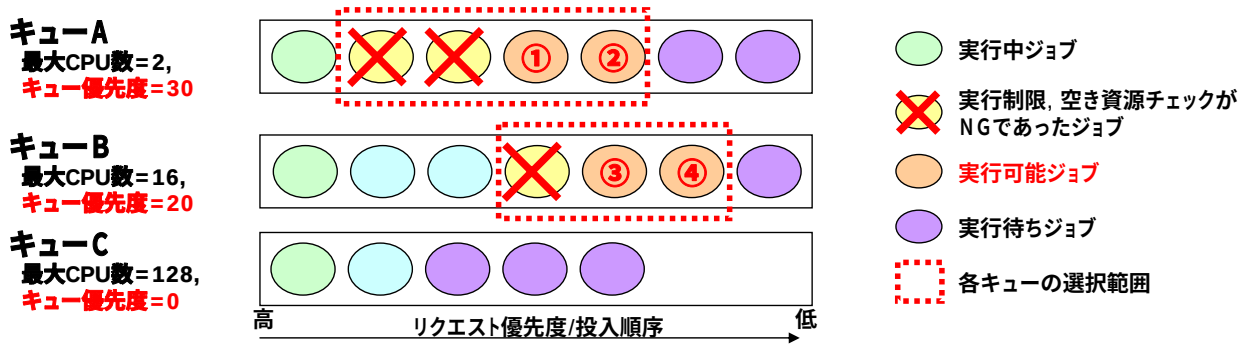
FUJITSU

実行可能ジョブが複数存在する場合、優先度により実行ジョブを選択

- 実行可能なジョブが複数ある場合、以下の順序で優先度を評価し、実行ジョブを決定

キューの優先度 > リクエスト(ジョブ)の優先度 > 投入順番

例：CPU数が少ないジョブを優先実行 (①～④の順に実行)



資源の重み評価による実行ジョブの決定

FUJITSU

ジョブが要求する資源の重みを評価して実行ジョブを決定

- ジョブ実行に必要な資源をファクタとし、資源毎の係数（ランク）と指定した資源量に応じた値（ファクタ値）を定義
- 以下の評価式により算出された評価値が最大のジョブを最優先に実行

$$\text{評価値} = \sum_{i=1}^{n} (\text{ファクタ}\ i\ \text{のランク} \times \text{ファクタ}\ i\ \text{の値})$$

例：CPU数が少なくラージページメモリ量の少ないジョブを優先実行

ファクタ	ランク	指定値	ファクタ値	指定値	ファクタ値
CPU数	10	～4	100	5～	10
ラージページメモリ量	5	～1024	100	1025～	10

第5回SS研大規模SMP運用WG資料

添付資料9-1
ジョブスケジューリングの定義
(空きノードを作るスケジューリング)

2006年2月17日

富士通株式会社
Linuxソフトウェア開発統括部

ノード集中割り当て機能について

ノード集中割り当て機能を提供予定

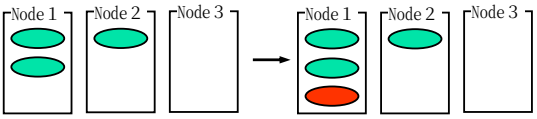
- **機能概要**
  できる限り空きCPUが少ないノードに詰めて割当てる機能
- **定義方法**
  NQS-JS のシステム、キュー動作環境定義文に以下を追加
  ーload policy = concentration
  JOBを1つのノードにできる限り詰め込む (必要最低限のノード利用)
  ーload policy = balancing (デフォルト)
  JOBを空きCPUが多いノードに割当てる (性能優先)

ノード集中割り当てポリシーご説明(1)

1) ジョブ集中割り当て：concentration とAbsolutely PACK 指定

ー空きCPU数の少ないNodeを選択します。

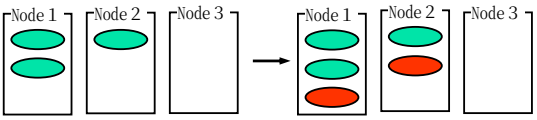
ーすべてのプロセスを1 つのNode に割り当てます。割り当てられない場合は、実行可能リクエストとして選択しません。



2) ジョブ集中割り当て：concentration と PACK 指定

ー空きCPU数の少ないNodeを選択します。

ーすべてのプロセスをできる限り少数のNode に割り当てます。割り当てられない場合は、他のNode にも割り当て、実行可能リクエストとして選択します。

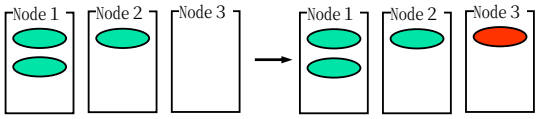


ノード集中割り当てポリシーご説明(2)

3) ジョブ分散割り当て：balancingとAbsolutely UNPACK指定

ー空きCPU数の多いNodeを選択します。

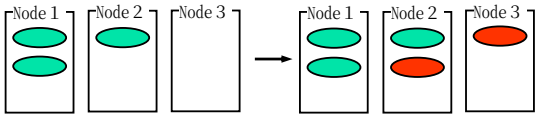
ー1 プロセス = 1 Node で分散させて割り当てます。プロセス数が割り当て可能Node 数より多い場合は、実行可能リクエストとして選択しません。



4) ジョブ分散割り当て：balancingと UNPACK指定

ー空きCPU数の多いNodeを選択します。

ー1 プロセス = 1 Node で分散させて割り当てます。プロセス数が割り当て可能Node 数より多い場合は、1 Node に2 つ以上のプロセスを割り当て、実行可能リクエストとして選択します。



添付資料11－1

ＣｅＮＳＳ運用における工夫 第１.０版

2005 年 2 月 16 日
ＪＡＸＡ　情報技術開発共同センタ

ＪＡＸＡ　ＣｅＮＳＳ運用において工夫した点のうち、機能として開発（ツール化）したものを以下にまとめる。本日、網掛の部分をデモさせていただきます。

No.	分類	名称	開発元	機能・目的	効果・備考
1.	利用	ＮＳコマンド	JAXA	利用者の利便性向上のため、簡単にジョブを投入する機構やジョブ状態の整形出力などのコマンド群を NS コマンドとして提供。 例えば以下。 1)actjob sysrscstat、rscsetstat *1 の情報を 80x24 の画面で参照可能な情報に整形出力する。 2)lsize LPG を考慮したセクションバイト数の出力し、プログラムの使用メモリの概算がわかります。ただし、スタック、動的配列などは含まれない。 3)ns-shell qsub スクリプトを簡素化。Shell を知らなくてもジョブ投入を可能とする。	システムの利用性向上。 （簡単な NS コマンドだけ覚えれば、システムが使える）。 ns-shell を除き他サイトでも容易に流用可能。 *1：複数ユーザから同時実行された場合の効率を考慮し、sysrscstat、rscsetstat の情報は定期的(1min)にログ保存し、その情報を整形している。
2.	利用	ＷＡＮＳ	JAXA	Web Acess to NS の略。 Web ベースで特別なクライアントを必要としない統一的な GUI で、JAXA 内外からのでシステム利用を可能とする。 ジョブ投入／ファイル操作／簡易可視化など	場所を問わず、システムの利用が可能。 海外からの利用実績もある。 他サイトで流用するにはカスタマイズが必要。
3.	利用	i－ＷＡＮＳ	JAXA	WANS のサブセットとして、携帯電話からシステム利用を可能とする。 現状、i-mode のみ対応。	ちょっとした空き時間に場所を問わず、システムの利用が可能。 他サイトで流用するにはカスタマイズが必要。
4.	利用	コンパイラ 世代管理	JAXA	コンパイル環境について、新機能重視なら最新版、安定性重視なら１世代前の版、というようにユーザがコンパイラ選択できる環境を提供する。	安全なコンパイラバージョンアップが可能。 他サイトでも容易に流用可能。
5.	利用	ユーザ フォーラム	JAXA FJ	利用者のスキル向上とそれによる稼働率向上を目指し、利用者へ定期的に、チューニング、ツールの使い方など、ノウハウを情報提供する。	利用者のスキル向上。利用者同士の交流促進。 他サイトで流用するにはカスタマイズが必要。

No.	分類	名称	開発元	機能・目的	効果・備考
6.	利用	ＴＵＴＲＯ	JAXA FJ	TUning TRaining Online sysytem の略。 利用者のスキル向上とそれによる稼働率向上を目指し、ユーザフォーラムの内容やその他有益な教育を、e-lerning（ストリーミング）として提供。 富士通製 INTERNET NAVIGWARE を利用。	利用者のスキル向上。いつでも都合の良いときに利用可能。 他サイトで流用するにはカスタマイズが必要。
7.	性能	プロファイラ診断ツール	JAXA	ジョブの規模が大きくなるとプロファイラの出力情報が大きくなり、解析するのに時間、手間がかかる。そのため、プロファイラの診断結果を分析し、問題箇所を指摘する機能を実現する。	開発中。
8.	性能	高速ｃｐ	JAXA OSS	通常の cp コマンドは io 長が小さく(8K,256K,etc)、SRFS に適していない。そのため、ファイルシステムと認識し、最適な IO 長を選択する高速な cp コマンドを実現する。 GNU の fileutil を改造。	SRFS 間でのコピー性能が１１倍（23.4MB/s→258.9MB/s）に改善。 他サイトで流用するにはカスタマイズが必要。
9.	性能	高速 可視化連携	JAXA FJ	SGI と GSN（HIPPI6400）接続された環境において、リアルタイム可視化のための高速データ転送機構（VisLINK に追加)、高速なデータ共用（STF 機構）を実現する。	PRIMEPOWER HPC2500 と SGI Onyx3400 間において、500MB/s のデータ転送性能を達成。ｃｐ コマンドに適用し、、100MB/s のコピー性能を達成。 他サイトで流用するにはカスタマイズが必要。
10.	ジョブ	ＮＳＪＳ	JAXA	柔軟なジョブスケジューリングと資源割当を行い、稼働率の向上、適切なターンアラウンドを実現する。 NQS 出口を利用して以下の機能を実現。 実行順序保障／マルチブロック／概括並列／プロジェクト管理／ プレステージ／年間カレンダ／運転制御など	センタ主導による柔軟なジョブスケジューリングを実現し、稼働率を向上。 他サイトで流用するにはカスタマイズが必要。
11.	管理	ＮＳLDAP	JAXA OSS	LDAP によるユーザ情報の一元管理を行うことにより、運用管理コストを削減する。 OSS の OpenLDAP を利用。	一般に行われる UID 管理だけではなく、失効管理、quota 情報、使用期間、課金情報、住所、連絡先を一元管理し、かつそれらをＧＵＩ操作可能とし管理業務を効率化。 他サイトで流用するにはカスタマイズが必要。

No.	分類	名称	開発元	機能・目的	効果・備考
12.	管理	NS アクション, 管理	JAXA FJ OSS	センタで取り扱う障害、ＱＡ，要望の情報を、そのやりとり、状態含めてＷｅｂ UI にて操作可能なデーターベース化し、運用管理コストを削減する。 いわゆるバグトラッキングシステム、リクエストトラッカー。 OSS の RRDtool、PHP 等を利用	記入もれや間違い、確認忘れをなくし、対応をスピードアップ。 他サイトで流用するにはカスタマイズが必要。
13.	管理	CMS	JAXA FJ	CeNSS Monitoring System の略。ジョブの割当状況や稼動情報を容易に把握するため、Ｗｅｂベースでジョブ状態を表示する。用途に応じて、5 種類開発。	稼動状況の容易な把握、問題点の早期発見が可能。 ユーザへの運用状況の提供。 イベントなどでの展示。 他サイトで流用するにはカスタマイズが必要。
14.	管理	ログ監視	JAXA OSS	/var/adm/messages 等に出力される重要なメッセージを監視し、即時通報（メール、オペコール)、即時アクション（コマンド発行）を行う。 OSS の Swatch を利用。	重大障害の早期発見、早期対応が可能。 他サイトでも容易に流用可能。
15.	管理	unyo_check	JAXA FJ	保守前後での確認すべき運用状態などのチェックを自動化（スクリプト化）し、手順ミス、考慮漏れを防ぐ。	環境の戻し忘れなど、保守作業後の単純ミスを撲滅。 他サイトで流用するにはカスタマイズが必要。

以上

第5回SS研大規模SMP運用WG資料

添付資料14-1
実行中ジョブの制限値変更について（検討結果）

2006年2月17日

富士通株式会社
Linuxソフトウェア開発統括部

実行中ジョブの制限値変更

実行中ジョブに対するジョブ制限値の変更機能として、下記のqmodifyコマンドを提供予定です。

－記述形式

qmodify [-u ユーザー名] –cp 経過時間　-lT CPU時間　リクエスト識別子

－機能概要

qmodify は、指定されたリクエスト識別子に対応するバッチリクエストの制限値であり、原則としてバッチリクエストの所有者だけが変更可能です。
例外として、システム管理者、NQS マネージャー/オペレータは、自らが投入していないバッチリクエストでも制限値の変更可能です。

資源	変更可能なリクエストの状態	変更可能な制限値の範囲		対応コンポ
		リクエストの所有者	管理者	
-cp (経過時間)	QUEUED, WAITING, HOLDING, RUNNING, RUNNING*	変更前の値～キューの値 (増加のみ可能)	0～2147483647秒 (増減可能、キューの最大を超える)	NQS
-lT (CPU 時間)				NQS, NQS-JS, 資源管理,CPU資源管理

第5回SS研大規模SMP運用WG資料

添付資料15-1
Shared-DTU機能の評価結果について

2006年2月17日

宇宙航空研究開発機構

UDTU vs shared UDTU (MPI)

- 通信が少ないMPIジョブ（単体実行）は、ブレない。
- 通信時間比率(*)は0.35～1.44%

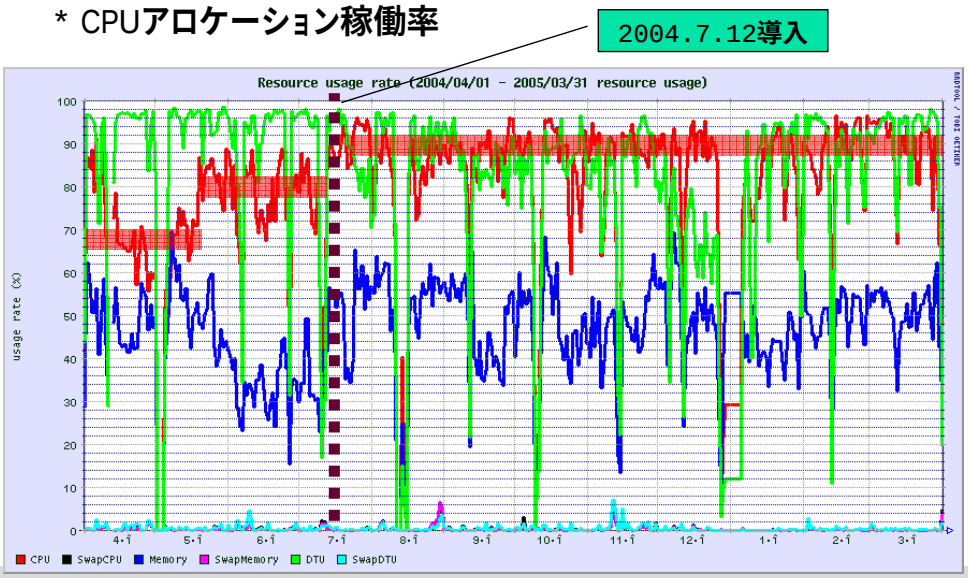
*プロファイラで求めた通信時間比率（同期/通信待ちも含む）

JOB	PROC NO	実行時間(sec)			通信時間比率(%)
		UDTU	ShUDTU	比率(%)	
JOB-A	8	235.3	236.5	100.5	0.35
JOB-A'	27	246.5	246.6	100.0	0.66
JOB-B	8	90.7	90.9	100.2	0.89
JOB-C	8	76.1	76.1	100.0	1.14

Shared UDTUの効果

- 稼働率の向上 (60-80%⇒90%以上)
  - UDTU枯渇によりCPUが使えない状況を解決
- 05年は年間稼働率(*)90%以上を達成！

  * CPUアロケーション稼働率



まとめ

- 通信の少ないMPIジョブは、Shared UDTUを使用しても大きくブレない。
- Shared UDTUの導入により、稼働率の大幅な改善を実現

第5回SS研大規模SMP運用WG資料

添付資料15-2
Shared-DTU機能の評価結果について

2006年2月17日

日本原子力研究開発機構
富士通株式会社

測定環境

◆実行測定環境

測定日時　：　2004/10/10 10:00 ～ 2004/10/12 1:00
測定マシン：　HPC2500(128CPU/ノード,占有利用)

- UDTU, Shared-UDTUとも3ノードに跨いで実行
- Shared-UDTUでの実行測定はDTU×1のみ使用

測定プログラム

◆実行測定内容

プログラム(a) 23プロセス並列ジョブ (ユーザ殿提供プログラム)
① UDTU 単体実行測定
② Shared-UDTU 単体実行測定
③ Largepageコピー 単体実行測定(ノード内実行)

プログラム(b) 24プロセス並列ジョブ (ユーザ殿提供プログラム)
① UDTU 単体実行測定
② Shared-UDTU 単体実行測定

プログラム(c) 8プロセス並列ジョブ (ユーザ殿提供プログラム)
① UDTU 単体実行測定
② Shared-UDTU 単体実行測定

プログラム(d) 8プロセス並列ジョブ (ユーザ殿提供プログラム)
① UDTU 単体実行測定
② Shared-UDTU 単体実行測定

プログラム(e) 8プロセス並列ジョブ (ユーザ殿提供プログラム)
① UDTU 単体実行測定
② Shared-UDTU 単体実行測定

測定データ

プログラム(a)
23並列

	時	分	秒	秒換算	ratio
単体 (UDTU)	0	9	27.37	567.37	1.00
単体 (SH-UDTU)	0	28	32.04	1712.04	3.02
単体 (LPCOPY)	0	13	29.60	809.60	1.43

UDTU実行時のMPI通信コスト: 4.47% (プロファイラ解析)

プログラム(b)
24並列

	時	分	秒	秒換算	ratio
単体 (UDTU)	0	10	20.73	620.73	1.00
単体 (SH-UDTU)	0	14	46.65	886.65	1.43

UDTU実行時のMPI通信コスト: 2.49% (プロファイラ解析)

プログラム(c)
8並列

	時	分	秒	秒換算	ratio
単体 (UDTU)	0	9	32.66	572.66	1.00
単体 (SH-UDTU)	0	9	53.02	593.02	1.04

UDTU実行時のMPI通信コスト: 0.01% (プロファイラ解析)

プログラム(d)
8並列

	時	分	秒	秒換算	ratio
単体 (UDTU)	0	5	11.75	311.75	1.00
単体 (SH-UDTU)	0	5	37.87	337.87	1.08

UDTU実行時のMPI通信コスト: 0.68% (プロファイラ解析)

プログラム(e)
8並列

	時	分	秒	秒換算	ratio
単体 (UDTU)	0	4	32.83	272.83	1.00
単体 (SH-UDTU)	0	7	29.70	449.70	1.65

UDTU実行時のMPI通信コスト: 1.08% (プロファイラ解析)

FUJITSU THE POSSIBILITIES ARE INFINITE

http://www.fujitsu.com/

第5回SS研大規模SMP運用WG資料

添付資料16-1
Share運用での課題に対する回答

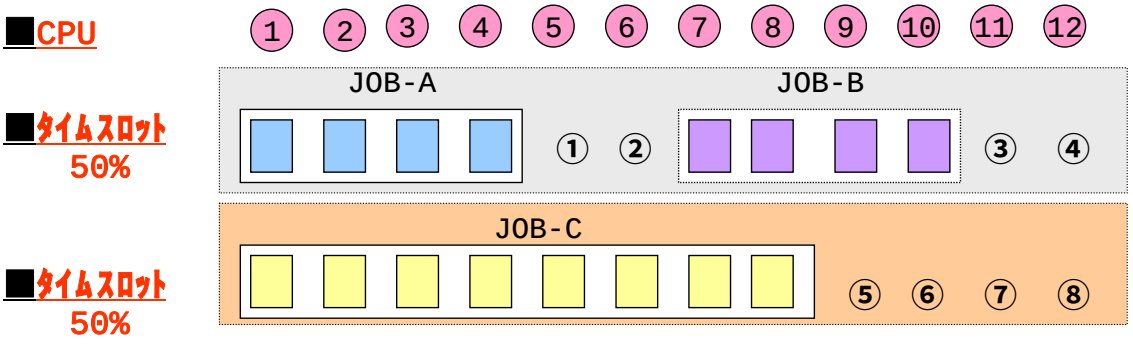
2006年2月17日

富士通株式会社

SHARE運用での資源割当

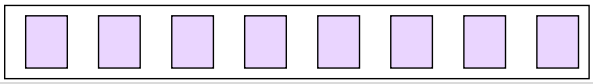
■使用目的：高並列JOBの場合、CPU使用率が低いケースがあり、
その空き時間を有効利用したい。

■課題：　並列JOBをSHAREで運用するとタイムスロットの
フラグメンテーションで期待していた資源割当てができない



★JOB-AorJOB-Bをタイムスロット1に移動できれば、JOB-Dが実行可能になる

次の実行待ちJOB　JOB-D

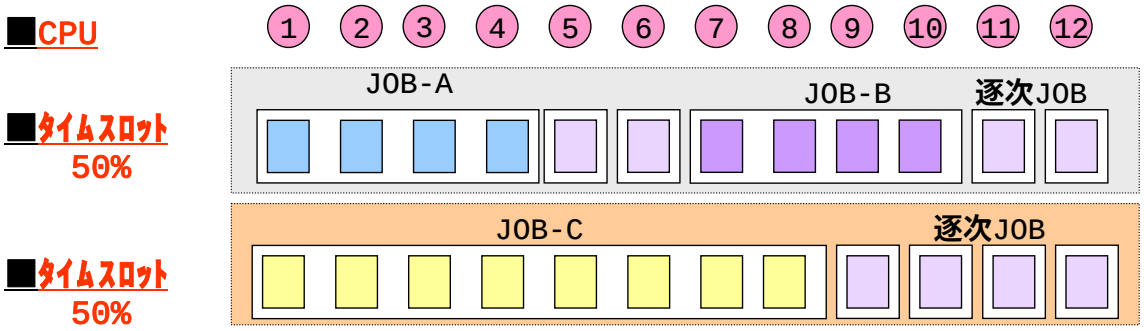


SHARE運用の推奨モデル

■推奨：並列JOBと逐次JOBのSHARE運用
- フラグメントで並列JOBが割当てられない部分を逐次JOBで埋める方式。
- 並列JOBがSLEEP中、スロット外でも逐次JOBは走行可能。

