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.

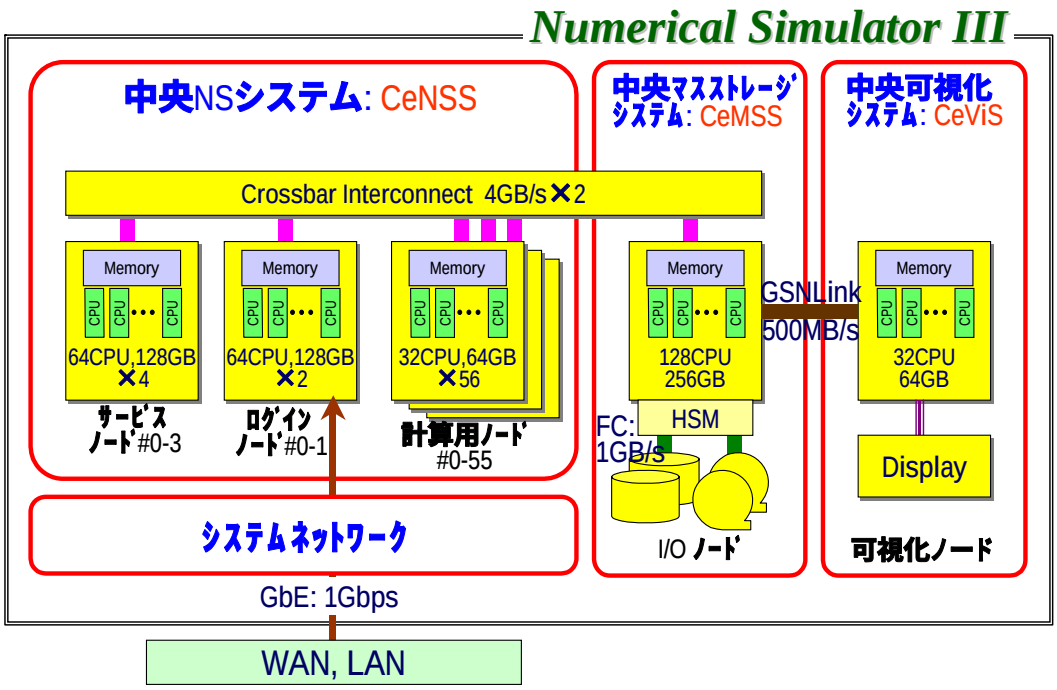

図 2.1　CeNSS の論理構成（2002 年 10 月）

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.）

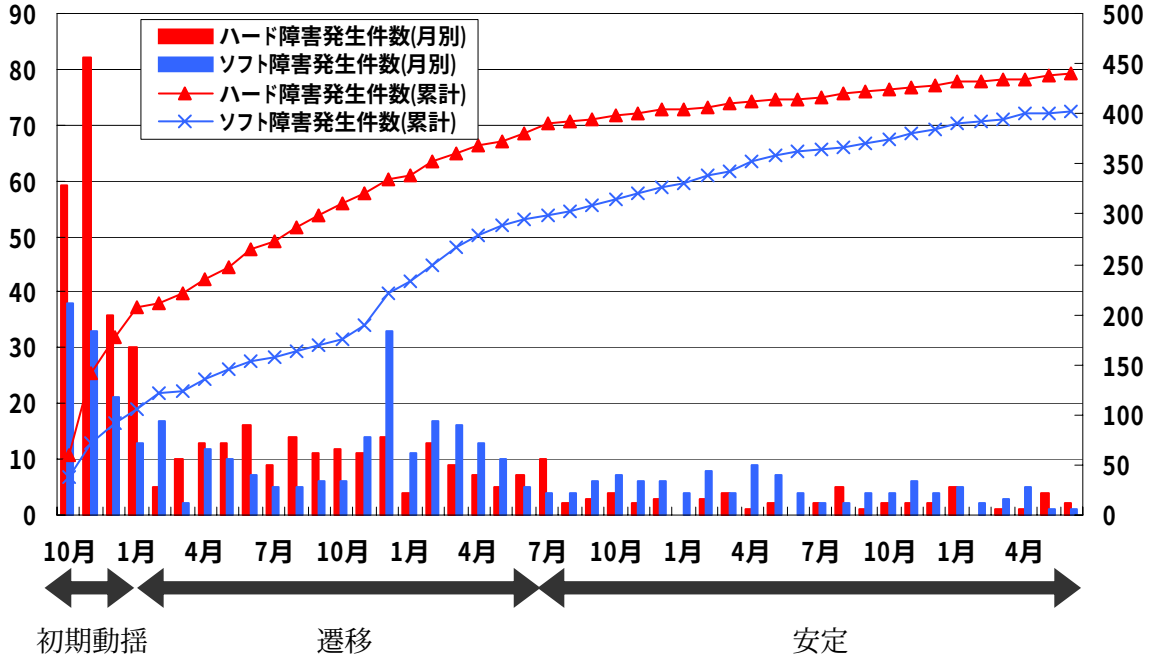

図 3.1　ハード/ソフト障害の推移

(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．

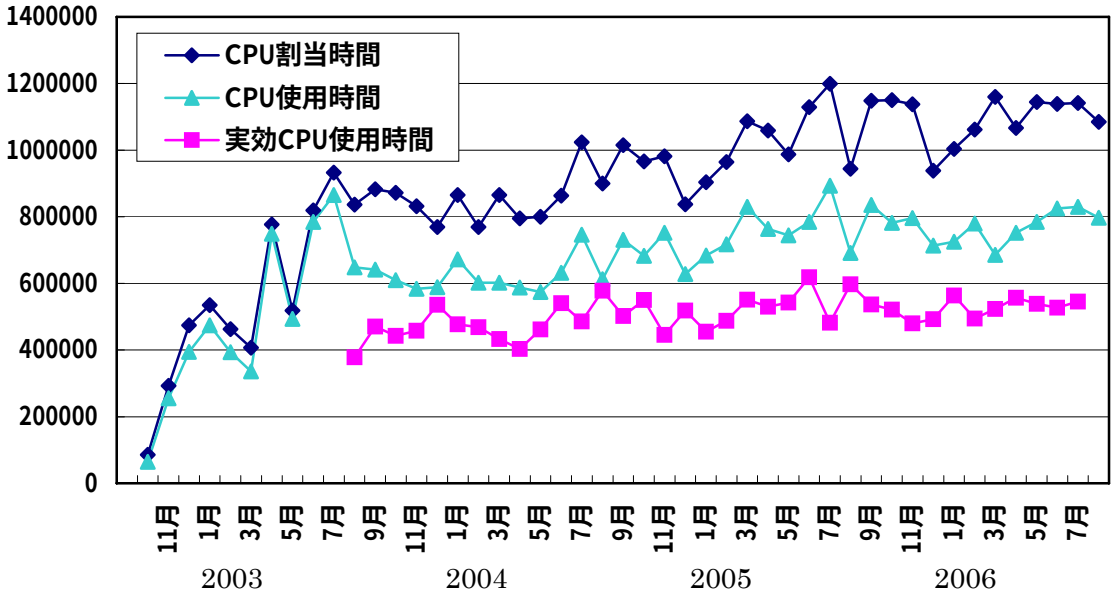

図 3.3　CPU 時間の推移

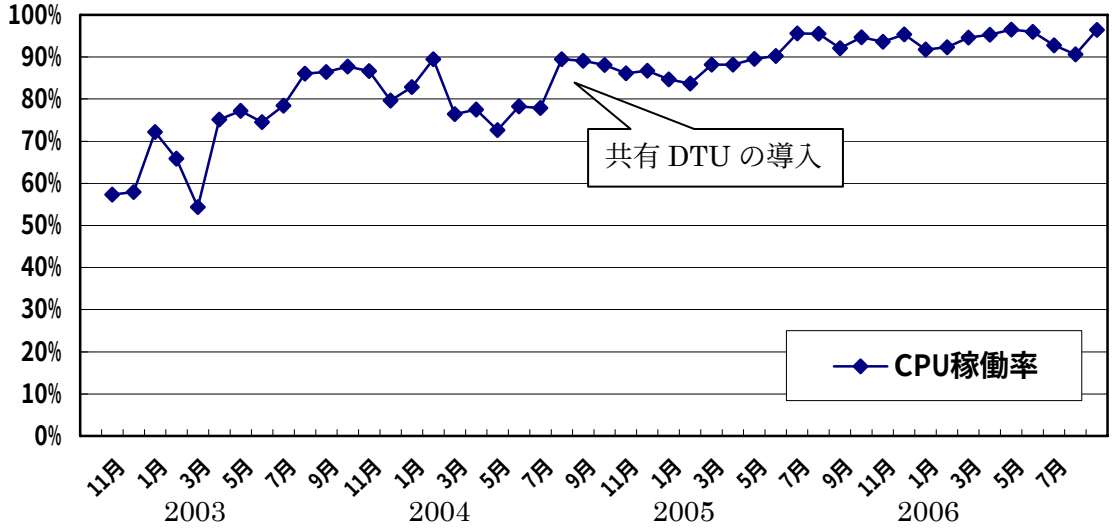

図 3.4　CPU 稼働率の推移

(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．

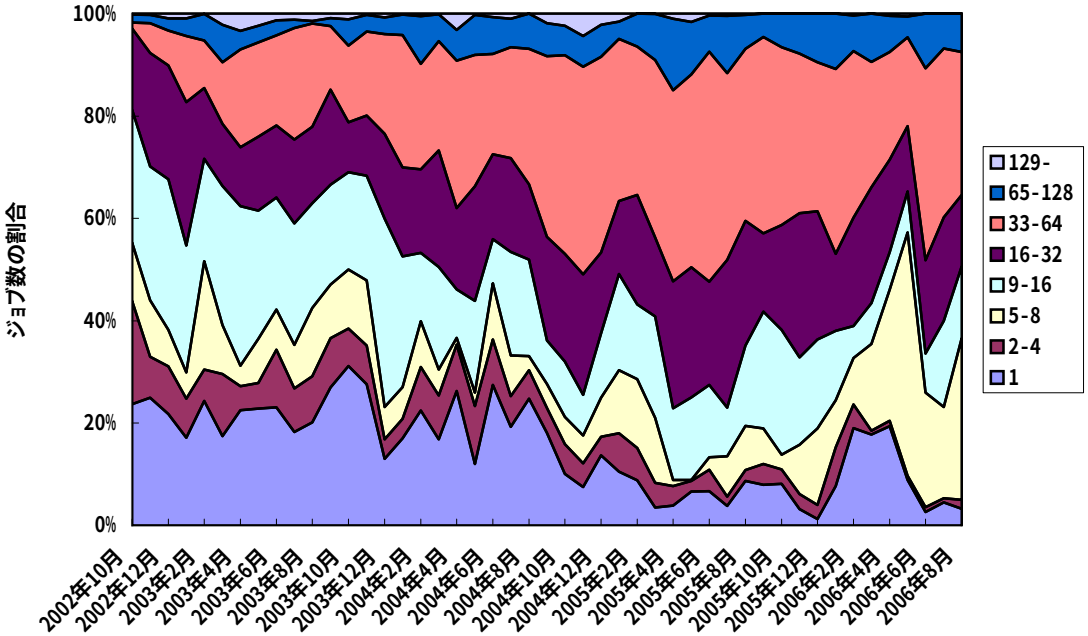

図 3.5(a) プロセス数の推移（ジョブ数）

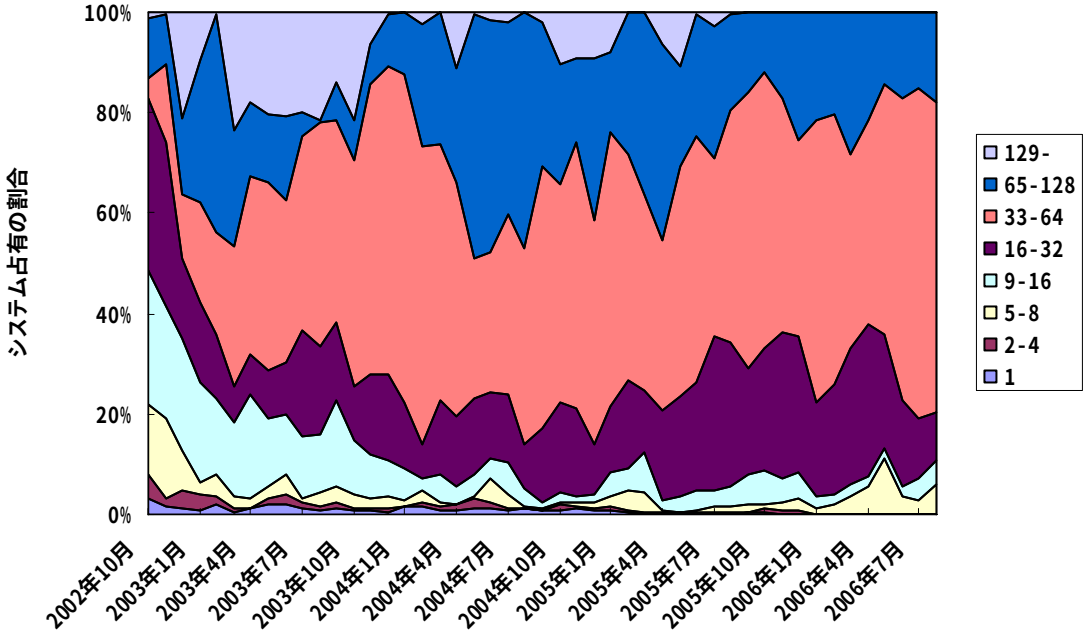