★タイムスロット間で自動的にJOBマイグレートは困難

■例：1並列JOBのCPU配分率：50%、1逐次JOBのCPU配分率：50%



FUJITSU THE POSSIBILITIES ARE INFINITE

http://www.fujitsu.com/

第5回SS研大規模SMP運用WG資料

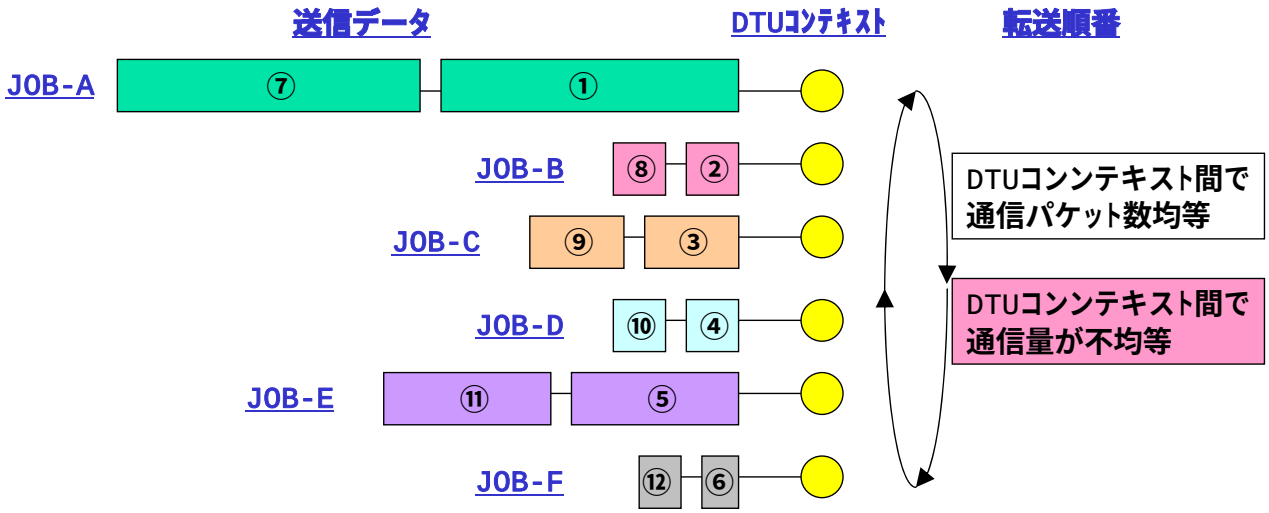
添付資料31-1
特定のジョブが他ジョブの経過時間に影響する問題について（分析結果と対策）

2006年2月17日

富士通株式会社
Linuxソフトウェア開発統括部

課題1：DTU競合による性能劣化

大IOサイズ/大容量転送JOBが存在すると小IOサイズ転送JOBの実行性能に大きな影響を与える(DTU競合)。

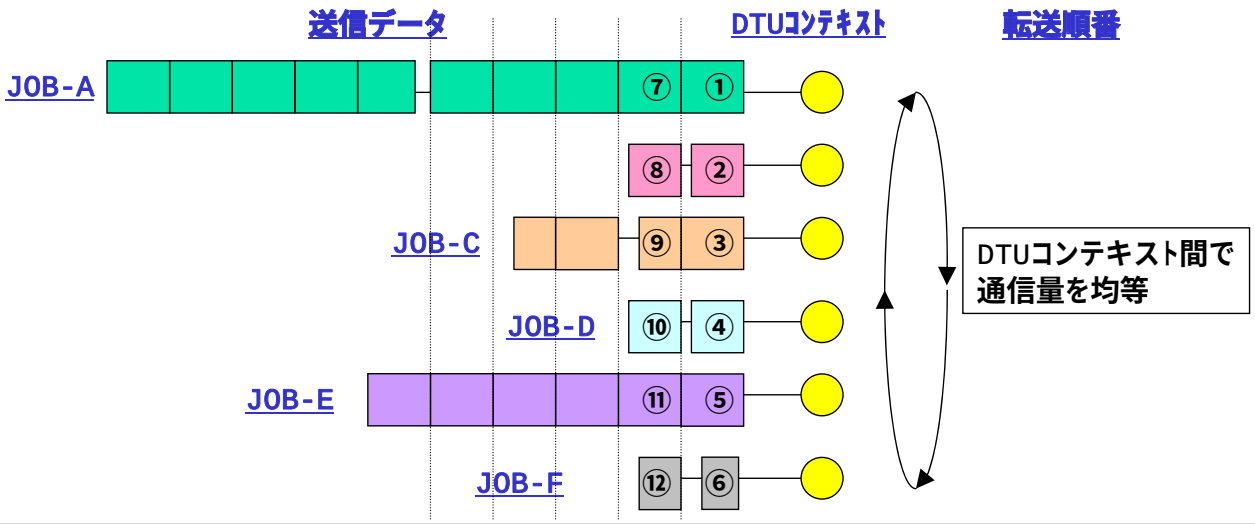


DTU通信量で均等化の実現

1回の転送単位を制限：通信データ量の均等を実現

環境変数：GMP_ONESIDED_SIZE=1MB(省略値：制限なし)

- XPF: /etc/opt/FJSVxpf/xpfexec.conf
- MPI: /etc/opt/FJSVmpi2/mpiexec.conf



今後のインタコネクト開発に向けて

どういう仕様のインタコネクトがいいか？

要件は

- JOB全体のスループットが最大になる
- JOBの実行時間のばらつきが少ない

現状は

- 1回の転送サイズを一律に小さくすると
  - トータルスループットが下がる。
    - ・特に転送サイズが大きいJOBが走行する環境では
- 1回の転送サイズを一律に大きくすると
  - 転送サイズが小さなJOBの実行時間のばらつきが激しくなる。

今後のインタコネクトは？

- QOSの導入しても、最低バウンド幅を確保できるだけで、バラツキはでるが、バラツキの幅は小さくなるか。
  空いていてもQOSしか使わない仕様ならバラツキはなくなる。。。。

第5回SS研大規模SMP運用WG資料

添付資料31-2
特定のジョブが他ジョブの経過時間に影響する問題の検証結果

2006年2月17日

宇宙航空研究開発機構

DTU通信量の均等化の効果検証

検証モデル

以下のJOB-A, JOB-B（JAXA様JOBを利用）の同時走行により効果を検証
■JOB-A ： DTU転送IOサイズ大のJOB
96プロセス/4スレッド(XPFortran)
235MB/秒、541回/秒の通信、最大DTU転送IO単位：203MB
■JOB-B ： DTU転送IOサイズ小のJOB
50プロセスMPI通信、3MB/秒、668回/秒の通信

検証結果（DTU転送IOサイズの調整）

JOB-A走行中のJOB-Bの実行時間：30%ダウン→15%ダウンに改善
■JOB-A ： DTU転送IOサイズを小さくした影響は5%ダウン
■JOB-B ： 混在時、JOB-Bの性能は15%向上

JOB	単独走行(秒)		JOBA, B混在	
	分割なし	分割=1MB	分割なし	分割=1MB
A	43.56	45.85(▲5.0%)		
B	160.0		205.67 (▲30%)	183.67(▲15%)

トータルスループットの検証

検証モデル

JOB-Aを含む大規模転送ジョブグループと、JOB-Bを含む小規模転送ジョブグループを混在した同時走行により効果を検証

検証結果（DTU転送IOサイズのデフォルト値サーベイ）

DTU転送IOサイズの調整による総スループット改善は困難。

No.	DTU転送IOサイズ	スループット（秒）		
		大規模	小規模	全体
1	指定なし(1GB)	1784	1746	1784
2	16MB	1886	1634	1886
3	8MB	1887	1634	1887
4	4MB	1931	1606	1931
5	1MB	1942	1592	1942

まとめ

- DTU転送転送サイズの調整により、ブレの軽減が可能な場合がある。
- ただし、DTU転送サイズのデフォルト値を変更すると、総スループット性能は悪くなるため、デフォルト値設定は困難。

添付資料 33-1

性能阻害箇所の検出ツール (宇宙航空研究開発機構様開発)

ユーザコマンド　　profchk(1)

【名前】

profchk - プロファイラ出力情報から性能阻害箇所を検出する

【形式】

/opt/NS/bin/profchk [-limit n] [-e] inputfile

【機能説明】

プロファイラで採取した情報から、性能阻害箇所と想定原因を
自動検出します。

【オプション】　以下にオプションを示します。

-limit n　性能阻害箇所および想定原因の情報を、検出順に n 個まで出力する。
　　省略した場合は n=5 となる。

-e　　メッセージを英語で出力する。省略した場合は日本語で出力する。

inputfile　対象となるプロファイラ出力情報ファイル名を指定する(必須)。

【使用方法】

まず入力データとなるプロファイラ情報を、以下の手順で用意してください。

(1) 解析対象プログラムを f90ns でコンパイルする。

(2) 下記の環境変数を設定してプログラムを実行し、性能情報ファイル(DProf*,GProf*)を採取する。

```
TRT_ENV="PMP=on" ;export TRT_ENV
PROF_STATS=7 ;export PROF_STATS
PROF_PA=sta,cov ;export PROF_PA
```

(3) 生成された性能情報ファイルのいずれか一つを引数に指定して、下記のコマンドでテキスト形式
に変換する。

例) mpprof DProf_12345.000.pri > prof.out

変換後のテキストファイルを引数としてコマンドを起動すると、以下の情報が出力されます。

(1) プロセス並列実行数
(2) スレッド並列実行数
(3) Location：性能阻害発生ルーチン名および行番号
(4) Blocking factor：(3)の性能阻害要因
(5) Possible cause：(4)の想定される原因
(6) 検出した件数

(1)(2)(6)は全体情報で１回だけ出力、(3)(4)(5)は検出した項目ごとに通し番号付きで出力されます。

(実行例1) 一般的な出力例

```
$ profchk prof1.out
   プロセス数：4
   スレッド数：16
-----------------------------------------------------------
[0]Location : subA._OMP_1_ ( 48 - 414 )
[0]Blocking factor: 浮動小数点数の演算効率が 400MFlops 以下です。
  値=23.146000Mflops
[0]Possible cause : 浮動小数点演算が少ないようです。
-----------------------------------------------------------
[1]Location : subB._OMP_1_ ( 56 - 60 )
[1]Blocking factor: 浮動小数点数の演算効率が 400MFlops 以下です。
  値=10.024610Mflops
[1]Possible cause : 浮動小数点演算が少ないようです。
-----------------------------------------------------------
2 blocking factor found.
```

(実行例2) 性能阻害条件が検出されなかった場合

```
$ profchk prof2.out
プロセス数：4
スレッド数：16
0 blocking factor found.
```

[エラー]

本コマンドの実行中に異常が検出された場合、以下のメッセージが出力されます。

profchk:error: cannot read profiling data.：入力データの読み込み中にエラーが発生した

profchk:error: Not found XXXXX block.：入力データに必要なデータが含まれていない

profchk:error: profiling data is invalid.：入力データの内容に不整合が発生している

profchk:error: invalid argument.：コマンドに無効なオプションが指定された

profchk:error: internal error.：その他の原因で内部エラーが発生した

[注意事項]

1. 説明と異なる手順で作成したプロファイラ情報を使用した場合、必要なデータが無いためエラーと
なる場合があります。
2. 本コマンドの検出基準は性能問題チェックシートに準拠します。
詳細は NS ポータルのプログラムチューニングを参照下さい。
3. 実行コスト比率 5%以上のルーチンが検出対象となります。
4. 本コマンドの出力情報の精度はプロファイラ情報に依存します。

[関連項目]

mpprof(1)

性能問題チェックシート

下記にアプリケーションの性能問題をチェックする項目を列挙します。アプリケーションのプロファイラ情報を採取し、各項目をチェックすることで、アプリケーションに性能改善の余地があるかどうかが分かります。
なお、プロファイラの採取方法はこちらを参照してください。

性能上の問題箇所が見つかれば、CeNSSチューニングガイド を参照し、プログラムの改善を行ってください。

(1)Sampling Cost

【チェック内容】
サブルーチンのSampling Costが10%を越えている。

【説明】
基本的にチューニングは高コストルーチンから着手します。高コストのルーチンほど、性能を改善することで得られる効果が大きいためです。
Sampling Costをチェックし、10%以上を占めているサブルーチンについてはチューニングを検討してください。

Sampling Cost

Samples	% Run	Barrier	Start	End	
1.683700e+04	**35.77**	0.000000e+00	18	269	**aaa_**
6.117000e+03	**12.99**	0.000000e+00	80	238	**bbb_**
3.474000e+03	7.38	0.000000e+00	20	72	ccc_
2.138000e+03	4.54	0.000000e+00	111	256	ddd_
1.894000e+03	4.02	0.000000e+00	269	342	eee_
1.697000e+03	3.60	0.000000e+00	18	78	fff_

(2)L2-miss率

【チェック内容】
L2-miss率が1%を越えている。

【説明】
L2-miss率が1%を越えているサブルーチンは、キャッシュが効果的に利用されていない可能性があります。キャッシュミスに着目してチューニング内容を検討します。

Statistics Performance

CPU	Commit	MIPS	MFLOPS	L2-miss	mTLB-ir	mTLB-or	Cover	
8.732334e+02	8.499728e+11	9.733626e+02	9.116430e+01	0.1302	0.0000	0.9809	51.8	aaa_
3.739155e+02	2.808601e+11	7.511325e+02	2.142955e+02	0.6746	0.0000	0.8679	61.1	bbb_
3.472051e+02	4.687254e+10	1.349996e+02	1.901587e+01	**9.4103**	0.0000	0.0009	98.5	**ccc_**
2.127186e+02	1.889833e+11	8.884192e+02	2.751608e+02	0.3547	0.0000	0.0000	99.5	ddd_
1.888756e+02	1.322634e+11	7.002672e+02	1.167205e+02	0.6511	0.0000	0.0000	99.5	eee_
1.692195e+02	2.704895e+10	1.598454e+02	2.587049e+01	**12.0801**	0.0000	0.0001	99.5	**fff_**

【改善方法】
具体的な改善方法については、CeNSSチューニングガイド のp.22「6.1.1.2次キャッシュミスの削減」を参照してください。

(3)TLB-miss率

【チェック内容】
TLB-miss率が0.02%を越えている。

【説明】
TLB-miss率が0.02%を越えているサブルーチンは、チューニングの余地があると思われます。アドレス空間へのアクセスに着目してチューニング内容を検討します。

Statistics Performance

CPU	Commit	MIPS	MFLOPS	L2-miss	mTLB-ir	mTLB-or	Cover	
8.732334e+02	8.499728e+11	9.733626e+02	9.116430e+01	0.1302	0.0000	**0.9809**	51.8	**aaa_**
3.739155e+02	2.808601e+11	7.511325e+02	2.142955e+02	0.6746	0.0000	**0.8679**	61.1	**bbb_**
3.472051e+02	4.687254e+10	1.349996e+02	1.901587e+01	9.4103	0.0000	**0.0009**	98.5	**ccc_**
2.127186e+02	1.889833e+11	8.884192e+02	2.751608e+02	0.3547	0.0000	0.0000	99.5	ddd_
1.888756e+02	1.322634e+11	7.002672e+02	1.167205e+02	0.6511	0.0000	0.0000	99.5	eee_
1.692195e+02	2.704895e+10	1.598454e+02	2.587049e+01	12.0801	0.0000	0.0001	99.5	fff_

【改善方法】
具体的な改善方法については、CeNSSチューニングガイド のp.27「6.1.2.TLBミスの削減」を参照してください。

(4)MFLOPS値

【チェック内容】
MFLOPS値が400以下である。

【説明】
MFLOPS値が400を下回っているサブルーチンは、改善の余地があると思われます。

Statistics Performance

CPU	Commit	MIPS	MFLOPS	L2-miss	mTLB-ir	mTLB-or	Cover

8.732334e+02	8.499728e+11	9.733626e+02	**9.116430e+01**	0.1302	0.0000	0.9809	51.8	**aaa_**
3.739155e+02	2.808601e+11	7.511325e+02	**2.142955e+02**	0.6746	0.0000	0.8679	61.1	**bbb_**
3.472051e+02	4.687254e+10	1.349996e+02	**1.901587e+01**	9.4103	0.0000	0.0009	98.5	**ccc_**
2.127186e+02	1.889833e+11	8.884192e+02	**2.751608e+02**	0.3547	0.0000	0.0000	99.5	**ddd_**
1.888756e+02	1.322634e+11	7.002672e+02	**1.167205e+02**	0.6511	0.0000	0.0000	99.5	**eee_**
1.692195e+02	2.704895e+10	1.598454e+02	**2.587049e+01**	12.0801	0.0000	0.0001	99.5	**fff_**

【改善方法】
具体的な改善方法については、CeNSSチューニングガイド のp.30「6.1.3.その他のスカラーチューニング」を参照してください。

(5)MIPS値

【チェック内容】
MIPS値が1000以下である。

【説明】
MIPS値が1000を下回っているサブルーチンは、改善の余地があると思われます。

Statistics Performance

CPU	Commit	MIPS	MFLOPS	L2-miss	mTLB-ir	mTLB-or	Cover	
8.732334e+02	8.499728e+11	**9.733626e+02**	9.116430e+01	0.1302	0.0000	0.9809	51.8	**aaa_**
3.739155e+02	2.808601e+11	**7.511325e+02**	2.142955e+02	0.6746	0.0000	0.8679	61.1	**bbb_**
3.472051e+02	4.687254e+10	**1.349996e+02**	1.901587e+01	9.4103	0.0000	0.0009	98.5	**ccc_**
2.127186e+02	1.889833e+11	**8.884192e+02**	2.751608e+02	0.3547	0.0000	0.0000	99.5	**ddd_**
1.888756e+02	1.322634e+11	**7.002672e+02**	1.167205e+02	0.6511	0.0000	0.0000	99.5	**eee_**
1.692195e+02	2.704895e+10	**1.598454e+02**	2.587049e+01	12.0801	0.0000	0.0001	99.5	**fff_**

【改善方法】
具体的な改善方法については、CeNSSチューニングガイド のp.30「6.1.3.その他のスカラーチューニング」を参照してください。

(6)網羅率(Cover)

【チェック内容】
網羅率が95%以下である。

【説明】
網羅率が95%を下回っているサブルーチンは改善の余地があると思われます。

Statistics Performance

CPU	Commit	MIPS	MFLOPS	L2-miss	mTLB-ir	mTLB-or	Cover	
8.732334e+02	8.499728e+11	9.733626e+02	9.116430e+01	0.1302	0.0000	0.9809	**51.8**	**aaa_**
3.739155e+02	2.808601e+11	7.511325e+02	2.142955e+02	0.6746	0.0000	0.8679	**61.1**	**bbb_**
3.472051e+02	4.687254e+10	1.349996e+02	1.901587e+01	9.4103	0.0000	0.0009	98.5	ccc_
2.127186e+02	1.889833e+11	8.884192e+02	2.751608e+02	0.3547	0.0000	0.0000	99.5	ddd_
1.888756e+02	1.322634e+11	7.002672e+02	1.167205e+02	0.6511	0.0000	0.0000	99.5	eee_
1.692195e+02	2.704895e+10	1.598454e+02	2.587049e+01	12.0801	0.0000	0.0001	99.5	fff_

【補足】
網羅率 = CPU時間/経過時間*100

【改善方法】
網羅率が低下する原因には、次の項目が考えられます。
・データTLBミス率が高い。
・プログラムの実行時に入出力処理が多い。
・プログラムの実行時に領域の確保、解放処理が多い。
・スレッド並列プログラムの場合、並列化される範囲が少ない。または、スレッド間の同期待ち時間が長い。
これらを改善改善することで網羅率が改善する可能性があります。

(7)(プロセス／スレッド)バランス

【チェック内容】
プロセス(もしくはスレッド)並列バランスが悪い。

【説明】
プロセス(もしくはスレッド)並列バランスが悪いサブルーチンは改善の余地があると思われます。

```
                          - 0 +
   +-------------------------+-------------------------+
   |                         |************************|  +583%  3.394000e+03  Process   0
   |   ***********************|                         |  - 88%  6.000000e+01  Process   1
   |    **********************|                         |  - 84%  8.100000e+01  Process   2
   |        *****************|                         |  - 67%  1.620000e+02  Process   3
   |     *********************|                         |  - 79%  1.050000e+02  Process   4
   |            ************|                         |  - 47%  2.640000e+02  Process   5
   |    **********************|                         |  - 82%  8.800000e+01  Process   6
   |                   *****|                         |  - 20%  3.990000e+02  Process   7
   |     *********************|                         |  - 80%  9.700000e+01  Process   8
   |                         |**                       |  +  7%  5.290000e+02  Process   9
   |     *********************|                         |  - 78%  1.070000e+02  Process  10
   |                         |*********                |  + 36%  6.740000e+02  Process  11
   +-------------------------+-------------------------+
    Balance against average time per Process
```

【改善方法】
具体的な改善方法については、CeNSSチューニングガイド の p.37「6.2.4.ロードバランスの改善」もしくは p.39「6.3.2.ロードバランスの改善」 を参照してください。

(8)Invarid値

【チェック内容】
Invarid値が高い。

【説明】
Invarid値が高い場合、FALSE SHARINGが発生している可能性があります。

Cache Overlap

CPU	Commit	Invalid	WriteBack	Cover	
1.609896e+02	1.127468e+11	**1.4074**	0.1196	98.3	**aaa_**
1.562683e+02	1.448637e+11	0.0000	0.2737	98.0	bbb_
1.106383e+02	1.070335e+11	0.0000	0.1814	98.3	ccc_
1.048984e+02	8.034469e+10	0.0000	0.4074	97.9	ddd_
9.838431e+01	1.591080e+10	0.0001	3.5347	98.0	eee_