図 3.5(b)　プロセス数の推移（プロセス数×スレッド数×経過時間）

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．

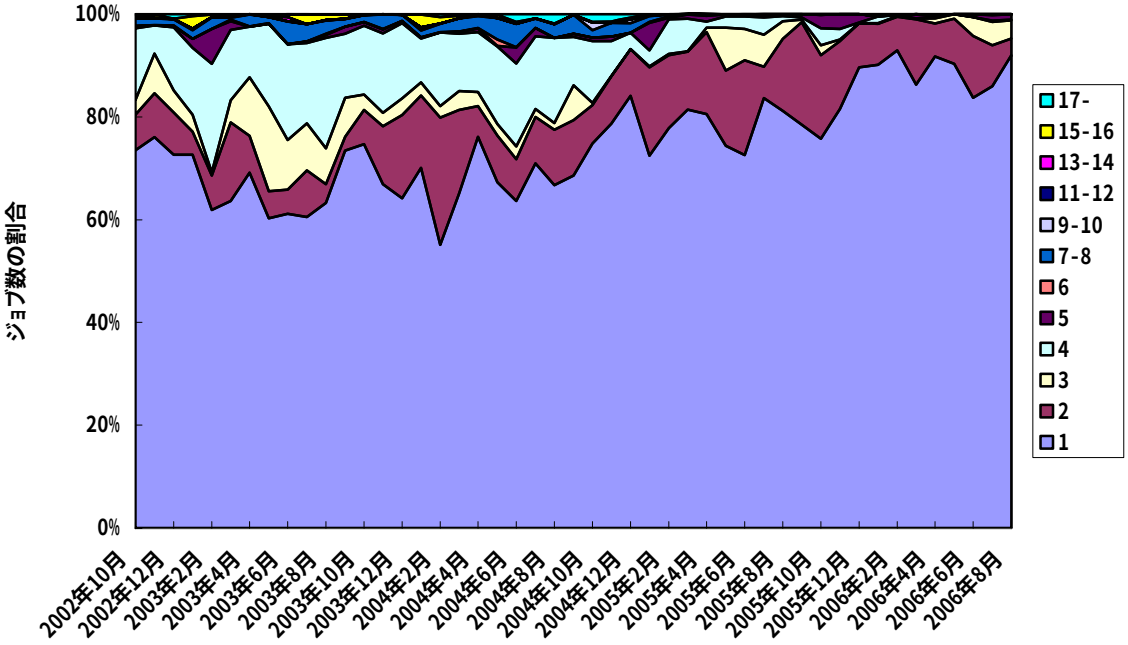

図 3.6(a)　スレッド数の推移（ジョブ数）

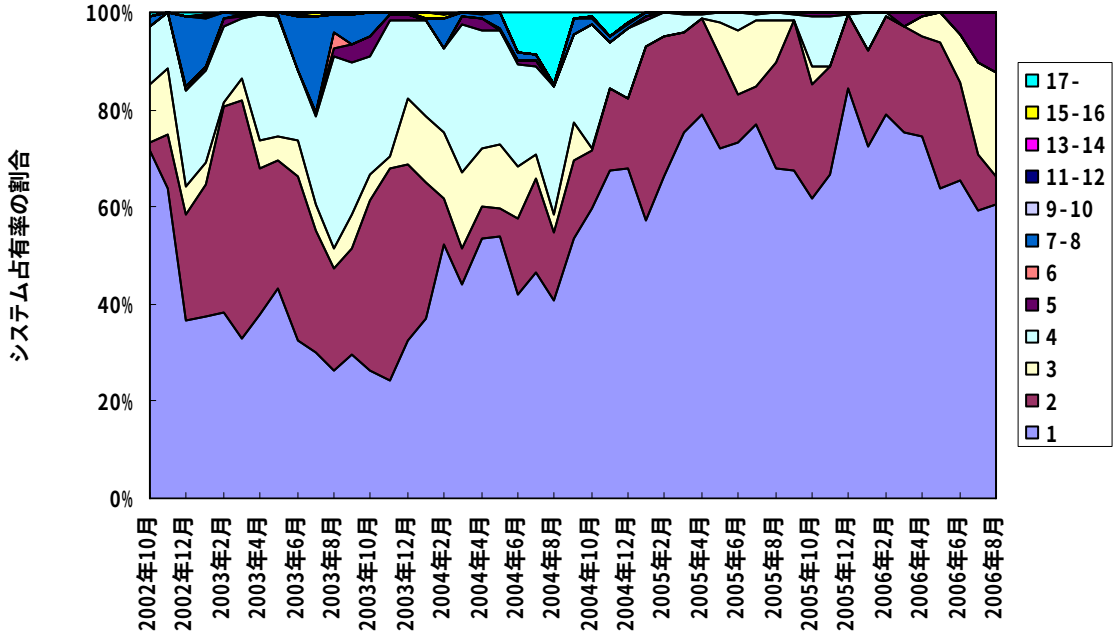

図 3.6(b)　スレッド数の推移（プロセス数×スレッド数×経過時間）

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.

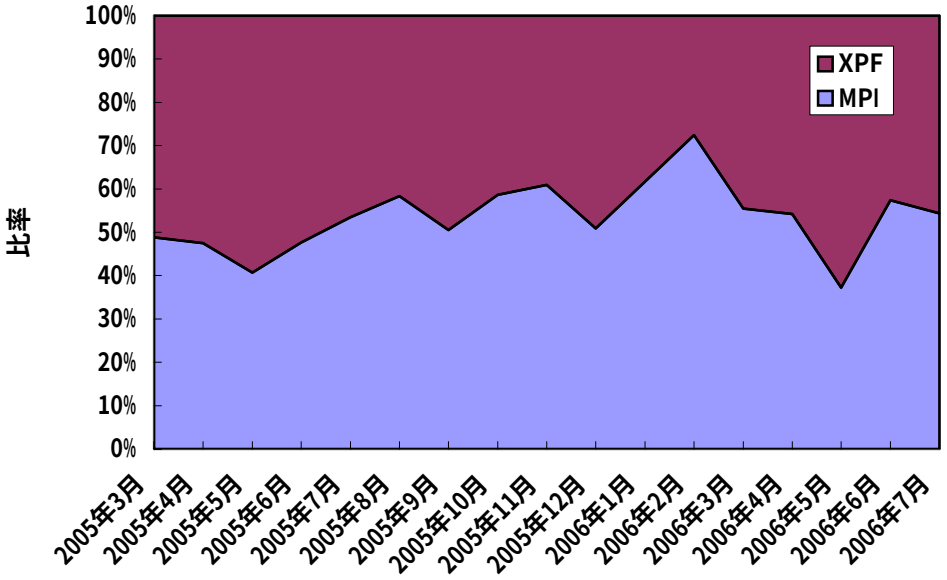

図 3.7　XPFortran 対 MPI の利用割合の推移（ジョブ数）

(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．

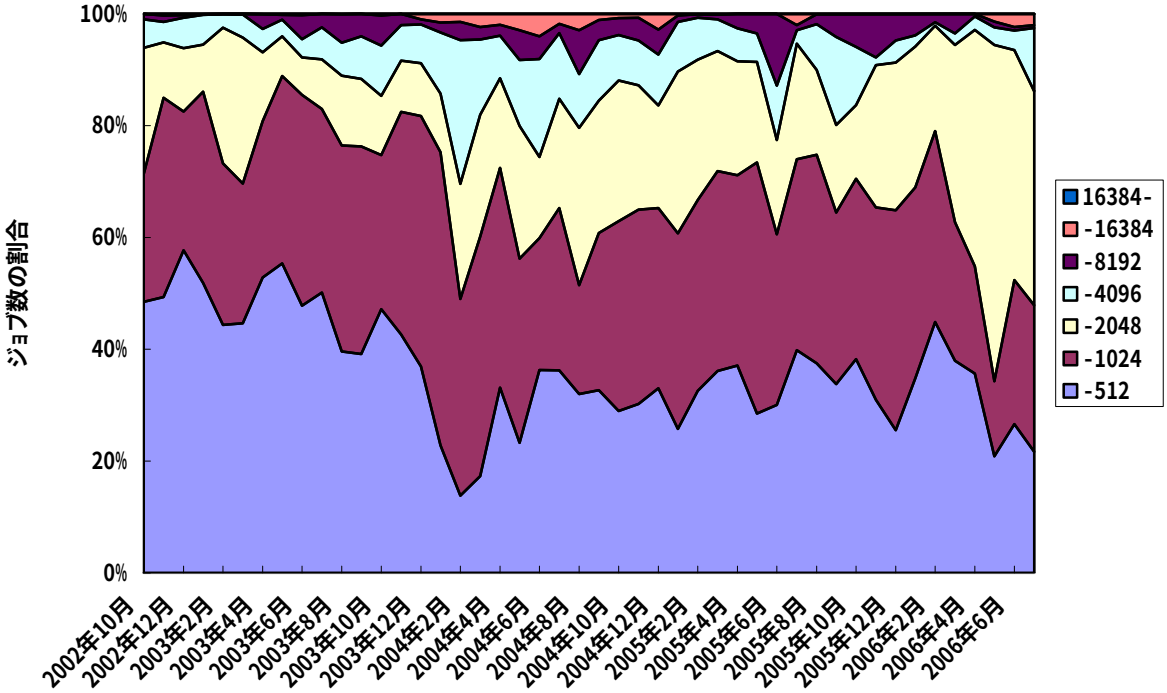

図 3.8(a)　ラージページ使用量（ジョブ数）

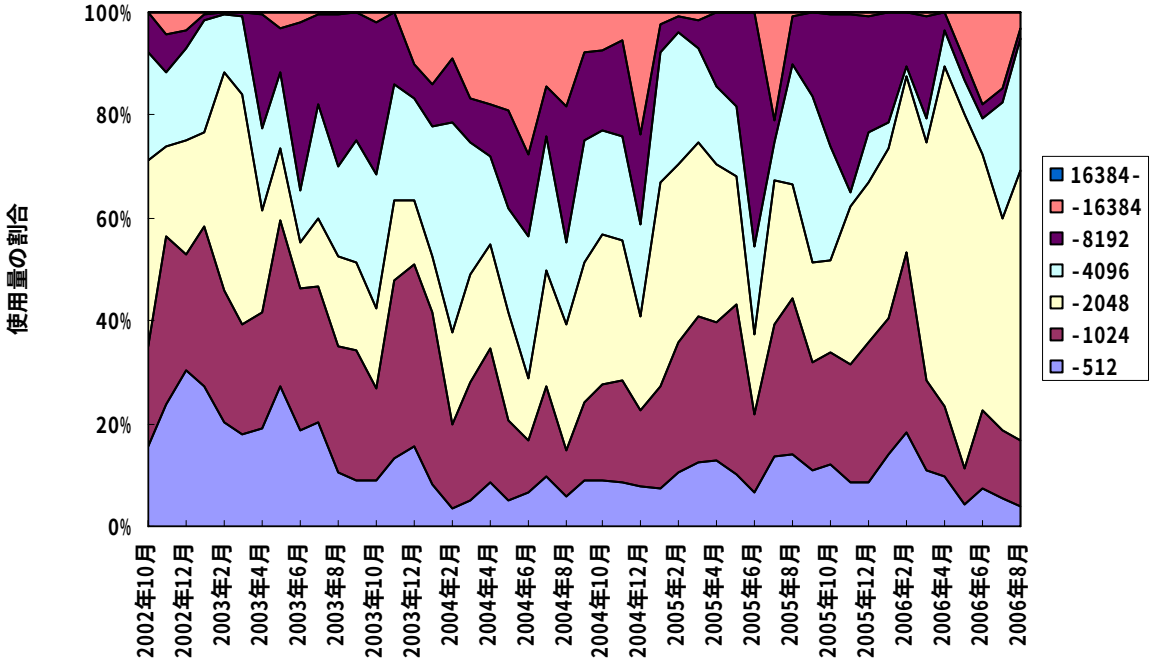

図 3.8(b)　ラージページ使用量（積算における割合）