【改善方法】
具体的な改善方法については、CeNSSチューニングガイド のp.18「5.3.並列性能の分析手法」を参照してください。

(9)WriteBack値

【チェック内容】
WriteBack値が高い。

【説明】
この値が高い場合、キャッシュメモリーの使用が偏っている可能性があります。

Cache Overlap

CPU	Commit	Invalid	WriteBack	Cover	
1.609896e+02	1.127468e+11	1.4074	0.1196	98.3	aaa_
1.562683e+02	1.448637e+11	0.0000	0.2737	98.0	bbb_
1.106383e+02	1.070335e+11	0.0000	0.1814	98.3	ccc_
1.048984e+02	8.034469e+10	0.0000	0.4074	97.9	ddd_
9.838431e+01	1.591080e+10	0.0001	**3.5347**	98.0	**eee_**

【改善方法】
具体的な改善方法については、CeNSSチューニングガイド のp.16「5.2.スカラー性能の分析手法」を参照してください。

(10)スレッド非並列化部分

【チェック内容】
非並列処理部分がある。

【説明】
スレッド並列プログラムにおいて、スレッド並列実行されていない場合、並列実行の阻害要因を 取り除くことで性能向上する可能性があります。

Sampling Cost

Samples	% Run	Barrier	Start	End	
7.784000e+03	6.51	7.000000e+00	468	639	**aaa_**
5.960000e+03	4.99	0.000000e+00	2	247	**MAIN__**
5.869000e+03	4.91	1.100000e+01	4	466	**bbb_**
5.689000e+03	4.76	1.000000e+00	2	169	**ccc_**
5.069000e+03	4.24	3.000000e+00	341	653	**ddd_**
3.816000e+03	3.19	0.000000e+00	722	764	eee._PRL_2_
3.720000e+03	3.11	1.700000e+01	488	796	**fff_**
3.663000e+03	3.07	0.000000e+00	230	274	ggg._PRL_10_
3.482000e+03	2.91	0.000000e+00	418	462	hhh._PRL_2_

【補足】
._PRL_X_がついていないサブルーチンは逐次実行を表します。

【改善方法】
コンパイルオプションに -Qp を指定してコンパイルします。すると、コンパイル時にプログラムリスト(.lst)が出力されるようになります。プログラムリストには、最適化情報が出力されており、並列化が阻害されている要因がわかります。他にもインライン展開の状況や、アンローリングの展開数がわかります。

プロファイラ情報の採取手順

①翻訳
コンパイル時に、-Ktl_trtのオプションを追加します。
f90ns -Umpi -Ktl_trt sample.f90 -o sample.exe

②実行
以下の環境変数を設定してジョブを実行します。

```
TRT_ENV="PMP=on"    : export TRT_ENV
PROF_STATS=7        : export PROF_STATS
PROF_PA="sta,cov"   : export PROF_PA
PROF_USERFUNC=1     : export PROF_USERFUNC
```

③解析
①②の手順でジョブを実行すると、カレントディレクトリに以下の名前のファイルが作成されます。

プロセス並列数=nnnのジョブを実行した場合
GProf_XXXXX.prof.pri
DProf_XXXXX.000.prof.pri～DProf_XXXXX.nnn.prof.pri(各プロセスごとのプロファイラ情報)
これらのファイルを、mpprofコマンドにより解析します。
以下にmpprofコマンドの実行例を示します。
mpprof GProf_12345.prof.pri > GProf_12345.txt
※注意
プロファイラ情報を取得するためにジョブを実行する場合、通常よりも実効時間が増大します。

④スレッド並列ジョブに関して
qsub時に、以下の点に注意する必要があります。
■スレッド並列数指定の環境変数を設定する。

・自動並列：PARALLEL環境変数
・OpenMP　：PARALLELおよび、OMP_NUM_THREADS環境変数
■qsubオプションlpには演算スレッド並列数＋1を指定する。
・プロファイラ採取のために、CPUを1つ使用する。

例1：自動並列の場合

```
#@¥$-q QJOB
#@¥$-r Sample00.01-01
#@¥$-lP 4
#@¥$-lp 3    ←☆PARALLEL環境変数＋1を指定
#@¥$
  :
PARALLEL=2; export PARALLEL
  :
mpiexec -n 4 Sample.out
```

例2：OpenMPの場合

```
#@¥$-q QJOB
#@¥$-r Sample00.01-01
#@¥$-lP 4
#@¥$-lp 3    ←☆PARALLEL環境変数＋1を指定
#@¥$
  :
PARALLEL=2; export PARALLEL
OMP_NUM_THREADS=2; export OMP_NUM_THREADS
  :
mpiexec -n 4 Sample.out
```

お問い合わせ

添付資料 38-1

第二版：2006 年 8 月 13 日
富士通株式会社

Parallelnavi エンハンス項目一覧

これまでの Parallelnavi の各バージョンにおけるエンハンス項目および、ご要望いただいた事項へ対応した内容を以下に示します。

項番	ﾊﾞｰｼﾞｮﾝ	エンハンス概要	備考(課題リストの項番)
1	2.0	Parallelnavi の「ラージページ機能」を利用するには、Parallelnavi のコンパイラでプログラムをコンパイルする必要があったが、Sun コンパイラで作成した既存プログラム（ロードモジュール）でも、ラージページ機能を利用可能とした。aout2lpg(1) でロードモジュールを書き換え、lpgexec(1) で実行する。	
2	2.0	外付けのジョブスケジューラを組み込むために、NQS･資源管理の内部インタフェースを公開。	JAXA 様 NSJS 向け
3	2.0.1	「UTMS-EX (課金･予算管理機能) 」が入力とするファイルのファイルサイズが 2GB を超えていた場合、正しく処理できない不具合(DISK 課金集計機能で発生) を修正。UTMS-EX が扱う全ファイルで 2GB 超えをサポート。	
4	2.1	SMP 機の特性として、同時実行している複数のジョブ間のメモリ競合などにより、同一内容のジョブでも実行時の状況により CPU 使用時間が異なる(ぶれる)場合があり、一定課金とすることができない。このため、メモリ競合の発生有無に関わらず、一定の CPU 使用量を表示する「実効 CPU 時間機能」をサポート。	固有修正
5	2.3	「UTMS-EX (課金･予算管理機能) 」において、従来ファイル名が固定であったプライスファイルをオプションで指定可能とし、課金額の切り替えなどへ対応しやすくした。	
6	2.3	qsub -oi オプション指定時にジョブ終了リストへ出力される統計情報のフォーマットを、センター毎にカスタマイズしたいとの要望へ対応するため、ジョブ終了リストの編集方法に関する内部情報を公開。	個別公開
7	2.3	ジョブサブミット時にジョブの使用資源量を省略した場合のデフォルト値および、指定可能な最大値を定義する NQS-JM の「資源設定機能（ジョブ使用資源量の標準値･最大値）」において、定義できない資源（DTU 数やバリア数など）があったが、全ての資源を定義可能とした。	項番 3, 項番 4
8	2.4	スワップアウトされたジョブは、ジョブの混雑状況によって長時間実行されない場合があるため、スワップアウト状態が一定時間継続されたジョブを、他ジョブをスワップアウトすることにより強制的にスワップインし実行する「強制スワップイン機能」をサポート。	
9	2.4	従来、1 ノードあたり最大 64 プロセス並列までしか利用できなかったノード内バリアを、最大 128 プロセス並列まで利用可能とした。	

項番	ﾊﾞｰｼﾞｮﾝ	エンハンス概要	備考(課題リストの項番)
10	2.4	1 ノードあたり最大 64 プロセス並列までしか利用できなかった DTU コンテキストを、128 プロセス並列まで利用可能とする「Shared-UDTU 機能」をサポート。	
11	2.4	従来、CPU で 1 ビットエラー (CE) が多発しても当該 CPU を使用し続けており、2 ビットエラー (UE) へ発展 (システムダウン) させてしまう可能性があった。このため、1 ビットエラー多発が発生した場合、当該 CPU 使用ジョブを強制終了&再キューイングした上で、CPU をオフラインにしジョブ運用から除外するよう改善した。	
12	2.4.1	従来、システム (NQS) 起動時にネームサービスの応答が無い場合、キュー上の全ジョブを削除していた (削除されたジョブはエンドユーザに再投入してもらう必要がある) 。このため、ネームサービスの運用変更作業ミスなどによっても、キューイングされていた全ジョブが削除されてしまうことがあった。NQS 起動時にネームサービスの応答が無い場合は、ジョブを削除せずキューを停止するように改善した。	
13	2.4.1	従来、ジョブの現在の経過時間を簡単に表示することができなかったため、リアルタイムに経過時間を表示する機能を qstat コマンドへ -i オプションとして追加した。本オプションは、経過時間制限までの残り経過時間も表示するので、ジョブ内から利用し経過時間制限打ち切りに向けた後処理 (中間結果出力など)が可能となった。	
14	2.4.1	従来、通常ページとラージページは同一の使用制限値しか定義できなかったため、大規模なラージページに合わせて制限値を定義にせざるを得なかった。このため、誤って通常ページを多量に使用するジョブを実行すると通常ページが枯渇し、システムのスローダウンを発生させることがあった。通常ページとラージページの使用制限値を別々に定義可能とし、通常ページ枯渇を回避可能とした。	項番 10
15	2.4.1	従来、NQS への不当なポートアクセスがあった場合、NQS が異常終了しジョブ運用を停止させてしまっていた。不当なポートアクセスがあっても、NQS を異常終了させないように改善した。	項番 34
16	2.4.1	従来、ジョブの CPU 使用時間制限は、プロセスへ割り当てた CPU の総和に対して行われており、スレッド並列化の促進を妨げる要因の一つとなっていた。このため、CPU 毎の最大 CPU 使用時間で制限する機能を追加した。	項番 1
17	2.4.1	従来の「強制スワップイン機能」 (項番 8 参照) では、不必要に実行中ジョブをスワップアウトすることがあったため、強制スワップイン対象ジョブの実行に必要な資源量が確保できるだけの最小限のジョブのみをスワップアウトするように改善した。	
18	2.4.1	ジョブ実行状況確認コマンド (rscsetstat) のレスポンスを改善した (128 プロセス並列ジョブ表示：20 秒→1 秒)。	項番 32

以降は、今後エンハンス予定のもの。2.4.2 は、2006 年 12 月末出荷予定。

項番	ﾊﾞｰｼﾞｮﾝ	エンハンス概要	備考(課題リストの項番)
19	2.4.2	システムダウンなどにより破壊された課金ファイルを修復する機能を追加。	2.4.1 向け固有修正提供済
20	2.4.2	実行中ジョブの資源制限値を動的に変更する qmodify コマンドをサポート。あと少しで実行終了するが、終了前に制限値に到達しそうなジョブを救済。	項番 14
21	2.4.2	従来、負荷分散を考慮し空き資源の多いノードを優先してジョブを実行させていたため、繁忙でない期間に空きノードを停止したい場合などでも、空きノードが発生しにくかった。空き資源の少ないノードを優先する「ノード集中割り当て機能」をサポートし、(空きノードを増やす運用を可能とする。	項番 9, 項番 35
22	2.4.2	従来、NQS データベース破壊により実行不可なジョブが発生した場合、NQS の内部ログのみに実行不可ジョブの発生記録が残されていたが、ジョブ実行不可となった旨をシステム管理者および(可能であれば) 当該ジョブ投入ユーザへ通知するように改善する。	
23	2.4.2	従来、ジョブ実行時にユーザ認証が失敗した場合、当該ジョブを削除していた。しかし、ユーザ認証の失敗は、一時的なネームサービスの停止などの場合が多く、ネームサービス復旧後に簡易に当該ジョブの再実行を可能とすべきである。このため、ジョブ削除は行わず、再キューイングするように改善する。	
24	2.4.2	従来、ジョブがラージページメモリの獲得に失敗しても、メモリ獲得失敗の事象のみロギングされ、どのジョブでの発生かを特定することができなかった。このため、不適切にラージページを使用しているエンドユーザを、システム管理者が指導することが困難であった。ラージページメモリ獲得失敗がどのジョブで発生したのかを特定できるよう、メッセージを改善する。	
25	2.4.2	従来、メモリの 1 ビットエラー (CE) 多発および、2 ビットエラー(アプリ走行時) が発生しても当該メモリを使用し続けており、後続ジョブが繰り返しメモリエラーで異常終了する可能性があった。このため、エラー発生メモリをラージページから切り離して当該メモリをオフラインし、ジョブ運用から除外するように改善する。	
26	将来検討	ジョブ実行開始してから一定時間経過後に、未使用なラージページを開放し、メモリの稼働率を向上させる。	項番 12

添付資料 39-1
JAXA 大規模 SMP クラスタにおける運用の課題と工夫

松尾裕一，土屋雅子
(宇宙航空研究開発機構)

1．はじめに

PRIMEPOWER HPC2500 による JAXA 大規模 SMP クラスタシステムの運用の現状と課題, 高度有効利用のために行った工夫と対策について述べる.

2．現行システムの概要

JAXA の大規模 SMP クラスタは，Central Numerical Simulation System（CeNSS）と呼ばれ, 富士通 PRIMEPOWER HPC2500 の 18 筐体から成る. 以前の分散ベクトルシステム「数値風洞（NWT)」（166PE, 280GFLOPS）から，2002 年 10 月に更新されたものである.（設備更新のため, 移行作業は 2002 年 4 月より開始され, 数値風洞は 2002 年 7 月に撤去された. 2002 年 10 月より 18 筐体で稼動し，約 3 ヶ月の試行期間を経て 2003 年明けより本格稼動した.）

各筐体は, すべて 128 個の CPU を有するが, このうち 14 筐体は計算部分を受け持っている. 計算部の各筐体は 4 ノードに分割され，分割された各ノード（計算ノード）は，64GB の共有メモリ空間を有する 32 ウェイの SMP（Symmetric Multi Processors）を構成し，DTU を通じてクロスバ結合ネットワークに接続されている. 計算ノードは全部で 56 ノードあり, 1,792CPU が割り当てられている．この部分の総演算性能は 9.3TFLOPS，メモリはトータルで 3.6TB に達する．残りの 4 筐体のうち，3 筐体は，それぞれ筐体あたり 128GB の共有メモリを有する 64 ウェイ SMP ノード 2 台からなり，サービスノード，ログインノードとして，コンパイルやデバッグ, 小規模処理, インタラクティブ処理, ログイン処理などに当たる. 最後の 1 筐体（IO ノード）は，128 ウェイの SMP として，IO 処理やファイルシステムの管理，その他の雑処理を担当している．IO ノードには，ストライピングされたファイバチャネルにより，RAID ディスク及びテープライブラリが接続され, HSM ソフトウェアにより, 階層ストレージとして利用されている．図 2.1 に CeNSS の論理構成，表 2.2 に主なスペックを示した.

このような構成にした理由として運用上重要な要素は,

(1) 以前の数値風洞システムのときから大多数のユーザは XPFortran または MPI によるプロセス並列化を済ませており，1 筐体を 128 ウェイ大規模 SMP による巨大な共有メモリマシンとして使うより, 4 分割して中規模の SMP ノードとしてノード間転送性能を重視する必要があった.

(2) 数値風洞において，フロントエンドの性能・機能が不十分であることによる使い勝手の悪さ（例えば，デバッグ等の小規模処理の機動性の悪さ）がユーザには非常に不評だったことを踏まえ，スワップ等の機能追加などに加えて計算ノードと同種のフロントエンドとして処理環境を同一化するなどにより使い勝手の向上に配慮する必要があった.

(3) 非定常シミュレーションなどの入出力を時系列で頻繁に行うジョブや小規模ファイルを多数吐き出すジョブなどが増えており，入出力系やファイル系を強化する必要があった.

などが挙げられる.（もちろん,ソフトウェアの標準化や ISV アプリの問題等の他の事情もあ

る.）このあたりのもう少し細かな事情・経緯やシステムの詳細は，文献(1)(2)を参照されたい.

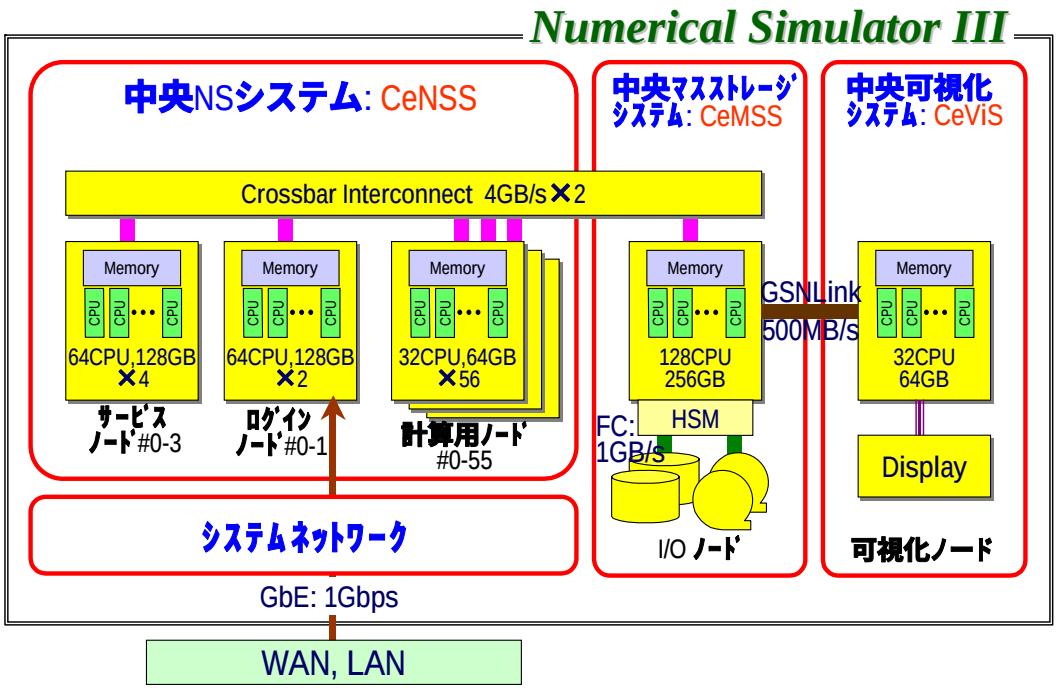


図 2.1　CeNSS の論理構成（2002 年 10 月）

表 2.2　CeNSS の主なスペック

理論ピーク性能	9.3TFLOPS		
総ユーザメモリ	3.6TB		
CPU	SPARC64 V, 1.3GHz, 2MB L2$		
計算ノード（ノード数，構成，CPU 数）	56	32waySMP	1,792
サービスノード	4	64waySMP	256
ログインノード	2	64waySMP	128
I/O ノード	1	128waySMP	128
システム総 CPU 数	2,304		
結合ネットワーク	クロスバ（4GB/s×2）		
ディスク容量	57TB		
テープライブラリ容量	620TB		

3．CeNSS における運用の推移と課題

今回のシステムは，ベクトルからスカラーへの方式変更ということもさることながら，CPU 数が 14 倍弱，メモリ量が 80 倍強，ストレージ量がディスクだけで 200 倍弱と，規模・物量の面で前システムに比べ相当に拡大したため，当初は，未経験によるパラメータ設定の困難や実際面の課題の噴出が懸念された. この 2006 年 10 月で導入して丸 4 年を迎えようとしているが，心配された更新後の大きな混乱もなく，その後も含めて試行錯誤ではあるが今までの勘を頼りに何とかやって来られたというのが率直な感想である．ここでは，統計情報を用いてこの 4 年間の運用の経緯を振り返るとともに，いくつかの論点から総括し課題などに言及してみたい.

§3.1　運用の統計情報

このシステムの運用で特徴的なのは，ISO9000 による運用管理を適用したことである.「ISO をスパコンの運用に」などという向きもあろうが，その是非は別な機会に論ずることにして，極めて有意義であったと思うのは，ISO の仕組みによって当初からの各種統計情報をきちっと取ることができたということである.（今でこそ ISO は定着しているものの，初期のころはベンダーをはじめとして JAXA 側の運用部隊にも建設的にご協力いただいた点には深謝したい.）

(1) ハード/ソフト障害の推移

図 3.1 は，運用当初からのハード障害及びソフト障害の件数を月別（棒グラフ），累計（折れ線グラフ）で示したものである．最初は多く徐々に少なくなるという典型的な漸近カーブを描いているが，途中 1 年後ぐらいの時点でソフト障害が一時的に増加している時期があるのが見て取れる．これは，ユーザがシステムに慣れて来て，システムの負荷が非常に高まり，それまでの運用では現れて来なかった障害が表に出てきてことによるものであると解釈している.「更新して 1 年後ぐらいの負荷が高まった時期は要注意」という教訓？は，今後の運用に役に立てて行きたいと思っている.