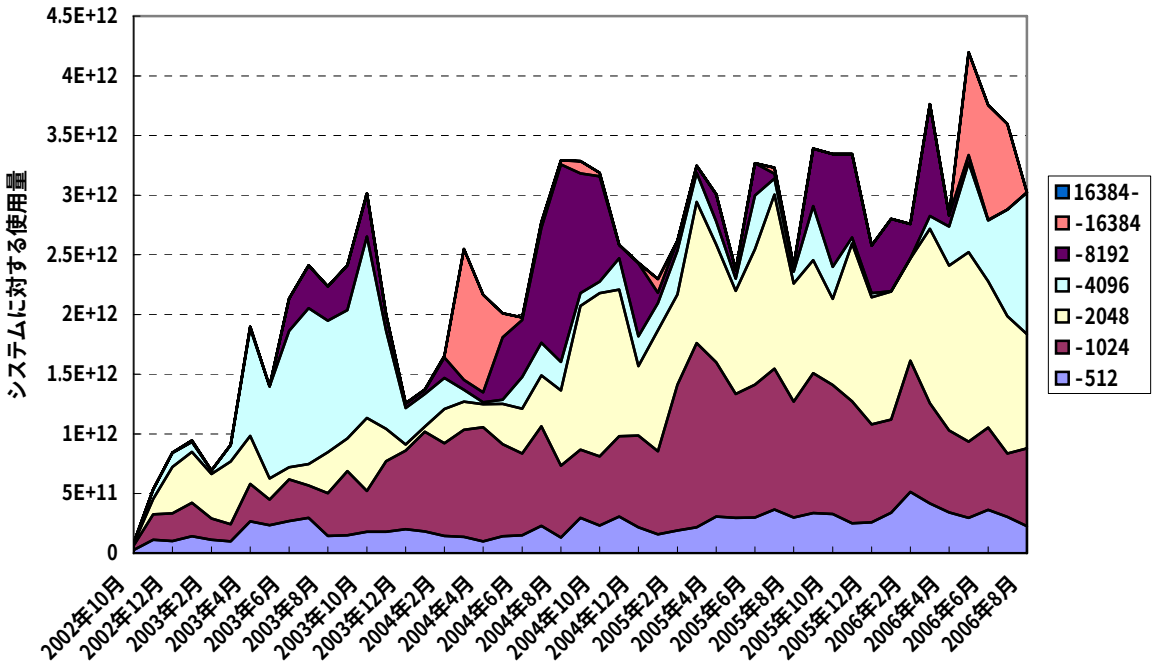

図 3.9　ラージページ使用量（積算）

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．

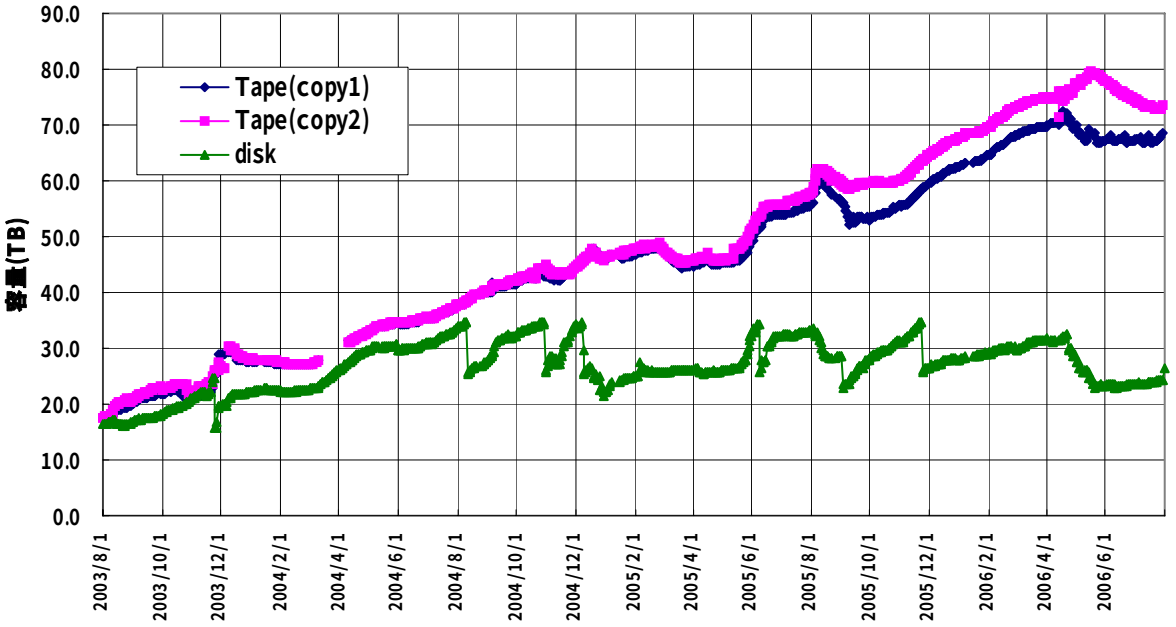

図 3.10　ストレージの使用量の推移

§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．

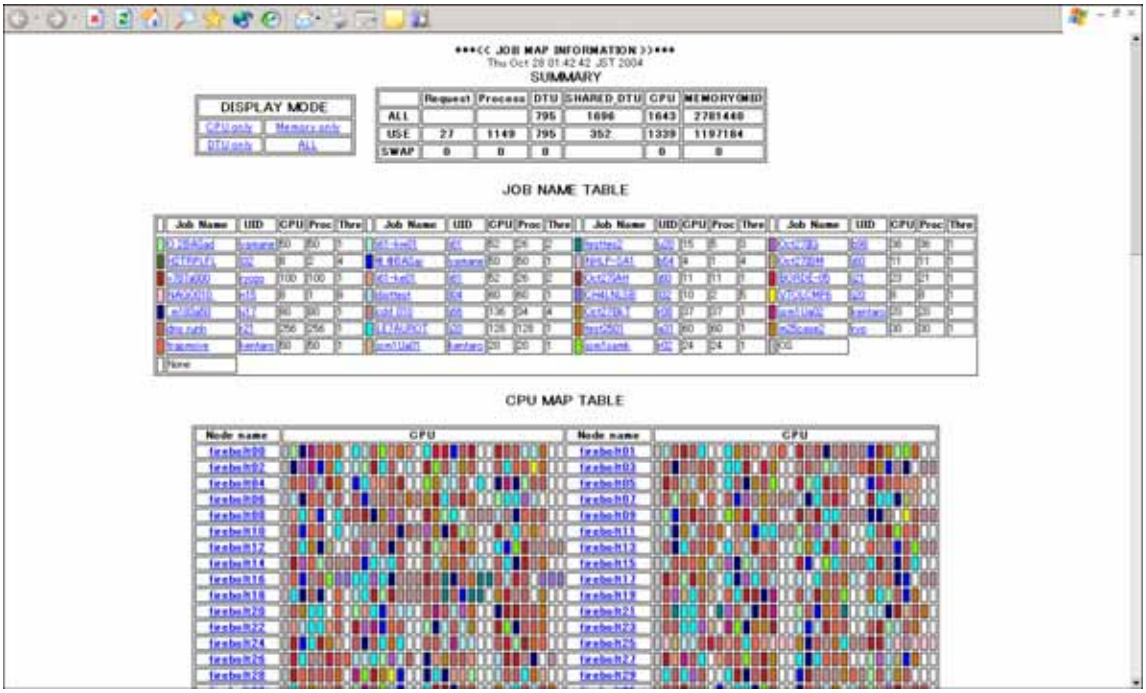

図 3.12　UNPACK 運用におけるプロセスの割付

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．

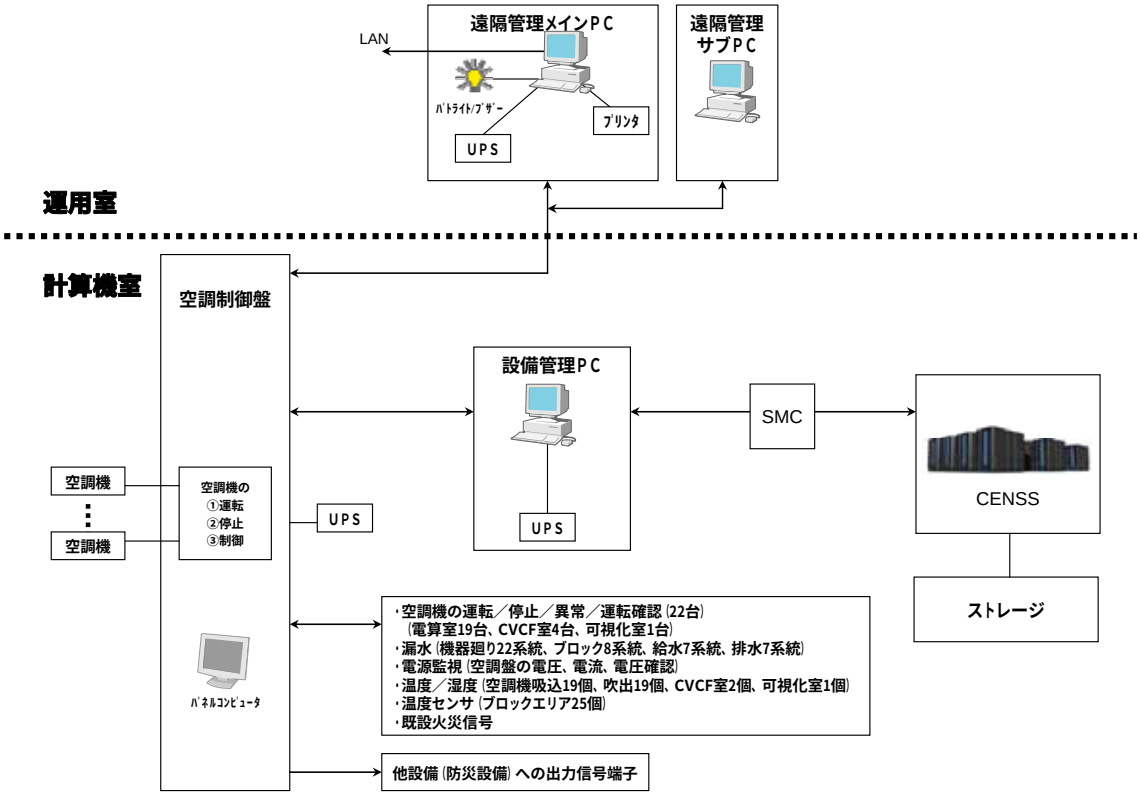

図 3.13　設備自動運転システム

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．

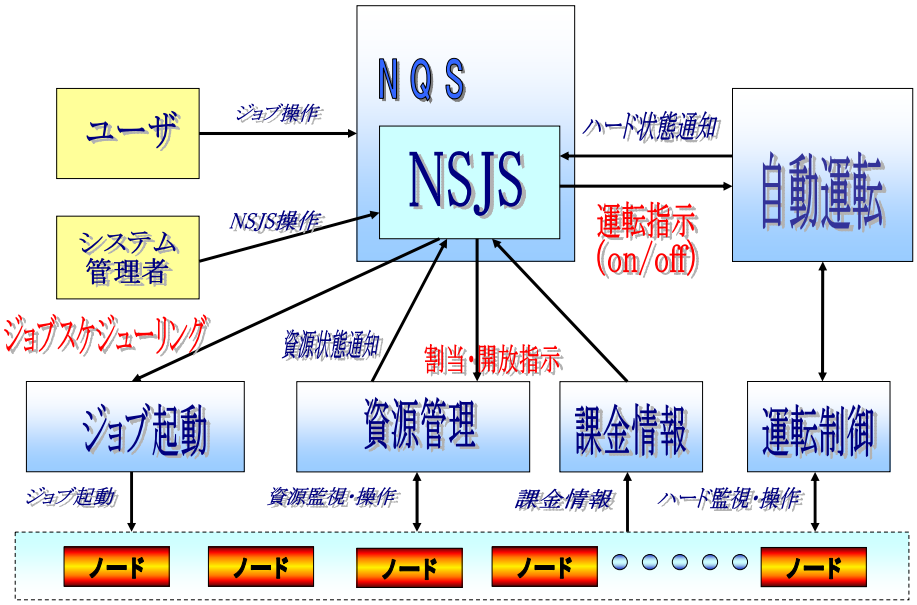

図 4.1　NSJS の位置づけ概念