月別の発生をみると，初期動揺期→遷移期→安定期ぐらいに時期を分けることができるであろう．運用的には遷移の期間を短くすることがポイントと思われる．一方，累計をみるとハード障害の傾きが 0 近くに漸近して来ているのは直感的にも理解できる傾向である．これは予兆監視により運用への影響を最小限に留めている効果も大きい. しかし, ソフト障害については, ハード障害ほど傾きが小さくなっていない．これは，ソフト障害は基本的になかなか取れにくいことを意味しているのか，ソフト障害は基本的にある一定頻度で起こりやすいものなのか，ベンダーも交えた突っ込んだ解釈・分析が必要であろう.（ちなみに，ここで言っている「障害」とは，あくまで運用管理サイドで把握し定義しているものであって，ベンダーからの情報ではないことに注意.）

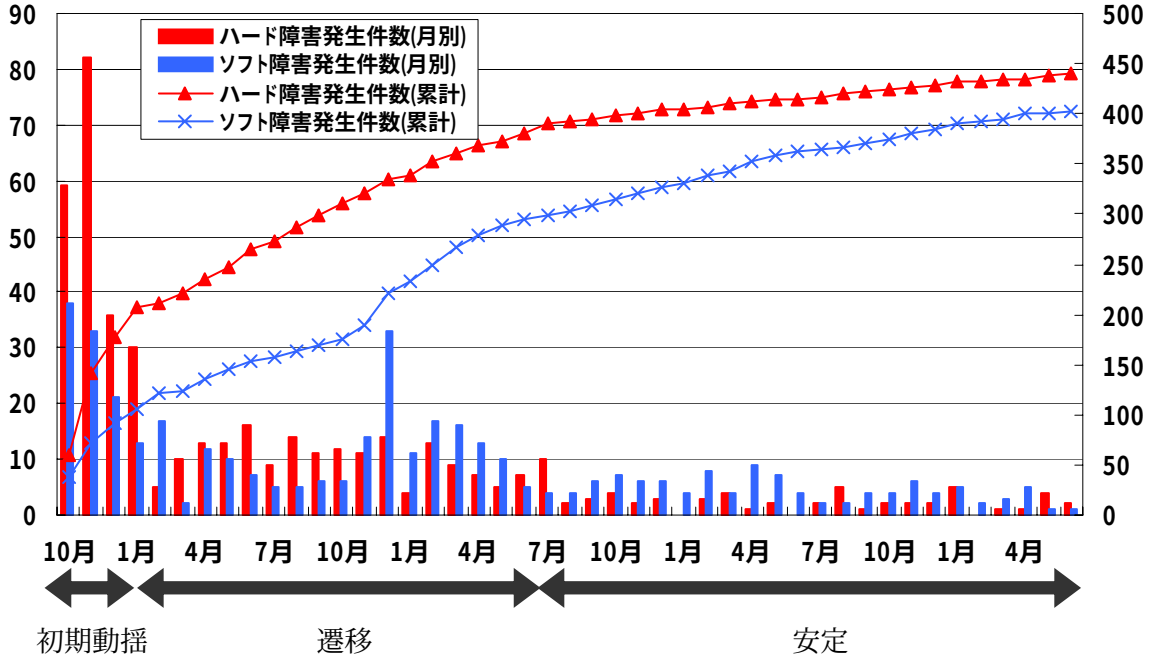


図 3.1　ハード/ソフト障害の推移

(2) 運用基本データの推移

運用基本データの推移を表 3.2 にまとめる．ここ 1 年半ぐらいは月平均値で，ジョブ処理件数 8,000 件, ジョブ運用時間 1,000,000 時間, CPU 稼働率 90%以上という数字が得られている．

表 3.2　CeNSS における運用の総括

月次	バッチジョブ処理件数	CPU 割当時間	CPU 稼働率	CPU 使用時間	実効 CPU 使用時間	ジョブ運用時間	電源投入時間	運用日数
2002 年 10 月	27,724	85,308	57.3%	64,009		148,800	824,656	5
11 月	30,115	293,042	57.9%	255,364		505,920	858,985	17
12 月	15,638	474,683	72.2%	394,853		657,696	1,033,525	17
2003 年 1 月	10,005	534,778	65.8%	474,324		812,448	929,143	21
2 月	10,453	462,522	54.3%	393,232		851,136	959,363	22
3 月	4,864	407,142	75.2%	335,911		541,632	1,030,450	14
4 月	11,295	776,419	77.2%	749,382		1,005,888	1,243,017	26
5 月	9,524	518,844	74.5%	493,894		696,384	1,086,001	18
6 月	12,268	819,403	78.4%	784,390		1,044,576	1,249,588	27
7 月	13,021	932,358	86.1%	865,076		1,083,264	1,268,287	28
8 月	9,593	836,167	86.5%	647,703		967,200	1,203,572	25
9 月	13,003	882,208	87.7%	641,238	378,308	1,005,888	1,219,041	26
10 月	10,000	871,920	86.7%	609,308	470,422	1,005,888	1,242,451	26
11 月	11,698	831,705	79.6%	584,132	442,933	1,044,576	1,206,040	27
12 月	13,107	769,290	82.9%	588,756	457,988	928,512	1,257,637	24
2004 年 1 月	10,464	865,273	89.5%	672,442	536,195	967,200	1,171,052	27
2 月	9,107	768,855	76.4%	602,484	477,613	1,005,888	1,163,899	29
3 月	8,252	864,746	77.5%	601,648	468,263	1,115,208	1,241,342	31
4 月	6,683	795,174	72.6%	587,958	433,019	1,095,065	1,019,591	30
5 月	6,392	799,577	78.2%	573,918	403,577	1,021,963	1,235,611	31
6 月	7,071	863,698	77.9%	632,304	462,128	1,108,297	1,199,412	30
7 月	9,631	1,023,768	89.5%	746,536	540,789	1,143,803	1,253,418	31
8 月	7,213	899,402	89.1%	612,541	485,711	1,009,067	1,171,749	31
9 月	7,597	1,014,837	88.1%	730,297	578,453	1,151,445	1,216,779	30
10 月	7,561	965,746	86.1%	682,654	501,909	1,121,485	1,266,194	31
11 月	7,737	981,669	86.7%	751,433	550,508	1,132,109	1,200,883	30
12 月	6,506	837,655	84.7%	628,233	446,005	979,738	1,100,073	27
2005 年 1 月	9,274	903,114	83.7%	683,409	518,150	1,077,645	1,089,229	27
2 月	11,066	963,801	88.2%	717,632	455,024	1,082,974	1,128,720	28
3 月	9,747	1,087,054	88.2%	829,770	487,909	1,232,740	1,265,859	31
4 月	6,916	1,058,924	89.6%	763,092	550,908	1,182,446	1,190,237	30
5 月	8,446	986,760	90.2%	744,533	529,940	1,093,670	1,204,578	31
6 月	8,537	1,128,756	95.6%	784,831	542,796	1,180,802	1,221,543	30
7 月	7,870	1,199,037	95.5%	892,740	618,449	1,255,824	1,264,016	31
8 月	7,588	943,621	92.0%	690,911	481,964	1,025,245	1,089,357	31
9 月	8,550	1,148,136	94.6%	834,972	596,940	1,213,140	1,227,422	30
10 月	7,652	1,150,604	93.6%	781,372	536,530	1,229,741	1,250,585	31
11 月	7,040	1,138,235	95.4%	795,796	520,985	1,193,472	1,202,992	30
12 月	7,297	938,062	91.8%	713,035	479,877	1,022,029	1,133,128	29
2006 年 1 月	7,443	1,003,819	92.3%	725,236	492,202	1,087,790	1,092,508	27
2 月	9,865	1,062,306	94.6%	779,963	563,887	1,123,280	1,134,856	28
3 月	7,404	1,160,357	95.3%	685,173	494,169	1,218,095	1,232,882	31
4 月	8,795	1,066,759	96.5%	751,545	522,886	1,105,648	1,150,463	30
5 月	10,790	1,144,923	96.0%	784,572	556,694	1,193,097	1,247,014	31
6 月	6,561	1,138,996	92.7%	825,026	538,852	1,228,382	1,227,368	30
7 月	7,260	1,141,966	90.6%	829,513	527,496	1,259,771	1,243,572	31
8 月	8,520	1,084,960	96.4%	797,450	545,801	1,125,297	1,173,176	31
平均/月	9,854	885,668	86.2%	662,013	505,424	1,027,280	1,162,154	27.2

図 3.3 に各種 CPU 時間の推移をグラフで示す．ここで，

CPU 割当時間：　　CPU を実行ジョブに割り当てていた時間．ジョブが使用していなくても割り当てられていれば加算される．

CPU 使用時間：　　utime　+　stime

実効 CPU 使用時間：　　CPU 時間からメモリ待ちや DTU 転送待ち時間を除いたもの．

をあらわしている．図において，CPU 割り当て時間が少しずつ増大しているのは，まさに運用側のいろいろな努力による賜物である．2003 年 7 月以前に線が重なっているのは，きちっとしたデータが取れていなかったことによる．平均値でみると，CPU 割当時間：CPU 使用時間：実行 CPU 使用時間 ＝ 1：0.75：0.57 となっていて，CPU 割当時間に対する有効な CPU 稼働時間は約 6 割に留まっており，プログラムの作りの問題などもあるにせよ，ベンダーにはこの数字を真摯に受け止めていただきたいと思う．図 3.4 に CPU 稼働率（＝CPU 割当時間／ジョブ運用時間）の推移を示す．2005 年 5 月以降は常に 90%を越えていて，利用率としては限界に達している．

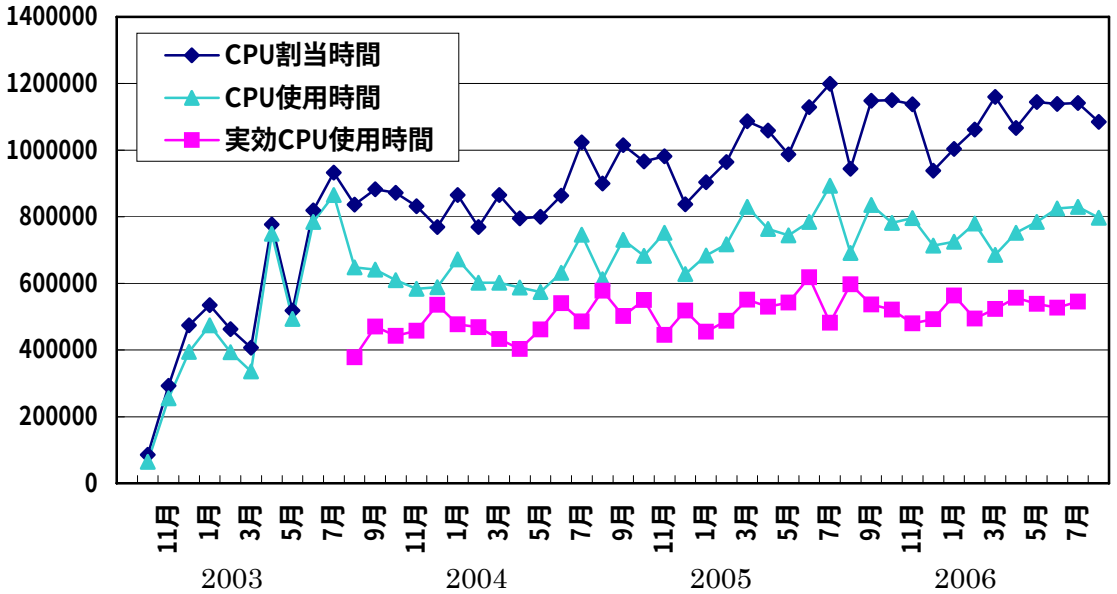


図 3.3　CPU 時間の推移

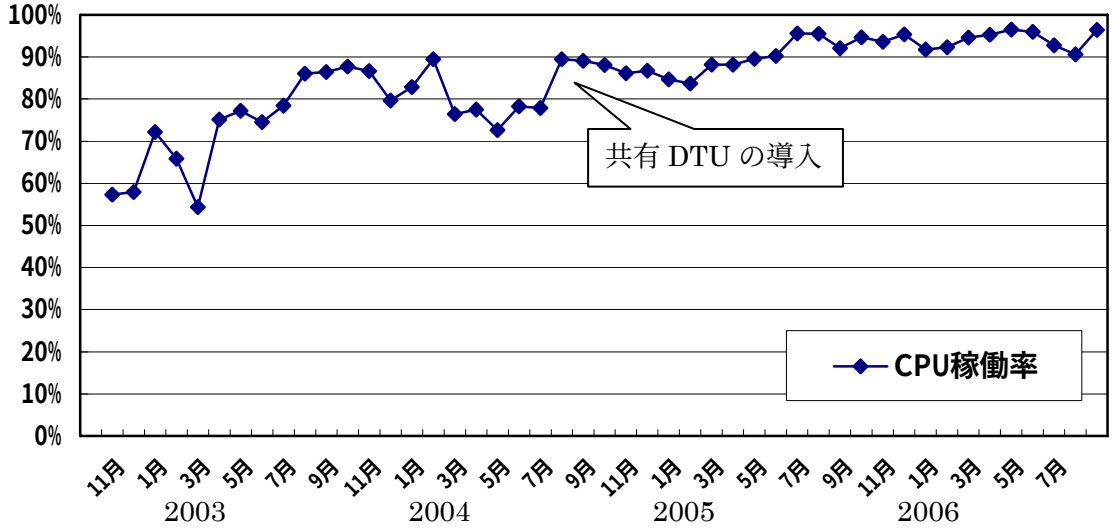


図 3.4　CPU 稼働率の推移

(3) プロセス数，スレッド数の推移

図 3.5(a)及び(b)は，運用当初からのプロセス数の推移を，全ジョブ数に対する割合，プロセス数×経過時間でみたときの割合をそれぞれ示したものである．システム管理者，テストジョブユーザを除き，60 秒以上実行されたジョブを対象とした．ジョブ数，システム占有割合ともに，33 プロセス以上の高並列ジョブが増加傾向にあるのがわかる．

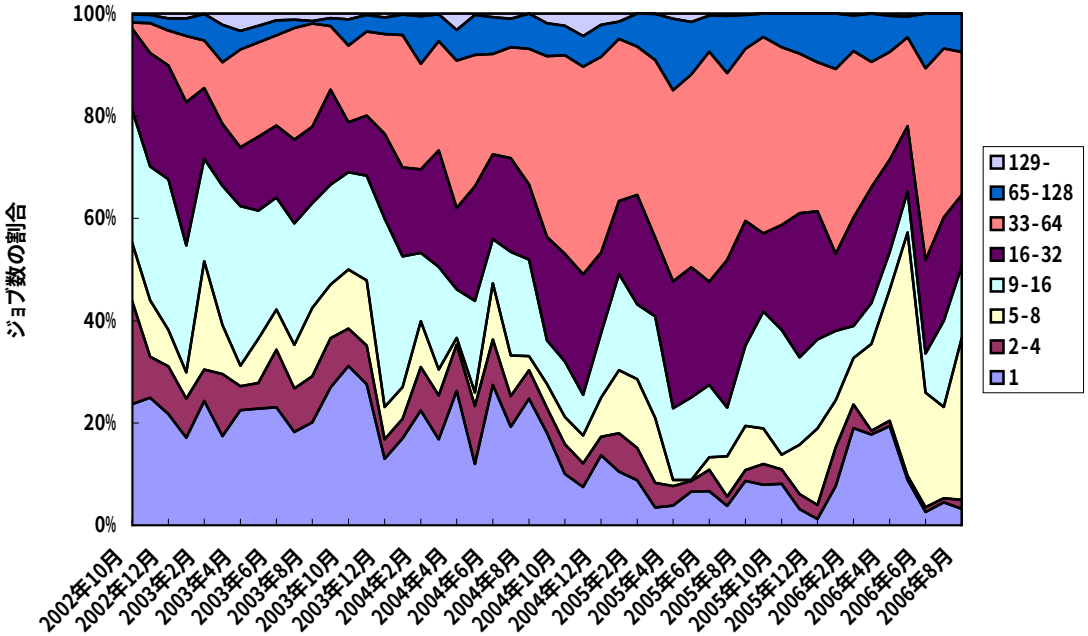


図 3.5(a) プロセス数の推移（ジョブ数）

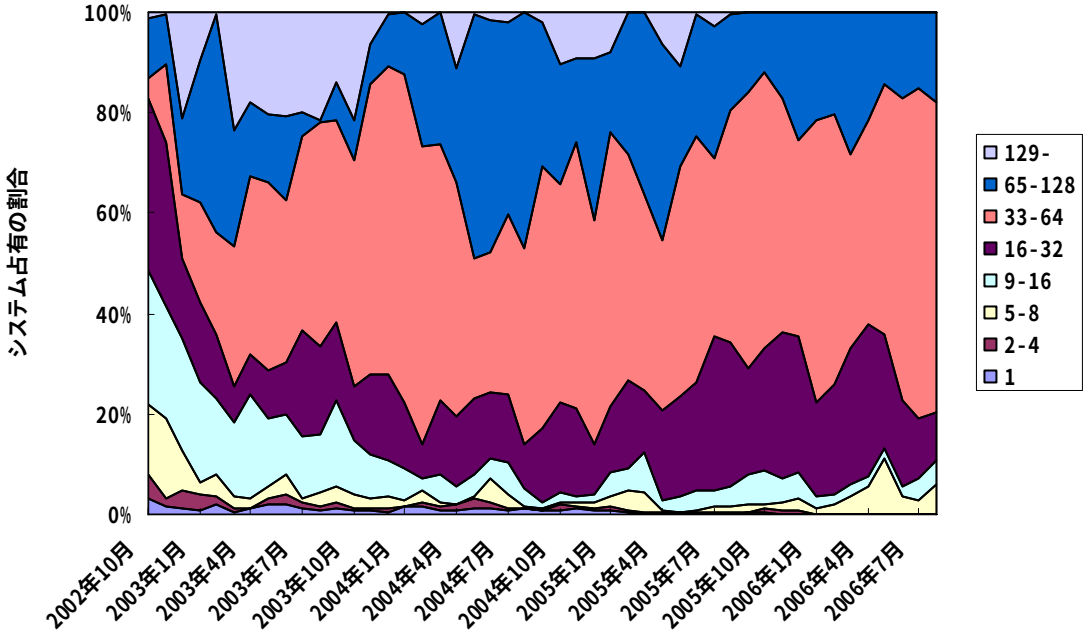


図 3.5(b)　プロセス数の推移（プロセス数×スレッド数×経過時間）

一方，図 3.6(a)及び(b)は，運用当初からの利用されたスレッド数の推移を，全ジョブ数に対する割合，プロセス数×経過時間でみたときの割合をそれぞれ示した．システム管理者，テストジョブユーザを除き，60 秒以上実行されたジョブを対象とした．最初の頃，ジョブ数としては多いが時間として少ないのは，一生懸命トライはしたということであろうか．その後は，非スレッドのジョブが増加する傾向にあり，スレッド利用者は減っている傾向が現れている．また，5 スレッド以上の多スレッド並列がほとんどないのもある意味特徴的である．

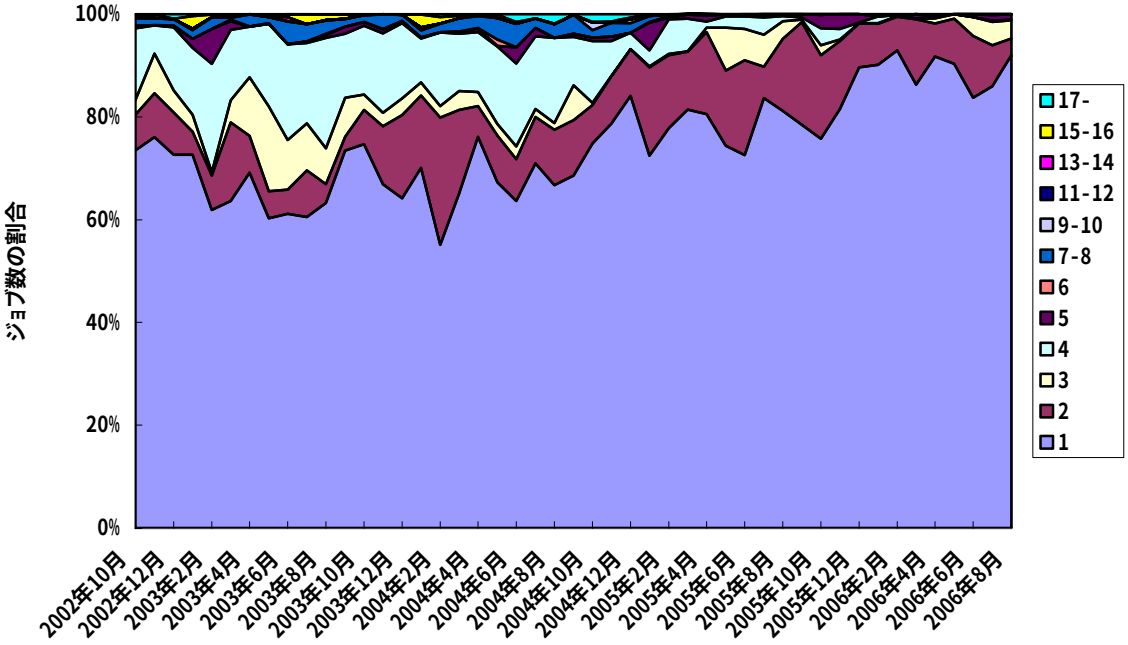


図 3.6(a)　スレッド数の推移（ジョブ数）

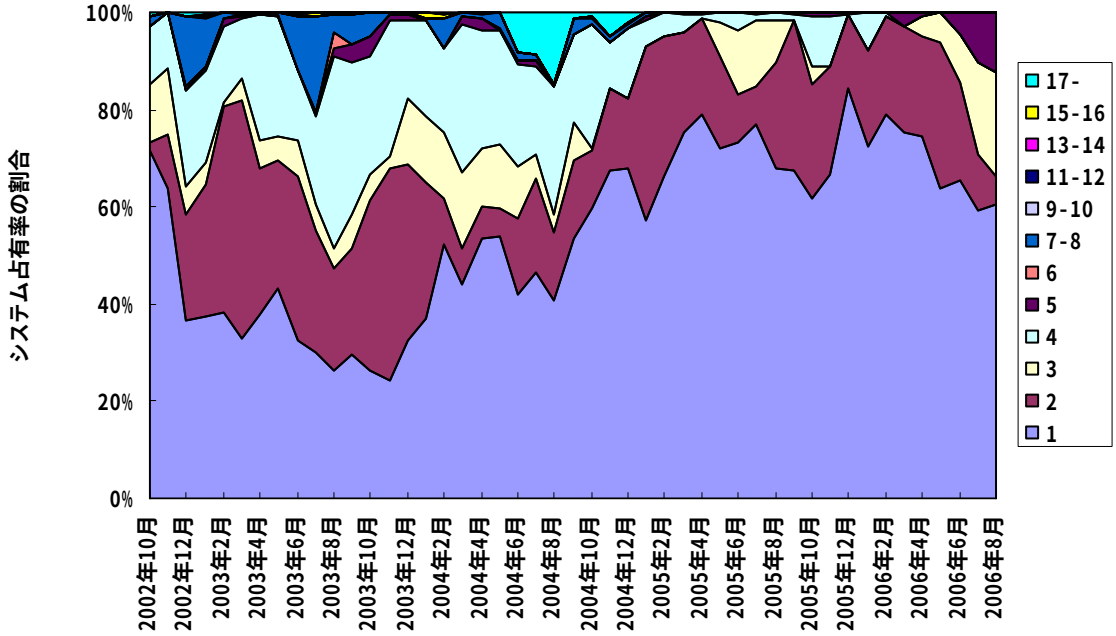


図 3.6(b)　スレッド数の推移（プロセス数×スレッド数×経過時間）

こうしたプロセス数，スレッド数の推移が示しているある一定の傾向は，あくまで，JAXA のジョブ運用の方針なりパラメータ設定なりに依存していることにまず注意しなければならない．たとえば，129 プロセス以上のジョブが極端に少ないのは，このようなジョブがいろいろな理由から現行の運用では流れにくくなっているからであって，高並列のジョブの実行に問題があるとか流す要求がないというわけではない．そうはいっても，（初期の 3 ヶ月ほどの試行期間を除けば）運用方針やジョブの設定パラメータの大きな変更はしていないので，こうした並列度が全体として拡大している傾向は，やはりパラメータ等の設定によって誘導されたものではなく，ユーザ利用の自然な動向として現れてきたものである解釈できるであろう．

そのあたりを念頭に置いて図 3.5，3.6 を見直してみると，プロセス数が増えてスレッド数が減っているわけであるから，基本的にスレッド並列やハイブリッド並列が減って MPI や XPFortran のみのプロセス並列が増加していることになる．その理由としては，1)マルチブロックジョブのようにプロセス数を変えられない場合，スレッドを使うと使用する利用 CPU 数が増え，ジョブ実行の優先順位が下がってしまうので，待ち時間を入れるとスレッド利用のメリットがなくなってしまう，2)スレッド並列，特に自動並列の性能がそもそも上がらないのでスレッドを使う気がしない，などが考えられ，ユーザにとってのスレッドの位置づけや使うメリットが明確でなくなっている傾向が表れている．

(4) タスクタイプの推移

図 3.7 に，過去約 1 年間の XPFortran と MPI の利用割合（ジョブ数）の推移をプロットした. 未だに XPFortran の利用が 4 割ほどあるが, 多少のでこぼこはあるもののなんとなく MPI の利用が少しずつ増えて来ているように見える.