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．

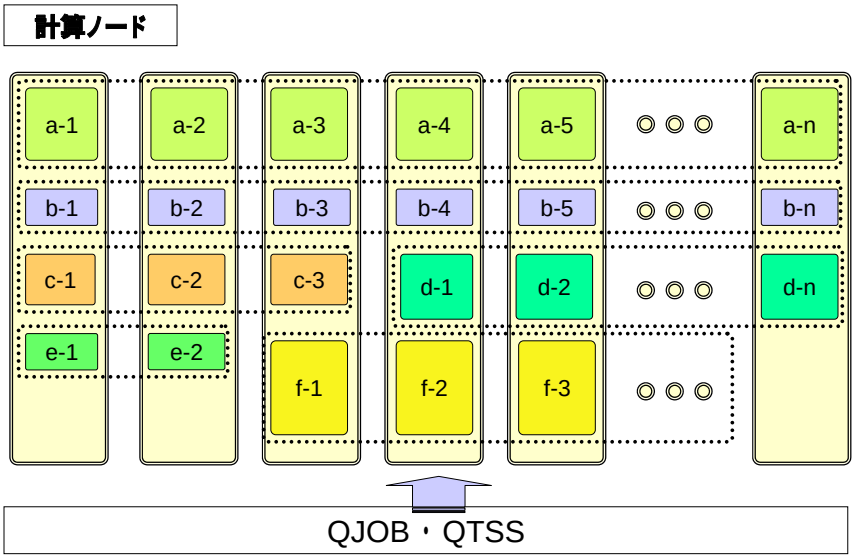

図 4.2　CPU 資源割当方式

(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．

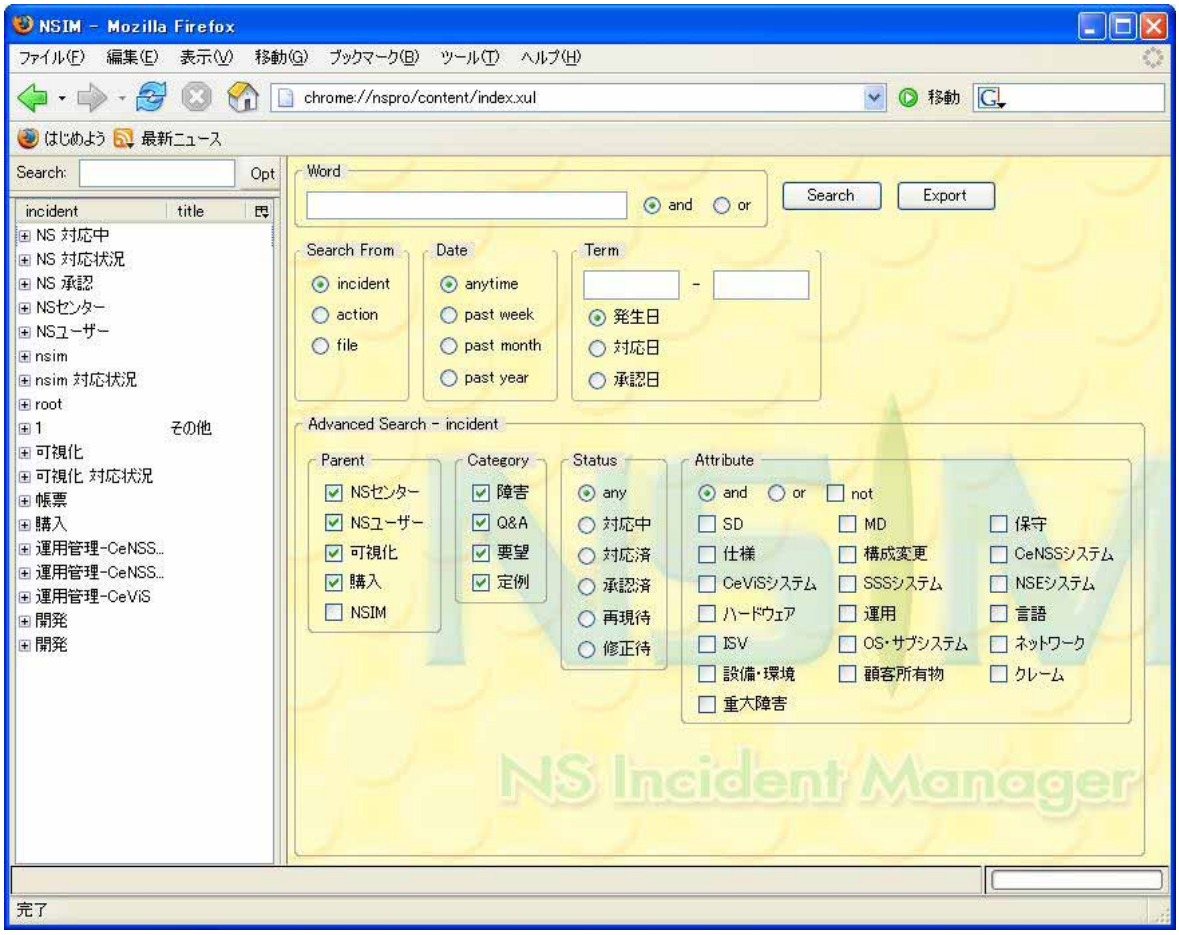

図 4.3　NS インシデント管理（NSIM）

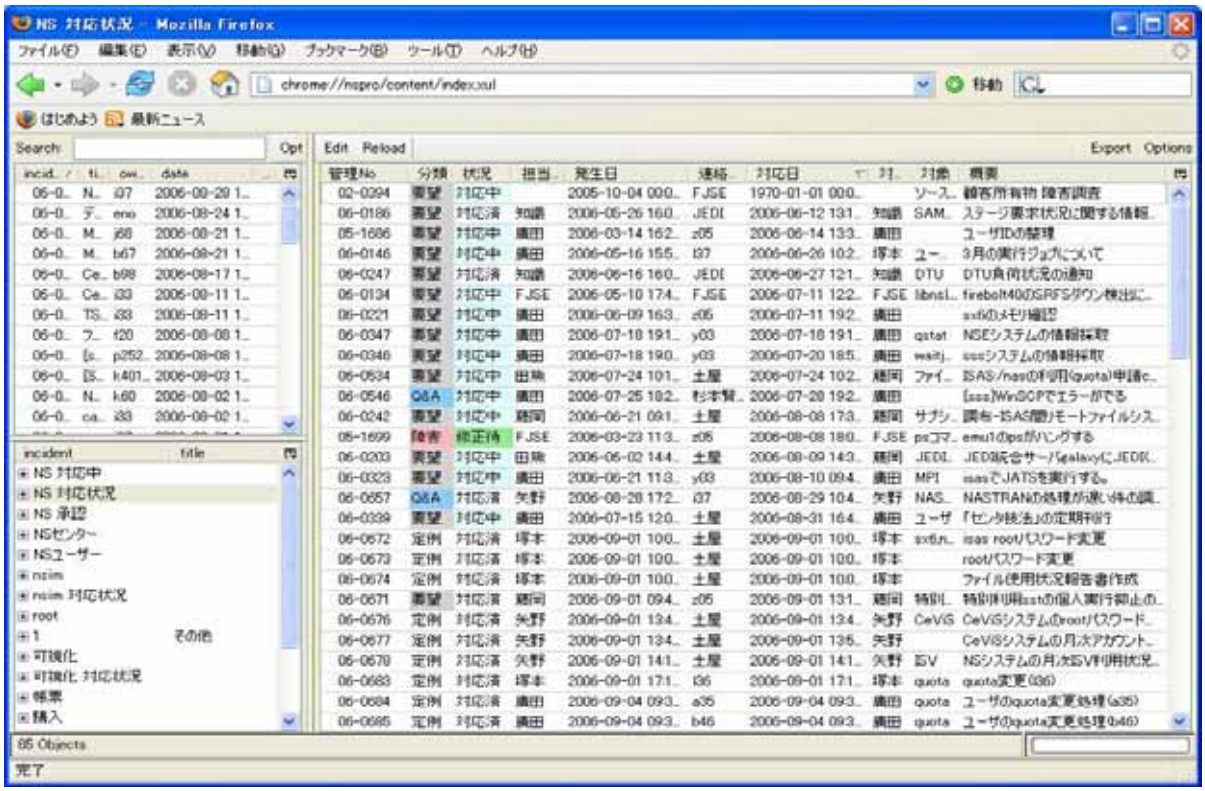

図 4.4　　NSIM 検索画面