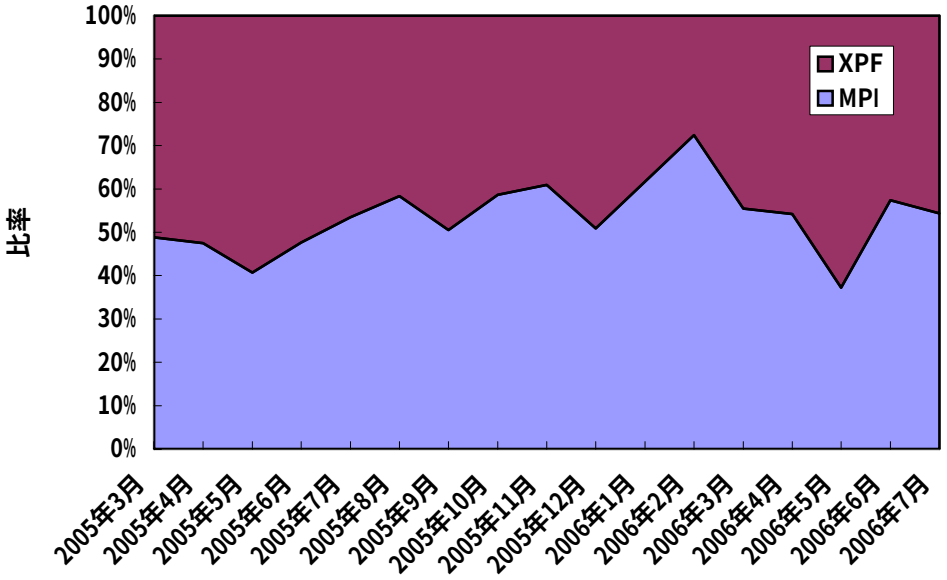


図 3.7　XPFortran 対 MPI の利用割合の推移（ジョブ数）

(5) メモリ使用量の推移

図 3.8(a)及び(b)は，ラージページ使用量のジョブ数における割合，実際の使用量積算の割合の推移を示したものである．明らかに，少しずつではあるが大規模ラージページの使用が増える傾向にあるのがわかる．

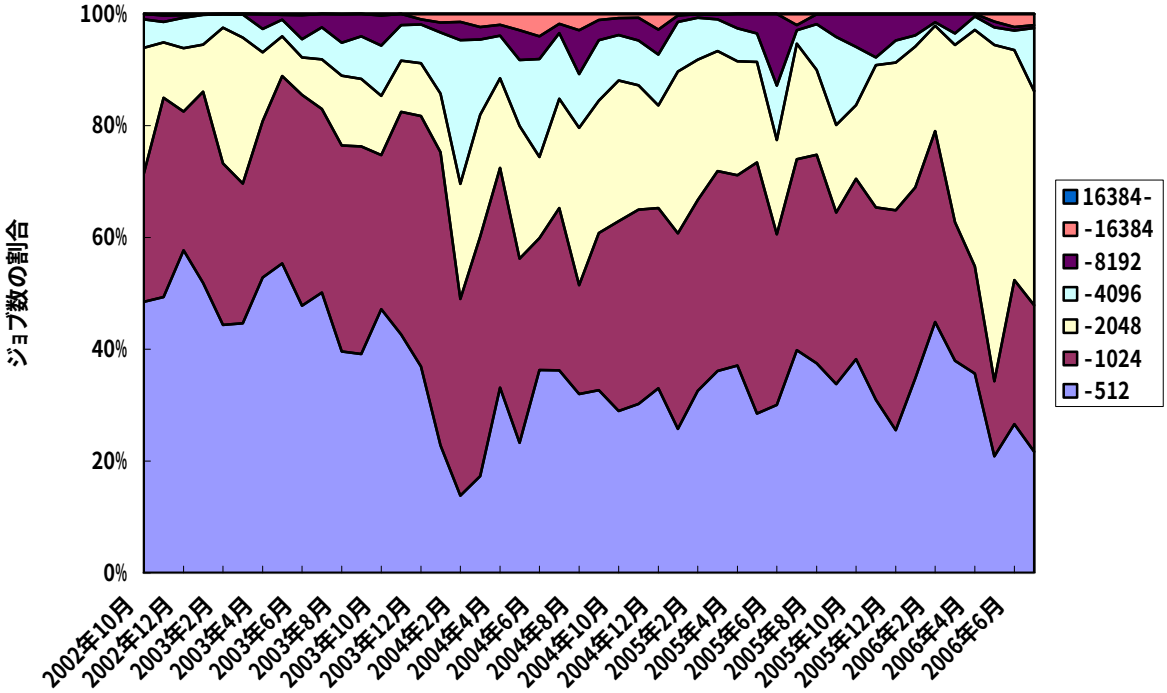


図 3.8(a)　ラージページ使用量（ジョブ数）

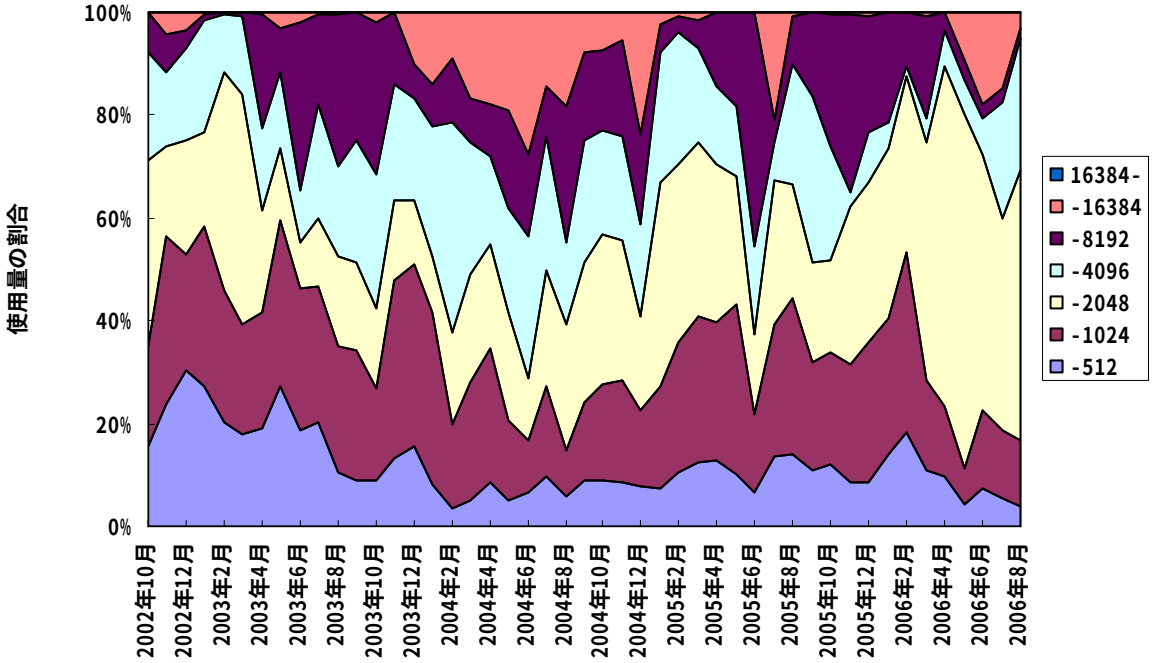


図 3.8(b)　ラージページ使用量（積算における割合）

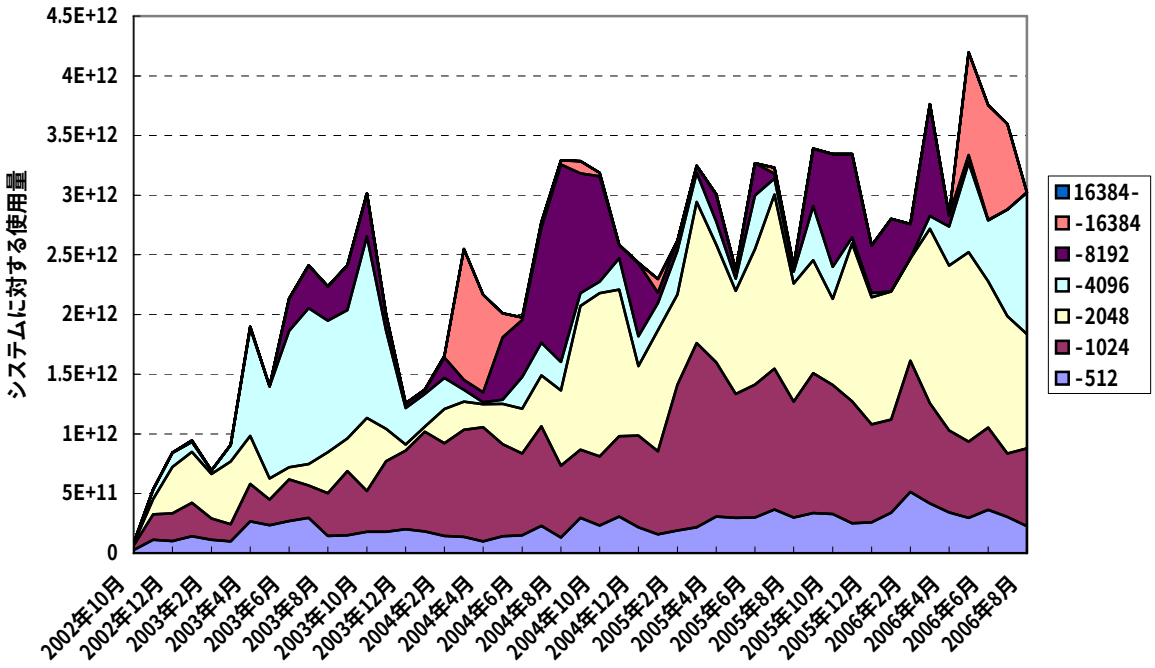


図 3.9　ラージページ使用量（積算）

また，図 3.9 はラージページ使用量の積算値の推移を示す．積算自体も少しずつ増加しつつあるものの，運用からみると，図 3.4 に示したように CPU 資源が逼迫しており，メモリ的にはまだ余裕がある状況にある．従って，メモリの有効利用に関しては今のところ運用の切迫した課題にはなっていないが，ユーザの要求メモリ量と実際にジョブで使用されるメモリ量の乖離が激しいなどの問題もあるので，今後は，動的なメモリ解放機構の実現等に取り組んで行く必要があるかもしれない．

(6)ストレージの利用状況

図 3.10 は，ディスク+テープのストレージの使用量の推移を示したものである．運用当初からのデータではなく，過去 3 年間分の推移であることに注意されたい．HSM による階層ストレージの運用をしているので，ディスクはキャッシュとして利用され，データが一定量に達するとテープにアーカイブされる．テープのデータ保管は，保全性を考慮し，ダブルコピーの運用としている．本システムの運用においては，ディスク＋テープのクオータは設定していないので，テープを含めればユーザは必要なだけストレージを利用できることになる．本グラフからいえるのは，データ量は比較的リニアに伸びてきているということである．これは将来のストレージの使用量が比較的予測しやすいことを意味している．ファイル数としては数 100 万ファイルのオーダである．

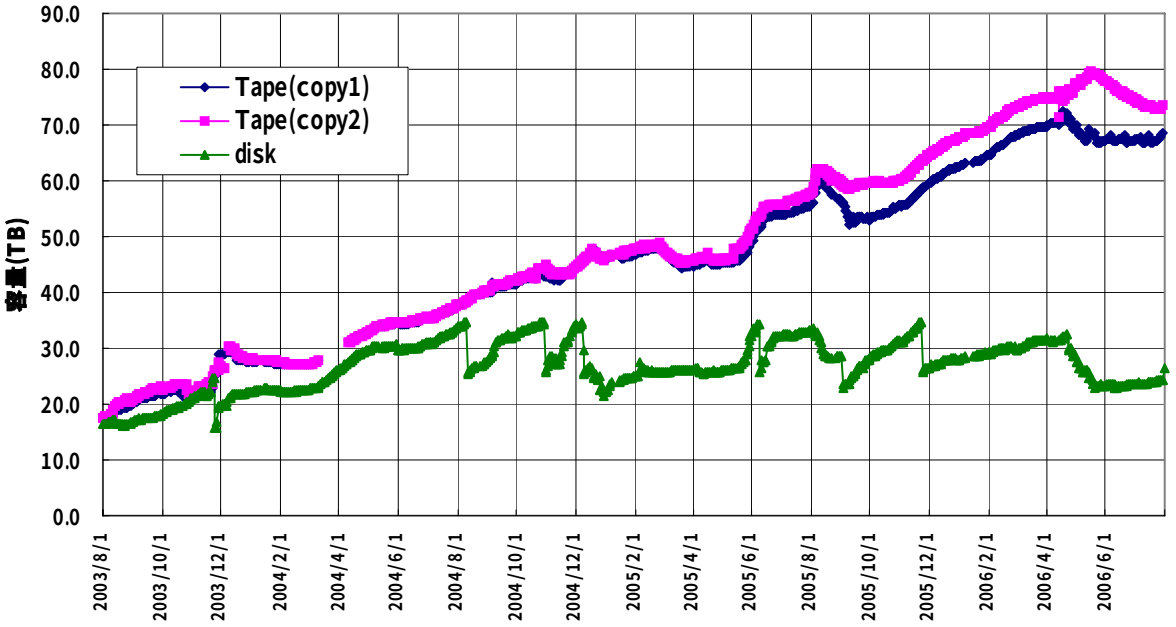


図 3.10　ストレージの使用量の推移

§3.2　運用の整理と課題

システム運用のここ 4 年間の推移は，主なところは以上の通りである．現在は，計算機としては安定期・円熟期に入って行く中で，「来たジョブを確実に処理し安定性を保持しつつ稼働率を最大にする」という運用本来の目的に留まらず，「独法化や宇宙 3 機関統合という大きな流れの下，計算需要を効率よく把握し所望の成果をタイムリーかつ最大限に発信する，多様化しつつある処理要求に的確に応える，という要求に対してより積極的な対応が求められている」というところであろうか．運用の課題と高度有効利用の工夫については，ほぼ 2 年目の時点で，SS 研科学技術計算分科会 2004 年度第 2 回会合において一度報告したところであるが，ここではその後の展開及び現状と課題についていくつかのポイントを絞って整理してみたい．

(1) ベクトル機からの移行

全部で 2,304 個のスカラーCPU から成る大規模並列システムを，ベクトル機の経験しかない我々が導入し運用するのは，移行も含め，一つの大きな挑戦であったことは確かである．しかし，効率的な日程，スタッフの奮起，ユーザの協力，ベンダーの支援という関係者の相補的な尽力により何とか乗り切ることができた．特に，JAXA では，後述のようにジョブスケジューラを自前で開発しているので，システムの状態や多様な処理要求に臨機応変に対応することができたことが，実際，大きかった．また，プログラム移行の観点からは，主アプリケーションである CFD では，プログラムは単純な多重ループ構造の組み合わせとして構成されていることが多いことから，当初，並列に関しては，「ベクトルループを自動並列ループにマッピングさせる」（図 3.11），データ構造に関しては，「C のプリプロを用いる」というわかりやすい移行モデルを敷いたのが，円滑な移行を実現できた要因の一つと考えても良いかもしれない．無論，大口ユーザに対する個別移行サポートやセミナーの開催，利用手引き整備などによる教育・啓蒙

```
          :
  !XOCL PARALLEL REGION
          :
  !XOCL SPREAD DO /IPN
       do 1000 n = 1, nblock
       do 1 l = 1, lmax
       do 1 k = 1, kmax
v      do 1 j = 1, jmax
v        di      = 1./q(j,k,l,1,n)
v        u(j,k,l) = q(j,k,l,2,n)*di
v         :
v        rmu(j,k,l,n) = (cc**1.5)*c2bp/(cc+c2b)
v        turmu(j,k,l,n) = 0.
v    1 continue
   1000 continue
  !XOCL END SPREAD DO
          :
  !XOCL END PARALLEL REGION
          :
```

```
          :
  !XOCL PARALLEL REGION
          :
  !XOCL SPREAD DO /IPN
       do 1000 n = 1, nblock
p      do 1 l = 1, lmax
p      do 1 k = 1, kmax
p      do 1 j = 1, jmax
p        di      = 1./q(j,k,l,1,n)
p        u(j,k,l) = q(j,k,l,2,n)*di
p         :
p        rmu(j,k,l,n) = (cc**1.5)*c2bp/(cc+c2b)
p        turmu(j,k,l,n) = 0.
p    1 continue
   1000 continue
  !XOCL END SPREAD DO
          :
  !XOCL END PARALLEL REGION
          :
```

図 3.11　並列プログラムの移行の例

の効果も大きかったと思われる．ただ，自動並列・スレッド並列は内側ループしか並列化されないなど困難も多く，図 3.6 で見たように少しずつ利用者が減っているのを見ると，もう少し周到な初等教育が必要だったかもしれない．

(2) 大規模 SMP クラスタ上でのジョブ運用

SMP ということでまず問題になったのは，ジョブの割り付け方に関する問題である．標準では，プロセス・スレッドを 1 ノードにできるだけ集める PACK という方式と，できるだけ分散させる UNPACK という方式があるが，自前のジョブスケジューラでやりくり可能であるし，分散させても空いている CPU にどんどん割り付けた方が最終的には効率が良いだろう，という判断から UNPACK 方式を選択した．図 3.12 に UNPACK 運用におけるプロセスの割付状況を示す．UNPACK の問題は，性能の低いプロセスが一つあるとノード全体が足を引っ張られて性能が低下する危険があるということだが，幸い過度な性能低下はほとんど出現していない．

次に問題になったのは DTU 資源不足である．1 ノード 32CPU に対して DTU 資源は 16 しかないので，スレッドを用いないプロセス並列のジョブだけの場合，16 プロセスまでしか入れることができず，結果として CPU が遊んでしまうという問題である．これに対しては，「共有 DTU」という機構を導入し，1 DTU=1 プロセスの制限をはずすことで解決した．（共有 DTU の評価結果は，第 5 回 WG 資料 15-1，15-2 を参照されたい．） 図 3.4 の 2004 年 7 月以前の CPU 稼働率が低いのは，DTU ネックで CPU をフルに使いきれなかったせいである．ただし，共有 DTU は性能に問題があるので，MPI ジョブ専用としている．

システム構成は，初期の試行段階を除き大きな変更はしていない．スレッド利用が増えていないので，32CPU/ノードという構成は，初期のままである．ニーズに応じて，サービスノード，ログインノードにおける処理パラメータを少しいじった程度であろうか．ただ，ノードにユーザジョブのプロセスまたはスレッドをすべて割り付けてしまうと，OS のプロセスが立ち上げられずに性能が劣化する問題（これを「OS Jitter」と呼ぶ？）が現われたため，計算ノード 1 ノードあたりの利用可能 CPU 数を 31 に制限した．このあたりの資源の有効利用についてはベンダー側に何とかしてもらいたいところではある．

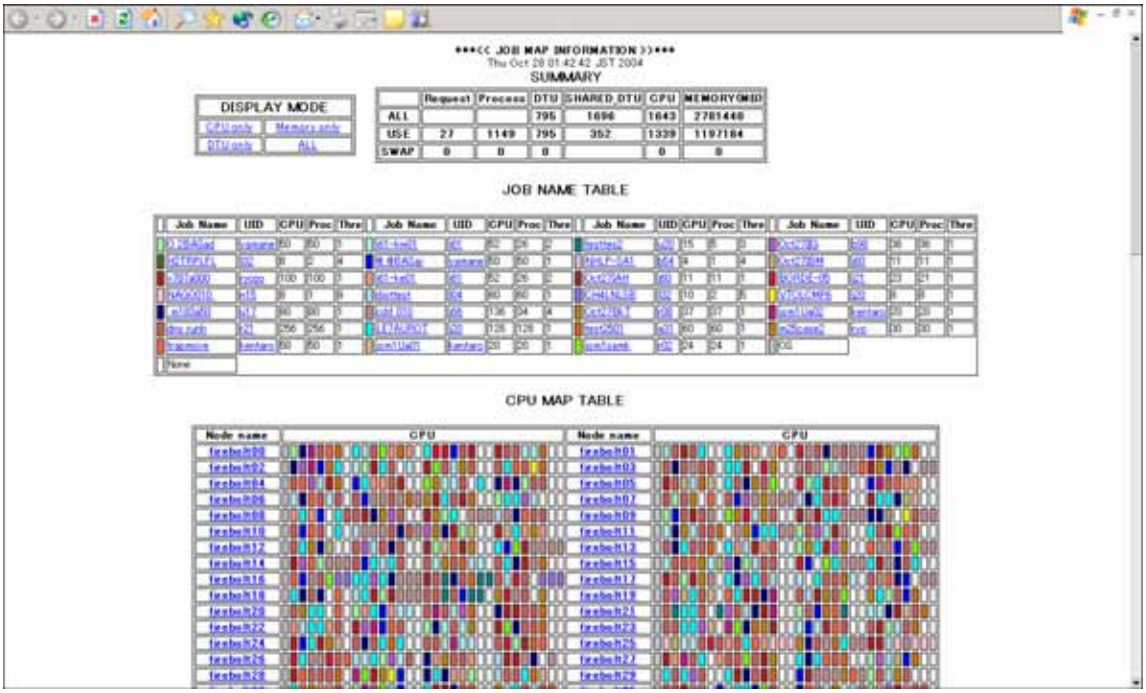


図 3.12　UNPACK 運用におけるプロセスの割付

図 3.3 の CPU 使用時間と実効 CPU 時間の間に乖離がある問題，他のジョブの存在により性能がブレる問題の原因は，当初，SMP 特有のメモリの取り合いによるコンテンションだと思われた．しかし，分析の結果（第 5 回 WG 資料 2-1，2-2），DTU における転送待ちが主要因であることがわかった．これについては，ジョブ毎に転送のパケットサイズが不均等で大容量転送ジョブがあると待たされてしまうことから，転送量の均等化を行うことで転送待ちを多少は緩和することができた（第 5 回 WG 資料 31-1，31-2）．SMP といえばメモリネックという先入観があったので，これはある意味で思わぬところに落とし穴があったという印象である．バンド幅を増やせば解決するのかもしれないが，スケジューリングと転送アルゴリズムによりブレ幅が大きくなっている可能性があり，コストとの兼ね合いもあるので次の世代においては解決すべき課題である．

(3) システム管理

運用管理と空調などの設備との連携を完全な形で取り入れたのも，目立たないところではあるが，このシステムの大きな特徴である．大規模システムになると，個別の機器の電源のオンオフやそのシーケンスなどは大きな手間であるしオペミスも発生しやすい．そこで，今回のシステムでは，図 3.13 にあるように，ほとんどすべての機器をネットワークで接続し，設備管理 PC を通じて監視・指令を送ることにより，電源のオンオフなどの処理を完全に自動化した．スパコンの昨今の電力問題は深刻の度合いを増しているが，空調なども含めた設備全体としての節約を考える時代に来ているのであろうと思われる．実際，このシステムでは前回の数値風洞システムの電力使用の約 6 割程度で済んでいる．このようなシステムの課題として，初期コストの問題は一番大きいと思うが，一方で，こういった仕組みを作るためのインターフェースが意外にきちっとしていないというのが，実際にやってみてわかったことである．

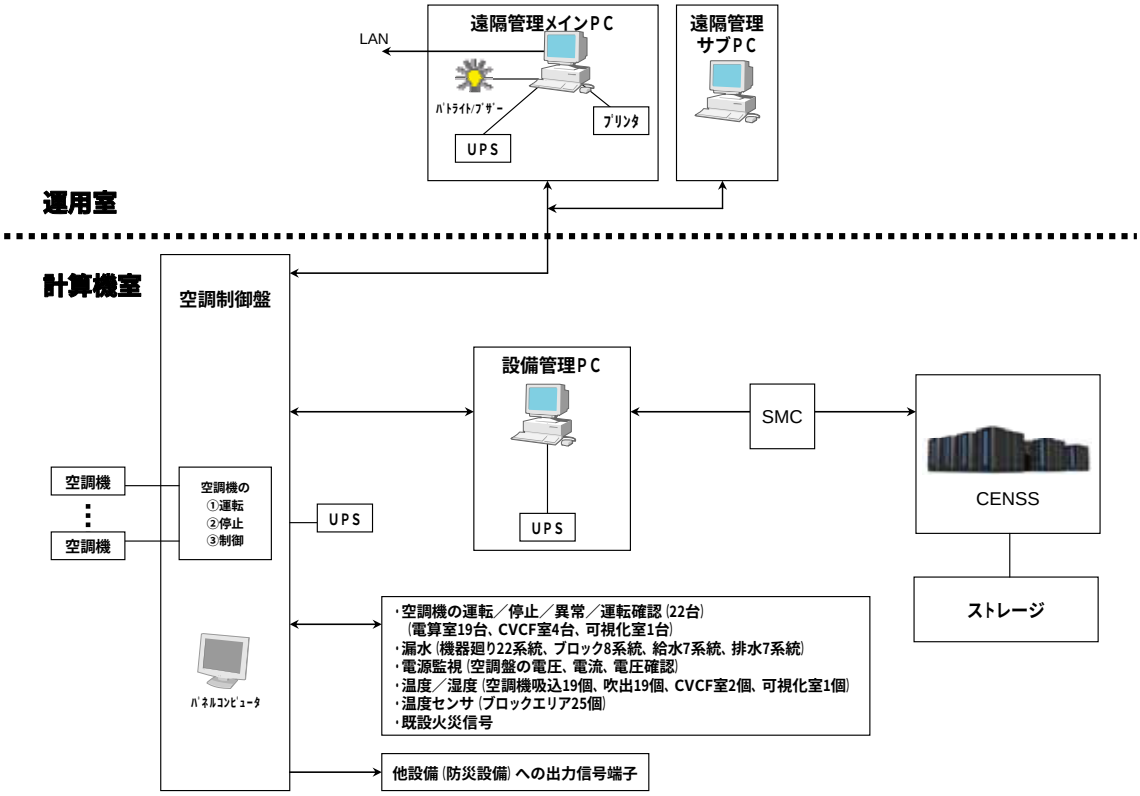


図 3.13　設備自動運転システム

運用管理からみたときの Solaris OS は，UNIX OS としての成熟度も高いので（いわゆる「枯れた」OS なので），信頼性という点ではかなり高いのではないかと思われる．障害の遍歴は図 3.1 の通りであるが，安定期に入った今日では，全系ダウンに繋がる致命的障害はほとんど発生していない．ただし，HPC との融和性の向上（例えば IPL 時間の短縮など）については，今までの VPP で培ってきた技術．ノウハウを活かす意味でも，（実現困難かもしれないが）今以上に是非取り組んでいただきたいところである．

(4) 入出力・ストレージ系

システム構成の観点からいえば，単体 IO ノードによる入出力は，1 ファイルは物理的にも 1 ファイルにできる，ジョブスケジューラから見たファイル配置が均一でスケジューリングし易い，IO ノードの実性能を推定し易い，などのメリットがあるが，IO ノードの障害 ⇒ 全系ダウンにつながる点で，ハードウェア的には冗長構成にしているとはいえある種の弱点である．この弱点を克服するには，ファイルシステム（つまり，ソフトウェア）として冗長構成とすることが必要となるが，大規模かつ高速 IO を必要とする HPC システムにおいて，このような冗長構成を採った例は知られていない．今後の課題と考えられる．

データの入出力性能，ディスク＋テープのストレージについては，容量に対する不満は少なく，図 3.10 にあるように総量は徐々に増加しているが想定の範囲内である．ユーザジョブによる IO 頻度は著しく増加しているが，入出力性能，安定性，信頼性に関する障害は少ない．ただ，C ライクなプログラミングスタイルや MPI の影響だと思われるが，小規模ファイルが非常に多

く，ファイル総数は数 100 万に達している．小規模ファイルの高速処理は全てのファイルシステムにとって不得意なものであるが，今後は小規模多数ファイルの処理系についても，性能向上の努力が必要になっていくと考えられる．また，もうひとつの課題としては，システム外も含めたファイル共有の要望が出始めているので, 何とか対応して行きたいと考えている. SRFS: Shared Rapid File System を拡張した SRFS on Ether の活用等が解決の一助となるかもしれない（第 5 回 WG 資料 1）.

(5) 処理性能

性能面では，特定の実アプリケーションについて，実効効率で 15%，スケールアップ性能で実効 1TFLOPS 越えを達成した．実効 1TFLOPS 越えは，CeNSS の導入当初からの大きな目標であり，これを達成できたことは一つの成果だと考えている．しかし,「実効効率がせいぜい 15%，しかも特定のアプリケーションで」というのは,「ベクトル機の半分程度の実効性能」というフレコミでスタートしたシステムとしては不満が残り，この点に関しベンダーの今後の奮起をぜひ期待したい. 特に, スレッド並列については, 前述の経緯のところでも触れたように, 現在ユーザのスレッド離れが進んでおり，昨今の CPU マルチコア化路線においてスレッド利用が議論されている中で，どのように位置づけて行くかを早急に明確化する必要があろう．性能について詳しくは，スレッド並列 WG の報告書をご参照いただきたい.

(6) 利用性

CeNSS では，NWT 時代に指摘されていた性能面以外での使いにくさ・移植性の問題や近年の計算機のシステムとしての多様化やすそ野の広がりを視野に，旧来型スパコンからの脱皮，旧来型のトップダウン的なアプローチと PC クラスタに代表されるボトムアップのアプローチの融合を目指して，標準化・汎用化の促進や PC 環境との連続性構築を図るという命題があった．そのために，構成的な配慮（２．で前述）もしたが，オープンソースの実装や ISV アプリケーションの導入，ウェブからの利用の構築などに積極的に取り組んだ.

オープンソースや ISV アプリへの親和性という点では Solaris OS は極めて有効で, 移植性も高いし信頼性も高い．ただし，スパコンとしての性能を出そうとすると，ソースの再コンパイルが必要となり，ISV アプリの場合は問題が多かった．しかし，スカラーシステムの使いやすさを今後とも活かして行こうと思ったとき，ISV アプリの移植の問題はやはり今後とも無視できないと考えており，HPC システムで ISV アプリを動かす確固たるメリットを構築して行く必要があると考えている.

(7) プログラミングとチューニング

大規模並列プログラミングに対して，プロセス並列とスレッド並列を組み合わせるハイブリッドプログラミングや性能チューニング技術等の導入は比較的スムーズに行われたと思われる. 特に，性能評価やチューニングを通じて，実性能面へのユーザの関心が集まるようになったのは，凡庸なプログラムを少しでも減らして設備としての有効利用を図る意味でも有効である. しかし，性能向上が困難なプログラムが実際に存在し，どのような有効打を打ったらよいか，という宿題も明らかになって来ており，今後の課題と捉えている.

経験によれば，チューニングの効果は，場合によっては非常に大きく出るが，そうでない場合もある．チューニングの原則論や事例集のようなものもあるようだが，ケースバイケースや問題規模によるということもあり，勘所を掴むのがなかなか難しい．「スカラチューニングは，ベクトルに比べて難しい．何をやってよいかわからない．フォルスシェアリングなんて特に！」という声が多かったような気がする．経験と勘以外の形式知を如何に積み上げるかが課題である．また，実際にチューニングを行う場合に，他ジョブの影響を受けて（性能ぶれ等により）チューニングによる差分が明確に出ない場合もあり，システムとしてチューニングしやすいような環境，有効なツールを如何に構築して行くかは課題である．

４．高度有効利用の工夫と対策

今回のシステムは，運用側にとってはかなりの部分が試行錯誤の連続であり，どうすればシステムの潜在能力をフルに引き出せるかは，ベクトルや汎用機のような確立されたものがあるわけではない．そういう中で，資源の有効利用と利用性の向上を中心に具体的な工夫と対策に取り組んで来た．以下では，そのうちの幾つかを例として挙げてみたい．

(1) ジョブスケジューラ NSJS の開発と利用

NSJS とは, NQS のセンター出口ルーチンとしてジョブの各種事象発生契機において, 図 4.1 に示すように，各種の OS 機能と連携し，CeNSS 独自のジョブスケジューリング機能を実現するプログラムである．NSJS 開発に際し，ジョブスケジューリングに必要となる新たなトリガーや多数のインターフェースが提供された．

NSJS 開発における基本的な思想は以下のとおりである．

- CeNSS 運用の最重要課題である「システムの高度有効利用を最大限に図る」運用システムの実現.
- 小規模／大規模ジョブが公平に混在実行できること．
- デバッグジョブの実行環境を最善なものとし，ユーザのプログラム生産性を向上し得ること．

NSJS の主たる機能を列挙すると以下のとおりである．

- システム運用計画に従った運転開始／終了処理のスケジューリング
- CeNSS 独自アルゴリズムによるジョブ実行起動のスケジューリング
- ジョブ実行に必要となる各種システム資源の割当／解放等のスケジューリング

NSJS は，CeNSS が有する超高速処理性能を十分に引き出すために本質的な役割を果たしている．一方，センターの経営方針に則り，システムの運用規則を定め，秩序ある運用を実現するためには，各種の運用機能が必要である．実運用においては，規則どおりの運用では対処し得ない事態が多々発生する．そのような事態に即応できる柔軟性も要求されるスケジューラが必要となる．NSJS は多数有する各種パラメータ操作によりこれらの要求に応えている．

NSJS は，平成 15 年 4 月より実運用に供したが，現在まで障害発生も非常に少なく，高品質なシステムプログラムとして完成され，順調に安定稼動している．NSJS は，独自開発プログラムという点で，機能拡張や変更等ならびに保守は，必要な時点で容易に早期実施でき，ジョブスケジューラとして一層完成度を高めることが可能である．

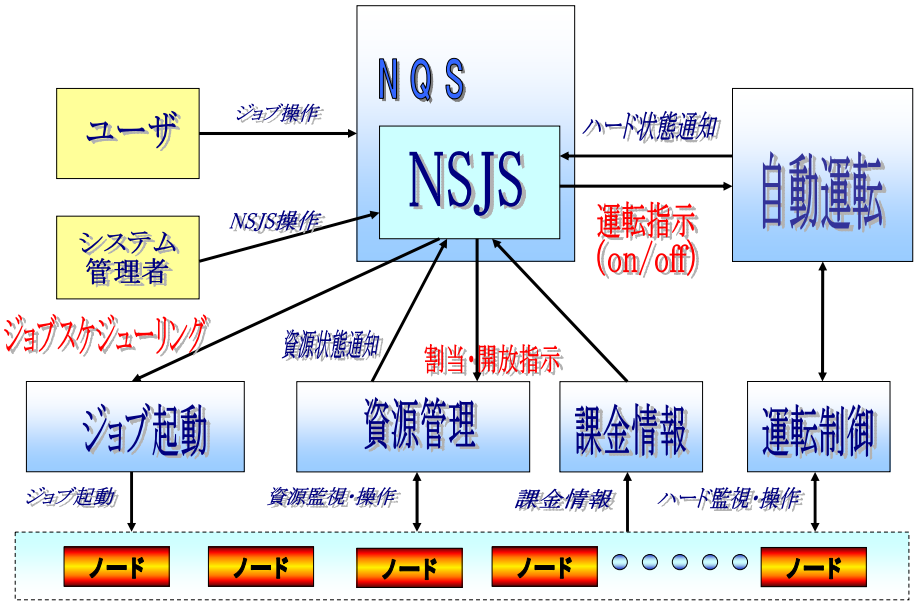


図 4.1　NSJS の位置づけ概念

NSJS のスケジューリング可能項目を以下に列挙する．

- 運用時間帯ごとの運用ノード，運用ユーザおよびジョブ制限値等の運用パラメータを定義する（運用パターン）．
- 1 年 365 日のカレンダーを定義し，それぞれの運用パターンを定義する．
- 不特定多数ユーザによるセンター運用であるが, 画一的ではないユーザ管理が実現する．
- プロジェクト開発ユーザグループをプロジェクトユーザとして一括管理する．
- 設備貸付ユーザを定義でき，ユーザのシステム利用を予算管理する．

NSJS のスケジューリングにおける主な特徴を以下に列挙する．

①CPU 資源割当方式

スケジューラが資源管理する資源の要素は CPU，メモリ，ユーザ DTU コンテキスト，ノード内バリア，ノード間バリアである．CPU の割当方式は最も CeNSS の性能を引き出すことができる割当方式（UNPACK 方式）を採用している．これは極力多数のノードに分散してプロセスを割当てる方式である．図 4.2 に CPU 資源割当方式示す．

②限定ノード実行方式

他ユーザジョブに大きな影響を及ぼす特定ユーザジョブを限定したノードで実行する．

③マルチブロックジョブの資源割当

マルチブロック構造格子用にプロセスごとに CPU やメモリを実際に必要な資源量だけを割当てるスケジューリング方式（使用しない資源は最初から割当ない）である．

④デバッグジョブのスケジューリング

デバッグジョブはレスポンスを保証するために最優先で実行処理する．実行時に割当可能資源が不足している場合には，直近に実行したジョブまたはデバッグジョブと同一ユーザ実行ジョブをスワップアウトさせて実行する．このスケジューリング方式により，デバッグジョブのレスポンスについては劇的に改善された．なお，大多数のデバッグジョブは 10 秒から 30 秒で実行起動できる．

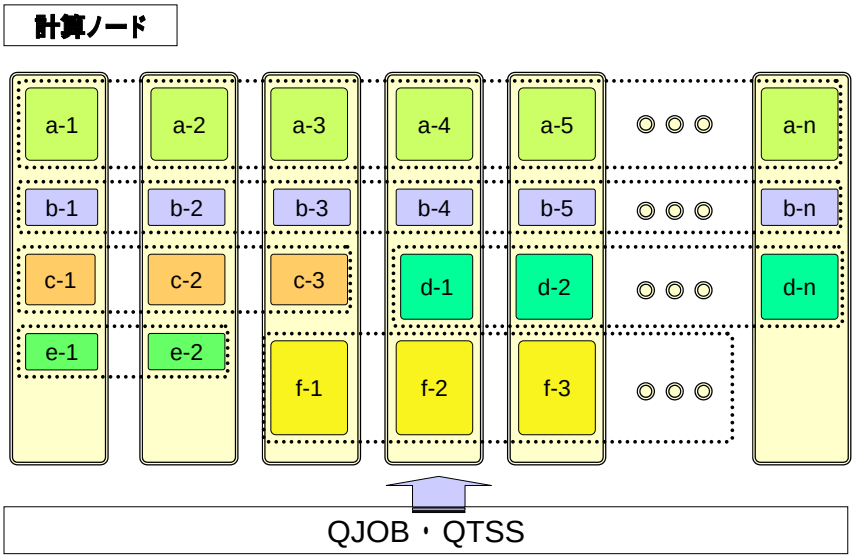


図 4.2　CPU 資源割当方式

(2) ノードの有効利用（2005/2/16 の話題）

システム導入当初より，多数の計算ノード運用において，最も効率的なジョブの割当方式について調査，検討し，試行的に運用を実施した．計算ノードとして運用するノードの資源構成には以下の二種類がある．

Thin ノード(56 ノード)：　32CPU，２DTU，メモリ（64GB）

Fat ノード（4 ノード）：　64CPU，１DTU，メモリ（128GB）

当初, Thin ノードと Fat ノードを混在して運用してみた. Fat ノードは, DTU 資源量が Thin ノードの半分で，ジョブのプロセスも 2 倍割当られるので，通信やメモリアクセス競合が Thin より激しく，性能も悪い．このため，混在運用すると，システム全体は Fat の性能になり，大多数の Thin が低性能に合わされる状況が分かった．この結果，Thin と Fat は独立に運用することとした．

ジョブの要求する各ノードの CPU 資源をジョブに割当る方式として以下の４種類がある．

<u>Absolutely PACK 方式</u>：リクエストを構成するプロセスすべてを同一のノードに割り当てる方式であり，すべてのプロセスを１つのノード に配置できない場合は，実行可能リクエストとして選択しない．

<u>PACK 方式</u>：リクエストを構成するプロセスすべてを同一のノードに割り当てる方式であり，すべてのプロセスを１つのノード に配置できない場合は，配置できないプロセスを他のノード に割り当てる式．

<u>Absolutely UNPACK 方式</u>：1 ノードに 1 プロセスを割り当てる方式であり，プロセス数が割り当て可能なノード 数より多い場合は，実行可能リクエストとして選択しない．

<u>UNPACK 方式</u>：1 ノードに 1 プロセスを割り当てる方式であり，プロセス数が割り当て可能なノード 数より多い場合は，1 ノード に 2 つ以上のプロセスを割り当てる．

CeNSS のジョブ形態および各割当方式の性能調査や試行運用を経て，導入初期から UNPACK 方式を採用している．

次に，ノード内のバリア資源について運用当初は，ジョブへの CPU 割当が自在に行えないという理不尽があった．当初，ノード内に 8CPU を実装する SB においてバリアグループが二つ存在し，同一リクエストの各プロセスはバリアグループを跨いで CPU を割当てることができなかった．このため，CPU が IDLE 状態でもプロセスに割当られない状況が発生し，稼働率低下を引き起こしていた．この問題は，バリアグループ方式という垣根を取り除き解消した．

システムの高度有効利用を阻む問題として，DTU 資源枯渇問題があった．並列プロセスを実行する場合には，プロセスごとに１DTU コンテキストを割当てる必要がある．１ノードは１６ DTU コンテキスト有し，割当て可能な最大プロセス数は 16 個であった．このため，1 プロセス 1 スレッド構成のリクエストが多いと, 1 ノード内の半数の CPU が DTU 枯渇で idle 状態となった．この問題は導入から約 2 年後に共有 DTU 機能が提供され解消した．共有 DTU とは，16 個のコンテキストのうち，1 つを複数プロセスで共有するというものである．この共有は CENSS における約 4 割強の MPI ジョブに有効な機能となった．

以上のような効率的運用を阻害する問題を解消し，平均 CPU 稼働率 90%を年間クリアしている．

(3) NS アクション管理，WANS，CMS（2005/2/16 の話題）

NS アクション管理とは，センタで扱う障害，Q&A，要望を，そのやりとり，状態を含めてウェブインターフェースにて操作可能なデータベース化し，運用管理の手間を削減するものである．本ツールの運用により，日々発生する種々なインシデントは的確に処理・対応され，かつ，これらの処理･対応状況は逐一運用管理チーム全員に伝わるので，情報共有ならびに，それぞれのノウハウの蓄積にもなり，システム運用管理に非常に有効なツールとなっている．現時点では，NS アクション管理に独自機能の追加と大幅な機能改善の理由から，新たなツールとして NS インシデント管理（NSIM）を独自開発し，運用に供している．

主な機能として，NSIM は図 4.3 にあるように，障害，Q&A，要望ならびに定例業務等の発生インシデントの登録および対応状況の記録，図 4.4 にあるような検索による状況表示が可能である．また，画面に検索・抽出した内容を CVS データにダウンロードと累計グラフを表示でき，会議資料として必要となる各種の定型帳表を１クリックで作成できる．

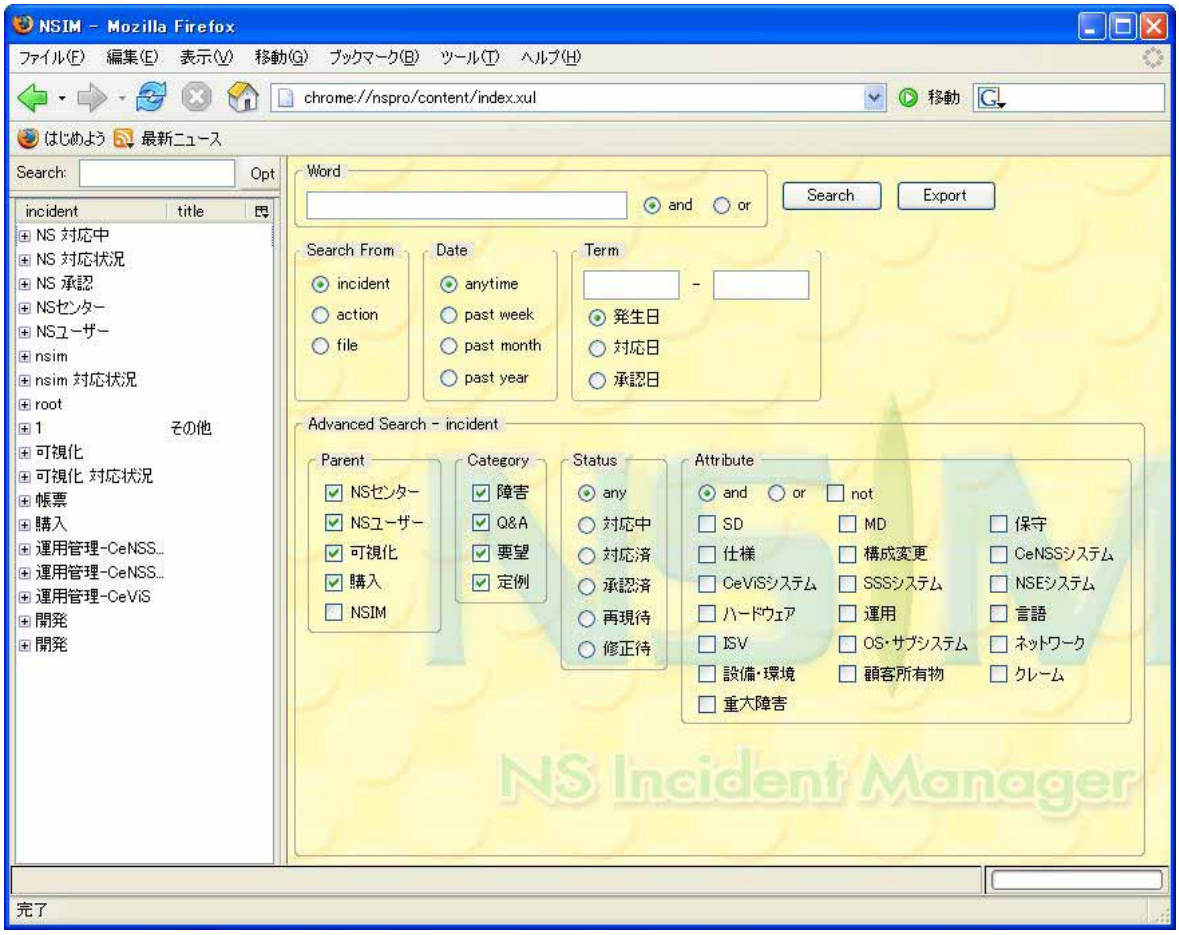


図 4.3　NS インシデント管理（NSIM）

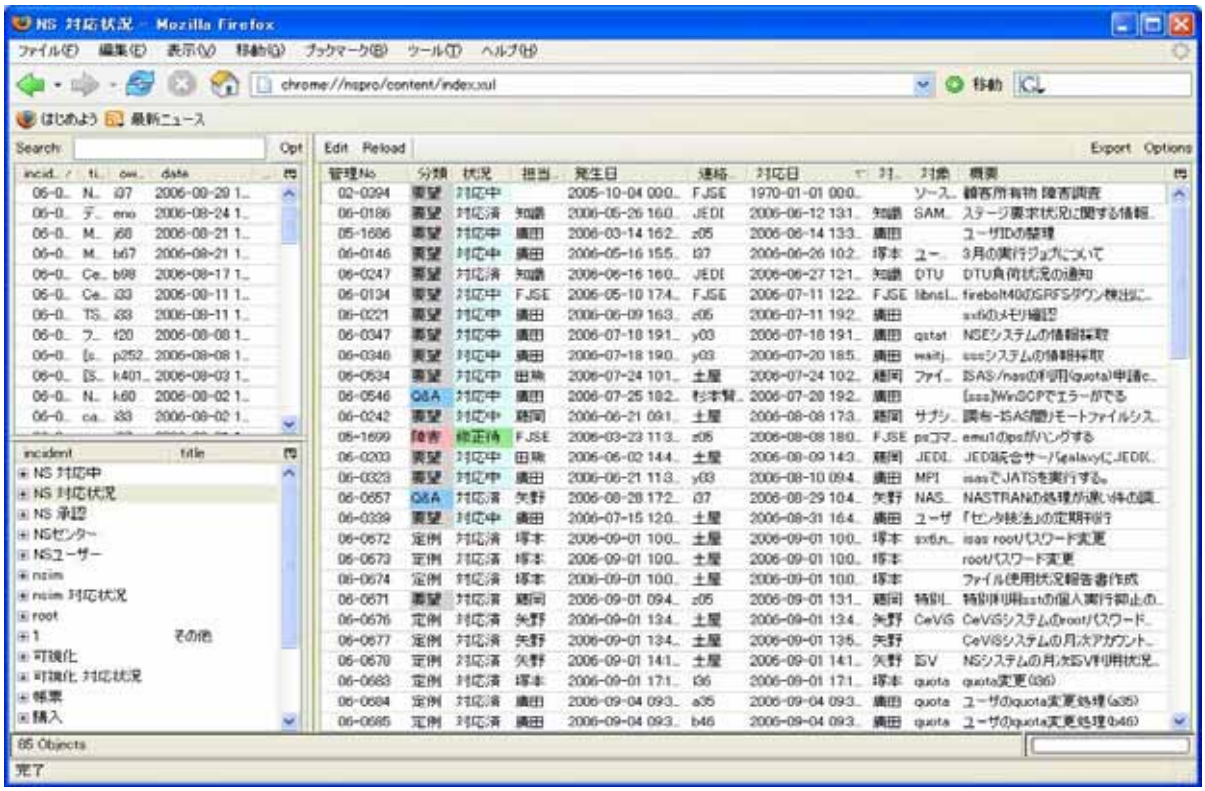


図 4.4　　NSIM 検索画面

WANS(Web Access NS)とは，ウェブブラウザから CeNSS にアクセスするツールである．普及したウェブ技術を用いて，ユーザがジョブの状況確認や簡単なファイル操作を手軽に利用可能にすることを企図した．WANS の主な機能として，ユーザは CeNSS のシステム混雑状況確認，ファイル編集や操作ならびにジョブの投入から実行結果検索等のジョブ操作が行える．ウェブブラウザから WANS にアクセスし，ログイン名，パスワードが正しく入力されると，図 4.5 の WANS のメニュー画面が表示される．同画面から，たとえば【File Tool】メニューをクリックすると，図 4.6 に示すファイル操作画面が表示される．ファイル操作画面からは，ファイルのブラウズ，編集，複写等，各種のファイル操作がウィンドウズ的に行える．図 4.7 はジョブ操作画面を示す．同図に示されるとおり，ジョブ操作画面からはユーザジョブ状態表示，ジョブキャンセル，ならびに結果表示や削除等の各種ジョブ操作が行える．昨今の技術を使えばウェブからほとんどの必要な操作を行えるように作り込むことも可能であろうが，メンテナンスの容易さやセキュリティを考えて限定的な操作のみを可能にしているのが現状である．なお，WANS の拡張版として，携帯端末からの操作が可能な iWANS という仕組みも開発している．

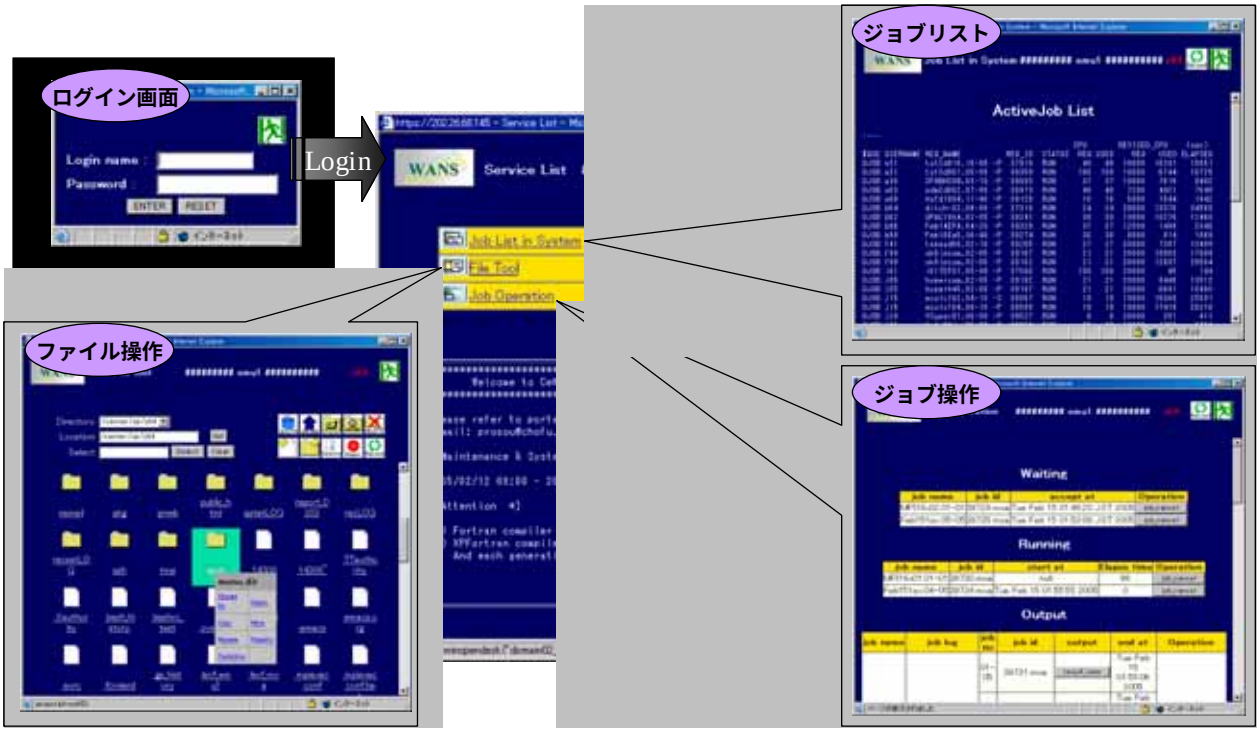


図 4.5　WANS のメニュー画面

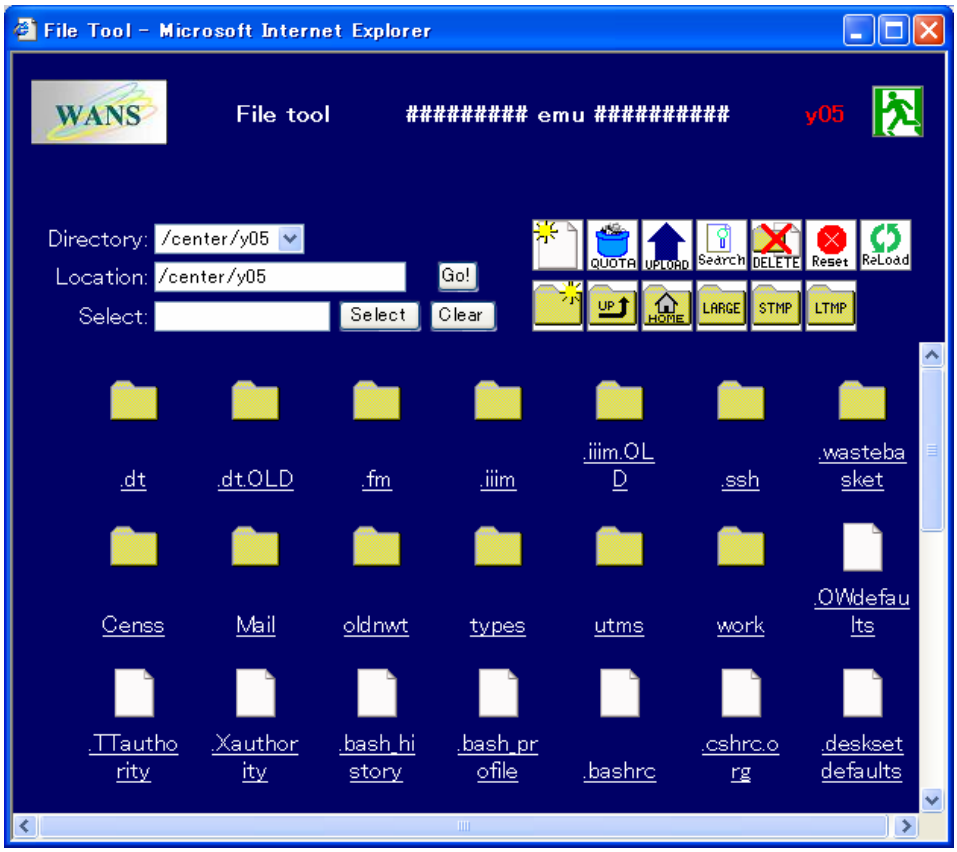


図 4.6　ファイル操作画面

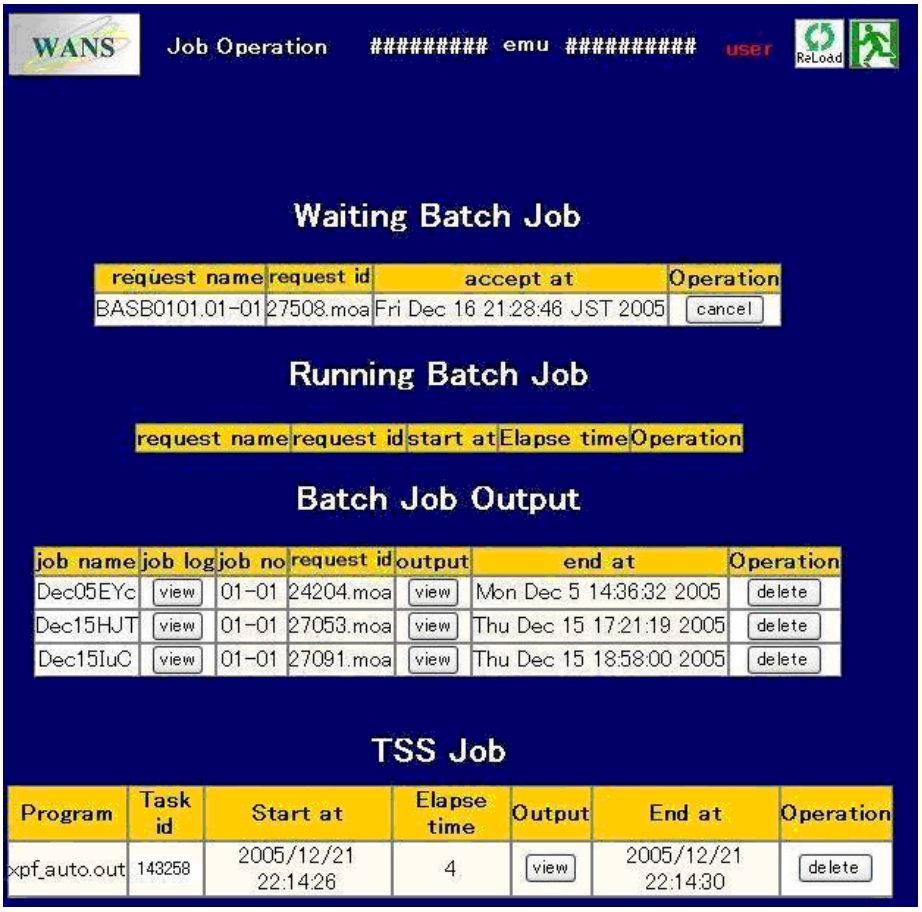


図 4.7　ジョブ操作画面

CMS（CeNSS Monitoring System）とは，CeNNS におけるシステムの利用状態をウェブブラウザによりグラフィカルに表示する管理者向けツール群（一部ユーザにも開放）の総称である．WANS 同様，普及したウェブ技術を用いて管理者が利用状況を手軽に随時確認できるようにすることを企図したものである．以前のシステムと異なり，今回のシステムは，管理すべき資源量が多く，また，どのような状態になるのかの予想もつかなかったので，多角的にシステムの状態がウオッチできるようにした．図 4.8 に示すとおり，CeNSS System Usage(CPU 利用率)，CeNSS Resource Information（ノード資源利用率），CeNSS Job Map(ジョブ CPU 割当状況)，CeNSS System Monitor(資源利用状況の履歴)，CeNSS Access Log Analysys（ポータルサイトアクセスの解析）ならびに CeNSS CPU Information(CPU 稼動状況)の６つのメニューを有しており，詳細なシステム稼働状況をダイナミックにモニタリングすることができる．図 4.9 の CeNSS System Monitor からは，直近１時間前，１日前，１週間前，１ヶ月前のシステム資源（CPU，メモリ，DTU）利用履歴が確認できる．図 4.10 には，処理の流れを示す．

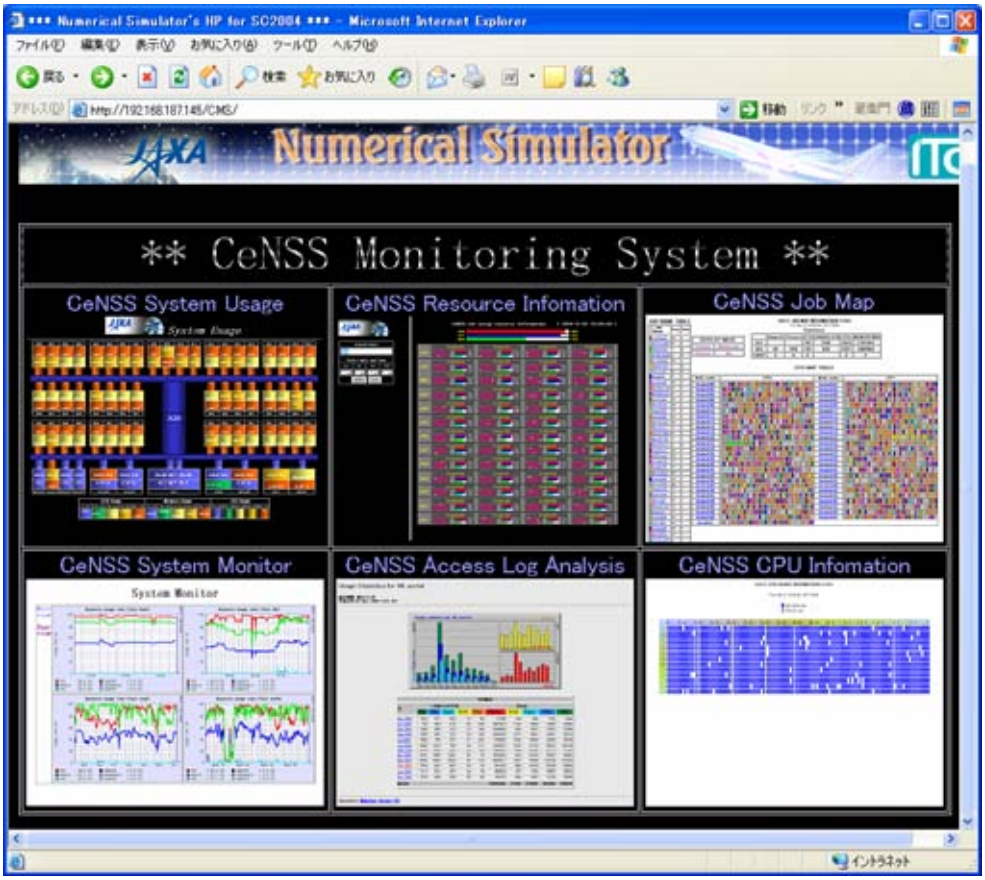


図 4.8　CeNSS Monitoring System のメニュー

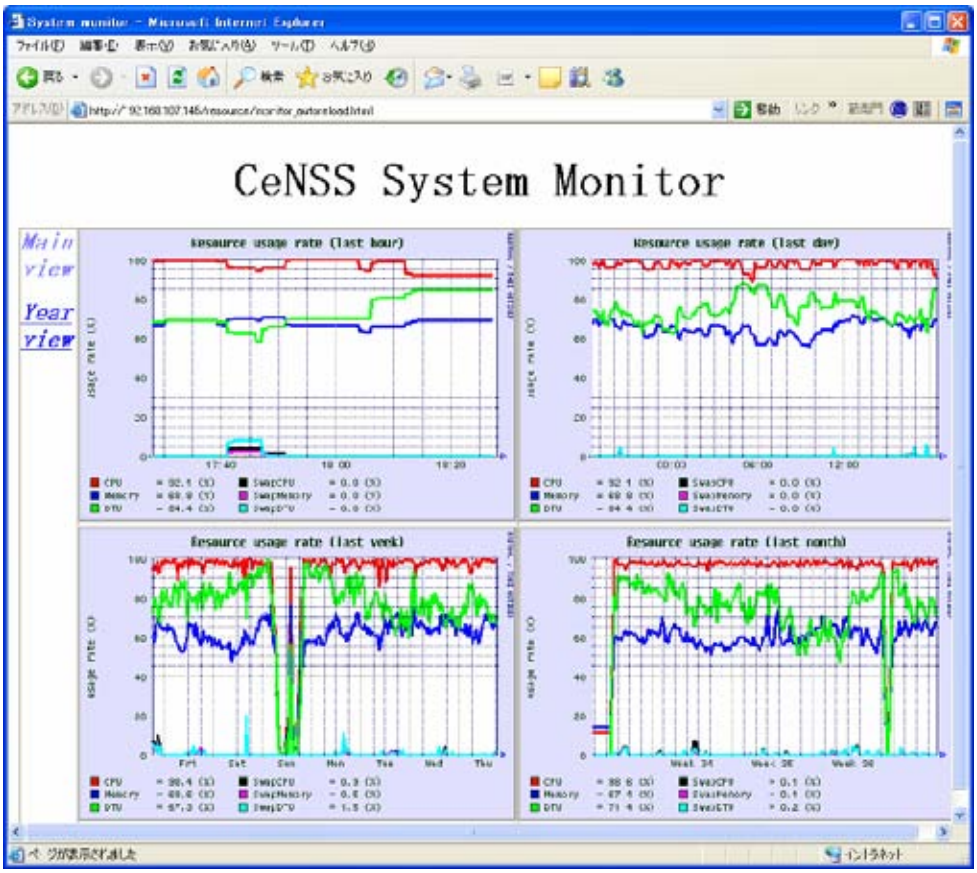


図 4.9　CeNSS System Monitor

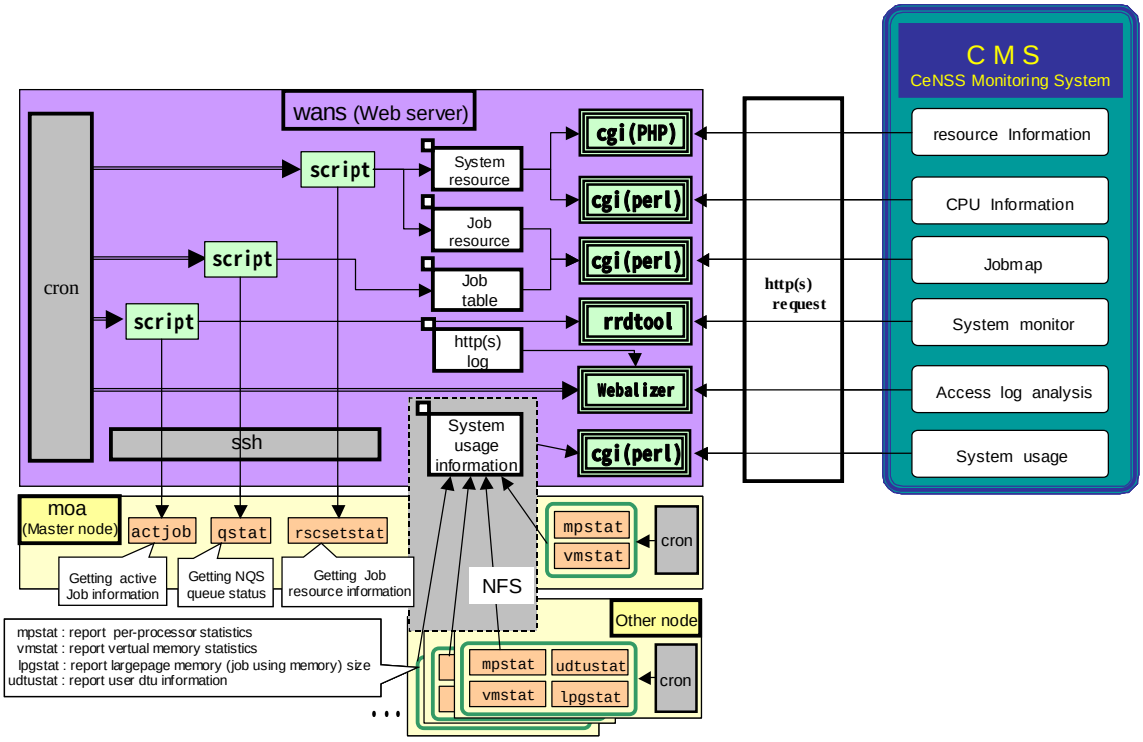


図 4.10　CeNSS System Monitor の処理の流れ

(4) オープンソースの利用

導入当初，以下の観点でオープンソースの活用を方針として，運用環境の改善にこれまで努めてきた．

(ア) 研究開発環境の標準になりつつある

(イ) 利用者の利便性向上（ツールの追加）

(ウ) 管理者の利便性向上（ツールの追加）

(エ) カスタマイズ可能（使えるものは活用する）

OS として Solaris を採用したことにより，豊富なオープンソースを苦労なく使えるようになったメリットは大きい. NWT のプロプライエタリ OS の時に味わった苦労を考えると隔世の感がある．

現在導入している，主な利用者向オープンソースを以下に示す．

acrobatreader，a2ps，canna ，gcc，ghostscript，ghostview，gmake，gnuplot，gv，gzip，g++，g77，ImageMagick，Java，kinput2，kterm，openGL，ruby，ssh，scp，screen，sftp，tcl/tk，tex，wnn xemacs ，xv

また，オープンソースをベースにした高速コピーコマンドの実現や，管理者向けとして OpenLDAP, RRDtools, swatch を活用した大規模運用管理ソリューションの実現など, CeNSS では最大限にオープンソースを活用した．詳細は，表 4.11 の運用工夫のために開発したツール類の一覧を参照されたい．

オープンソースの課題についても言及しておきたい．ユーザ認証という運用の根幹にかかわる部分に，自ら行った評価結果により，当時大規模クラスタの多数ノードに性能面で耐えられうる唯一の LDAP と判断し，OpenLDAP の採用に踏み切った．RAS を考慮し，マスタ・スレーブと冗長構成を取っているが，データの同期化，更新に失敗し，何度か運用を停止せざるを得ない重大障害を引き起こしている．オープンソースがゆえ，ベンダのサポートはなく，今でも根本解決には至らず，運用の工夫でなんとか回避している状況である．

以前は，ツールに過ぎなかったが，今やオープンソースでなんでも揃う時代である．

- □ オペレーティングシステム　Linux，Open Solaris…
- □ ファイルシステム　AFS，GFS，Lustre，XFS…
- □ データーベース　PostgreSQL，MySQL…
- □ 認証　OpenLDAP，GSI
- □ グリッド　Globus Toolkit…

ただし，先ほど述べたとおり，イニシャルコストの安さからの魅力だけで判断すると，無料ほど高いものはないという教訓に結びつくこともあり，オープンソースの利用には慎重な判断が必要である．

(5) その他

表 4.11 に, 今まで紹介したものも含めて運用で開発したツールや仕組みを一覧表にまとめた.

表 4.11　運用工夫のために開発したツール類の一覧

分類	名称	開発元	機能・目的	効果・備考
ｼﾞｮﾌﾞ	NSJS	JAXA	柔軟なジョブスケジューリングと資源割当を行い，稼働率の向上，適切なターンアラウンドを実現する. NQS 出口を利用して以下の機能を実現. 実行順序保障／マルチブロック／概括並列／プロジェクト管理／ プレステージ／年間カレンダ／運転制御など	センタ主導による柔軟なジョブスケジューリングを実現し，稼働率を向上. 他サイトで流用するにはカスタマイズが必要.
管理	NS アクション， 管理	JAXA FJ OSS	センタで取り扱う障害，ＱＡ，要望の情報を，そのやりとり，状態含めて Web UI にて操作可能なデーターベース化し，運用管理コストを削減する．いわゆるバグトラッキングシステム，リクエストトラッカー. OSS の RRDtool，PHP 等を利用	記入もれや間違い，確認忘れをなくし，対応をスピードアップ. 他サイトで流用するにはカスタマイズが必要.
管理	CMS	JAXA FJ	CeNSS Monitoring System の略. ジョブの割当状況や稼動情報を容易に把握するため，Ｗｅｂベースでジョブ状態を表示する．用途に応じて，5 種類開発.	稼動状況の容易な把握，問題点の早期発見が可能. ユーザへの運用状況の提供. イベントなどでの展示. 他サイトで流用するにはカスタマイズが必要.
管理	NSLDAP	JAXA OSS	LDAP によるユーザ情報の一元管理を行うことにより，運用管理コストを削減する. OSS の OpenLDAP を利用.	一般に行われる UID 管理だけではなく，失効管理，quota 情報，使用期間，課金情報，住所，連絡先を一元管理し，かつそれらをＧＵＩ操作可能とし管理業務を効率化. 他サイトで流用するにはカスタマイズが必要.
管理	ログ監視	JAXA OSS	/var/adm/messages 等に出力される重要なメッセージを監視し，即時通報（メール，オペコール），即時アクション（コマンド発行）を行う. OSS の Swatch を利用.	重大障害の早期発見，早期対応が可能. 他サイトでも容易に流用可能.
管理	unyo_check	JAXA FJ	保守前後での確認すべき運用状態などのチェックを自動化（スクリプト化）し，手順ミス，考慮漏れを防ぐ.	環境の戻し忘れなど，保守作業後の単純ミスを撲滅. 他サイトで流用するにはカスタマイズが必要.
利用	WANS	JAXA	Web Acess to NS の略. Web ベースで特別なクライアントを必要としない統一的な GUI で，JAXA 内外からのでシステム利用を可能とする. ジョブ投入／ファイル操作／簡易可視化など	場所を問わず，システムの利用が可能. 海外からの利用実績もある. 他サイトで流用するにはカスタマイズが必要.
利用	iWANS	JAXA	WANS のサブセットとして，携帯電話からシステム利用を可能とする．現状，i-mode のみ対応.	ちょっとした空き時間に場所を問わず，システムの利用が可能. 他サイトで流用するにはカスタマイズが必要.
利用	コンパイラ 世代管理	JAXA	コンパイル環境について，新機能重視なら最新版，安定性重視なら１世代前の版，というようにユーザがコンパイラ選択できる環境を提供する.	安全なコンパイラバージョンアップが可能. 他サイトでも容易に流用可能.
利用	TUTRO	JAXA FJ	TUning TRaining Online sysytem の略. 利用者のスキル向上とそれによる稼働率向上を目指し，ユーザフォーラムの内容やその他有益な教育を，e-lerning（ストリーミング）として提供. 富士通製 INTERNET NAVIGWARE を利用.	利用者のスキル向上．いつでも都合の良いときに利用可能. 他サイトで流用するにはカスタマイズが必要.
性能	プロファイラ 診断ツール	JAXA	ジョブの規模が大きくなるとプロファイラの出力情報が大きくなり，解析するのに時間，手間がかかる．そのため，プロファイラの診断結果を分析し，問題箇所を指摘する機能を実現する.	開発中.

分類	名称	開発元	機能・目的	効果・備考
利用	NS コマンド	JAXA	利用者の利便性向上のため，簡単にジョブを投入する機構やジョブ状態の整形出力などのコマンド群を NS コマンドとして提供. 例えば以下. 1)acctjob sysrscstat, rscsetstat *1 の情報を 80x24 の画面で参照可能な情報に整形出力する. 2)lsize LPG を考慮したセクションバイト数の出力し，プログラムの使用メモリの概算がわかります．ただし，スタック，動的配列などは含まれない. 3)ns-shell qsub スクリプトを簡素化．Shell を知らなくてもジョブ投入を可能とする.	システムの利用性向上. (簡単な NS コマンドだけ覚えれば，システムが使える). ns-shell を除き他サイトでも容易に流用可能. *1：複数ユーザから同時実行された場合の効率を考慮し，sysrscstat，rscsetstat の情報は定期的(1min)にログ保存し，その情報を整形している.
利用	高速コピー	JAXA OSS	通常の cp コマンドは io 長が小さく(8K,256K,etc)，SRFS に適していない．そのため，ファイルシステムと認識し，最適な IO 長を選択する高速な cp コマンドを実現する．GNU の fileutil を改造.	SRFS 間でのコピー性能が１１倍（23.4MB/s→258.9MB/s）に改善. 他サイトで流用するにはカスタマイズが必要.

５．おわりに

本稿では，PRIMEPOWER HPC2500 による JAXA 大規模 SMP クラスタシステムの運用について，統計情報や事例を中心に紹介した．旧航技研の時代から，ベクトルスパコンの運用経験は 20 年近くある JAXA だが，スカラーシステムについては，まだまだ駆け出しである．また，スカラーシステムは多様性が大きいので，運用のセオリーや王道のようなものがあるわけではない．今後とも試行錯誤を繰り返しながら，ベンダーやユーザとともに経験と実績を積み上げて行くしかない．そういった意味で，本 WG の活動を通じて，同じ悩み・課題を持ちシステムの高度利用に日々尽力されている方々と議論し，知見を深め，共通の認識を持つことができたのは極めて有意義であった.

以前のようにステレオタイプ的なスパコンの定義がはっきりしなくなっている今日ではあるが，時代やマシンアーキが変わっても HPC の本質は変わらないと思っている．超高速であること，科学技術の革新に寄与するアプリケーションの存在，共有資源としてのインフラ性，何らかの意味で先進的であること，などがその中心要素であろう．今後とも HPC の本質とは何かを常に念頭に置きつつシステムの整備・運用に取り組み，HPC の真髄を極めて行きたい.

参考文献

(1) 松尾，航空宇宙技術研究所共用計算機システム「数値シミュレータ III」－その構想とシステム概要，日本航空宇宙学会誌第 50 巻第 586 号，pp.262-269，2002.

(2) 松尾,「数値シミュレータ III」－システム性能特性/航空宇宙への初期応用成果と今後の展望，日本航空宇宙学会誌第 52 巻第 606 号，pp.175-185，2004.

以上

SS研大規模SMP運用WG

添付資料40-1
HPC Portalにいただいたコメントに対して

2006年10月26日
富士通株式会社
科学ソリューション統括部
相澤

コメント1:

SSLに対応した際、ファイルのアップロード遅延が発生。OSの違い(Windows2000とWindowsXp)によりアップロード時間が大幅(10倍程度)に違った(原子力機構様)

対応策:

アップロードCGI処理ロジックを改善することで2倍以上の性能改善。2007年1月以降に改版版ツールを提供予定。
注)クライアントPCの性能により、アップロード時間に差が生じることがあります。

コメント2:

パスワード変更機能を追加してほしい。(原子力機構様)

対応策:

原子力機構様向けに作成し、提供済です。
HPC Portalサービスの構築ツールとして、パスワード変更機能の部品化を検討中です。

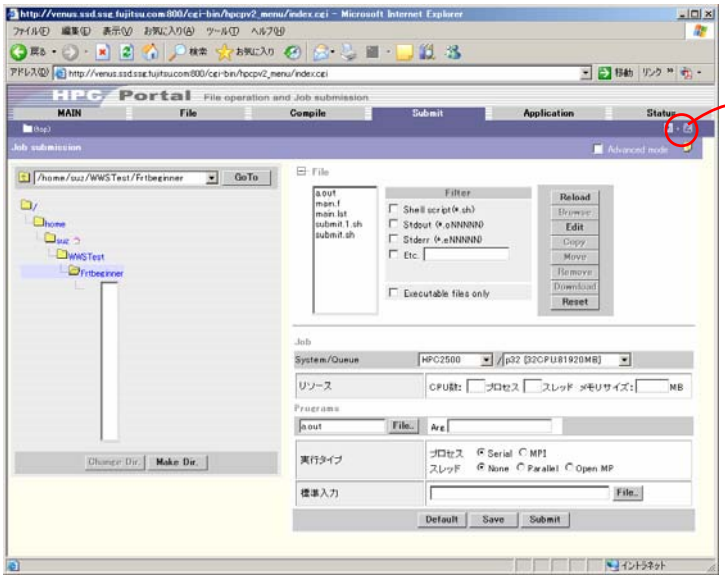
コメント3:

例えばジョブ投入の際にマニュアルを参照したい場合、一旦メインに戻らないとならないのは不便。マニュアルのトップページを別ウィンドウで表示する等の手段は?(名古屋大学様)

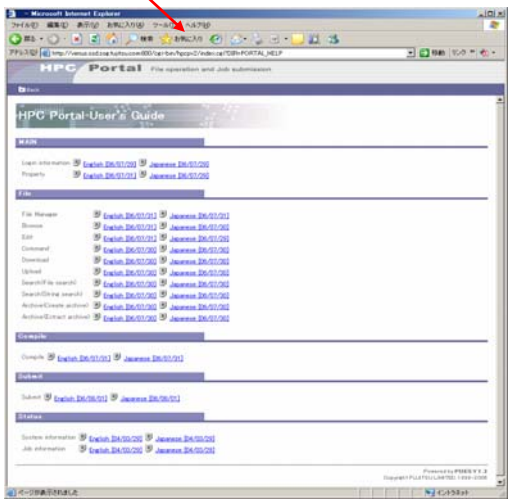
対応策:

導入カストマイズで以下の機能が提供可能です
(1)チャンネルフレームに、マニュアルのTopページへのリンクをアイコン等で表示する
(2)上記アイコンをクリックすると、マニュアルのTopページを別ウィンドウで表示する
ポータルの画面例を次頁に示します。

コメント3に対する画面例



ヘルプアイコンをクリックすることで
マニュアルのtopページを別ウィンドウに表示



コメント4:

「ファイル操作」で日本語のディレクトリ名が作成できてしまうのは、よろしくない。(名古屋大学様)

対応策:

標準提供機能では、日本語のディレクトリ、ならびに、日本語のファイルも扱えるようになっており、日本語を抑止するような制限はしていません。
導入カストマイズで日本語を抑止する機能を提供していきます。

コメント5:

自分の好きなエディタを使うことができれば便利。
例：Windows版emacs（名古屋大学様）

対応策:

HPC PortalはWebインタフェースを使ったサービスです。好みのエディタをWebインタフェースで実現することは困難です。
HPC Portalのファイルのupload/download機能を使いファイルをPC上に転送し、好みのエディタを使用する方法をご検討願います。

FUJITSU THE POSSIBILITIES ARE INFINITE
A Portal Solution for UNIX Servers and Cluster Systems

FUJITSU

All Right Reserved, Copyright © FUJITSU LIMITED 2006

FUJITSU THE POSSIBILITIES ARE INFINITE

http://www.fujitsu.com/

添付資料41-1
R&DサーバWWSite構築サービス
商品体系

富士通株式会社

Portalを構築するサービス及び部品

- R&DサーバWWSite構築サービス *(PROPOSE)*
  - 前提ソフトウェア(PDS)のインストール整備
  - 運用環境への適用カストマイズ
  - アプリケーション固有インターフェースのカストマイズ構築
  - 可視化インターフェースのカストマイズ構築
  - Portalのカストマイズ構築
  - Globus連携も可能
- 構築支援ツール
  - 計算サーバ連携システム(HPC Portal)
  - AVS/Express連携システム(WebExp)
  - 認証・セッション管理システム(CHASEM)
  - Portal構築ツール (POESY)

サービス商品体系

R&DサーバWWSite構築サービスの構成

サービス商品	型 番	サービス内容
R&DサーバWWSite分析サービス	SV200600	サービス対象システムの運用条件、利用形態、インターフェースのカストマイズについてお客様へ のヒアリングを行い、システムの分析と最適な利用環境の設計を行います。
R&DサーバWWSite導入サービス	SV200601	R&DサーバWWSite分析サービスの結果を元に、前提となるソフトウェアのインストール、運用設定、お客様固有インターフェースのカストマイズ構築を行います。
R&DサーバWWSite可視化サービス	SV200604	AVS/Express Developperをベースにして、お客様の可視化要件に応じたWeb可視化環境をカストマイズ構築します。
R&DサーバWWSite運用支援オプション	SV200602	上記サービスで構築したシステムのQ/A対応、トラブル対応、利用ログの集計をリモートにて行います。
R&DサーバWWSite運用変更オプション	SV200603	上記サービスで構築したシステムに対し、運用条件の変更に伴う利用環境の再構築、インターフェース仕様の変更、機能の追加、新たな計算サーバ・可視化サーバの利用環境の追加構築等を行います。

・価格はいずれも個別見積りとなります。お気軽にお問い合わせください。
・本商品は、構築、カストマイズ作業を実施するサービス商品です。Portalの部品はこれらのサービス商品の構築支援ツールとして添付されています。

対象システム

- 計算サーバ
  VPP5000, PRIMEPOWER, Sun及びSun互換機, Linux クラスタ
  PRIMEQUEST
- 可視化サーバ
  PRIMEPOWER, Sun及びSun互換機, SGI(IRIX)
  (AVS/Express Developperを有するシステム)

お問い合わせ先

富士通株式会社
科学ソリューション統括部
担当: 万谷, 鈴木
poesy@ssd.ssg.fujitsu.com

4. 推進委員の所感 ～WG 活動を振り返って～

Sorry…

（SS 研会員限定情報につき、掲載できません）

＊SS 研会員の方は、認証付の版をご参照ください＊

Sorry…

（SS 研会員限定情報につき、掲載できません）

＊SS 研会員の方は、認証付の版をご参照ください＊

Sorry…

（SS 研会員限定情報につき、掲載できません）

＊SS 研会員の方は、認証付の版をご参照ください＊

Sorry…

（SS 研会員限定情報につき、掲載できません）

＊SS 研会員の方は、認証付の版をご参照ください＊

Sorry…

（SS 研会員限定情報につき、掲載できません）

＊SS 研会員の方は、認証付の版をご参照ください＊

5. まとめ
～大規模 SMP 運用 WG の活動について～

大規模 SMP 運用 WG は、PRIMEPOWER　HPC2500 を導入している会員間で、利用者へのサービスを提供する際の問題点を挙げ、ベンダーとともに解決策を考えることを 2 年間（2004 年 10 月～2006 年 10 月）行なってきた。

いずれの会員でも、ベクトル並列機（VPP）から SMP クラスタ（HPC）のスパコンに置き換えられ、計算能力（処理能力）が飛躍的に大きくなった。このため、ジョブ実行待ち時間が劇的に短縮され、即ち、殆ど待たされることなく実行され、ジョブスケジューリングに関連する問題がいくつか顕在化していないという面も否めないと思われる。また、利用者が分かり易い利用環境を、特に、初心者に優しいインターフェイスを検討する必要があるが、ここでは、十分に検討する時間がなく、今後の問題となるであろう。

次に、この WG で検討された問題点を大きく分類して、検討した内容の要旨および今後の検討に委ねられた事項について要約する。

（1）リソース使用量の指定とアカウントデータ

センターの運用者（OS も含めて）は、利用者が使用しようとする数多くのリソース（特に、CPU、メモリ）の量を把握し、最適のジョブスケジューリングを行なう必要がある。利用者はどのように必要なリソース量を理解し、指定するかが問題である。

・CPU 使用時間
・CPU の割当てとジョブスケジューリング
・アカウントとしての CPU 時間
・主記憶の割当てとページサイズ
・実行中ジョブの制限変更

【留意点】センター運用ポリシーの多様化への対応　→　機能の on/off　switch

（2）OS およびツールの改善

ジョブの依頼に関して、VPP までは、MSP からのジョブ入出力の機能などを活用して初心者向きの機能が提供されていたが、HPC では、ジョブ依頼のスクリプトを利用者自ら作成するという計算機の初期の段階に戻っている。これに対しては、ポータルを作成している会員もあるが、標準的なポータルが望まれる。

また、OS に求められる機能を、ポータルによって実現するほうが、融通性が大きくなる傾向がある。

・NQS ジョブ依頼
・チューニング支援

（3）ファイルシステム

SRFS と UFS（万能ファイルシステム？）

（4）総合可用性

・故障を前提とした可用性

（文責：金澤　正憲（WG まとめ役））