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．

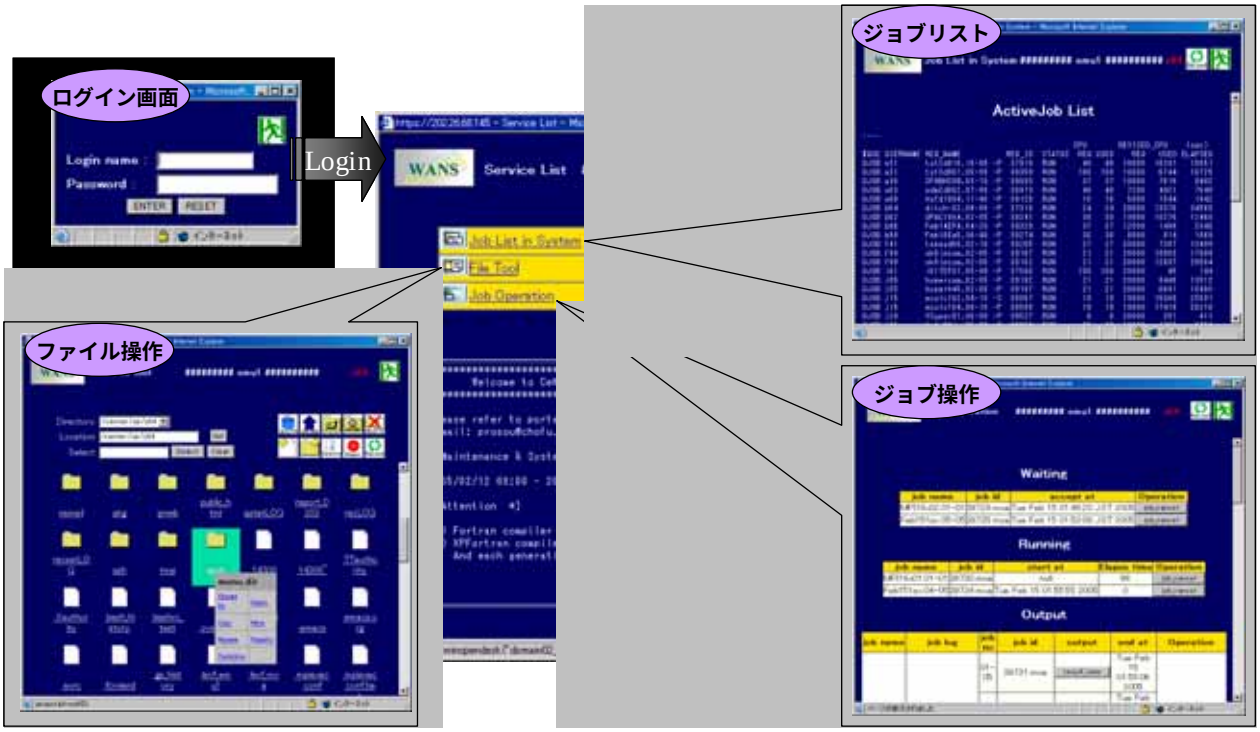

図 4.5　WANS のメニュー画面

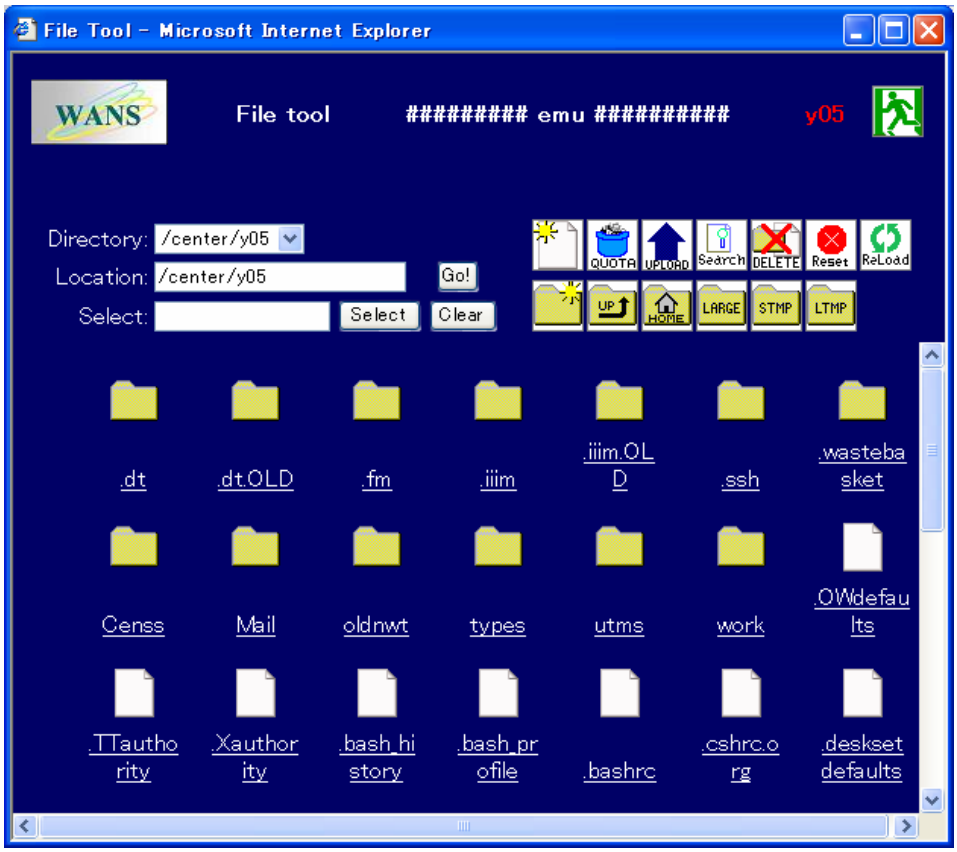

図 4.6　ファイル操作画面

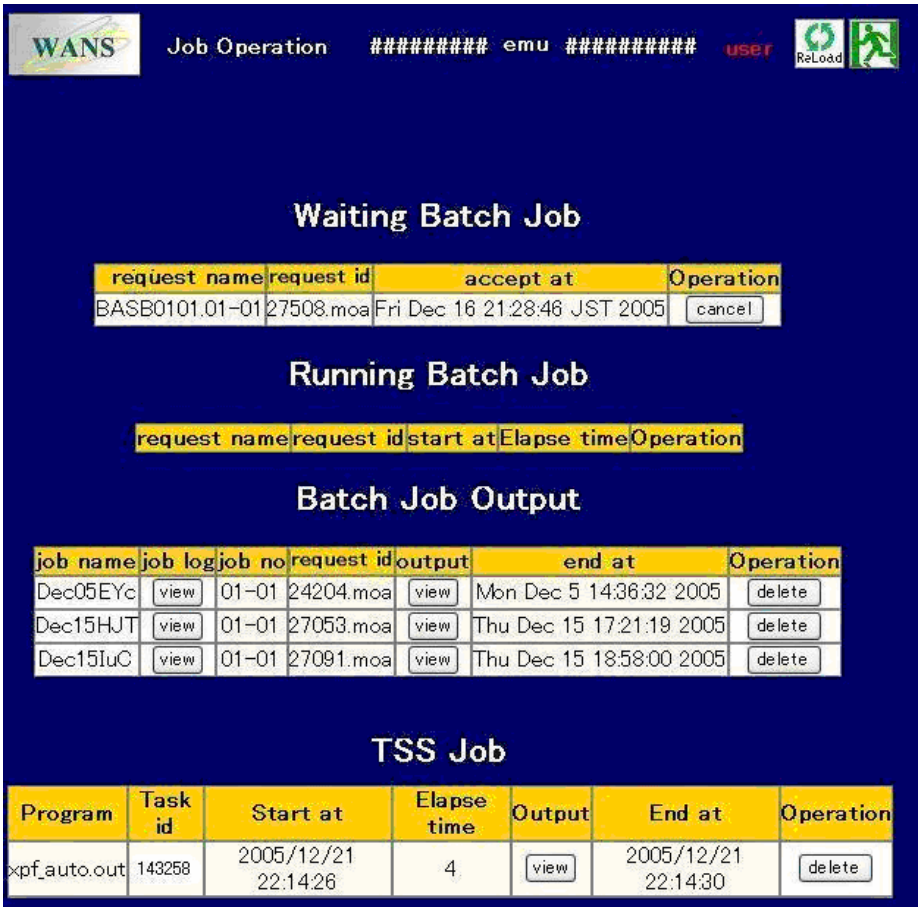

図 4.7　ジョブ操作画面

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．

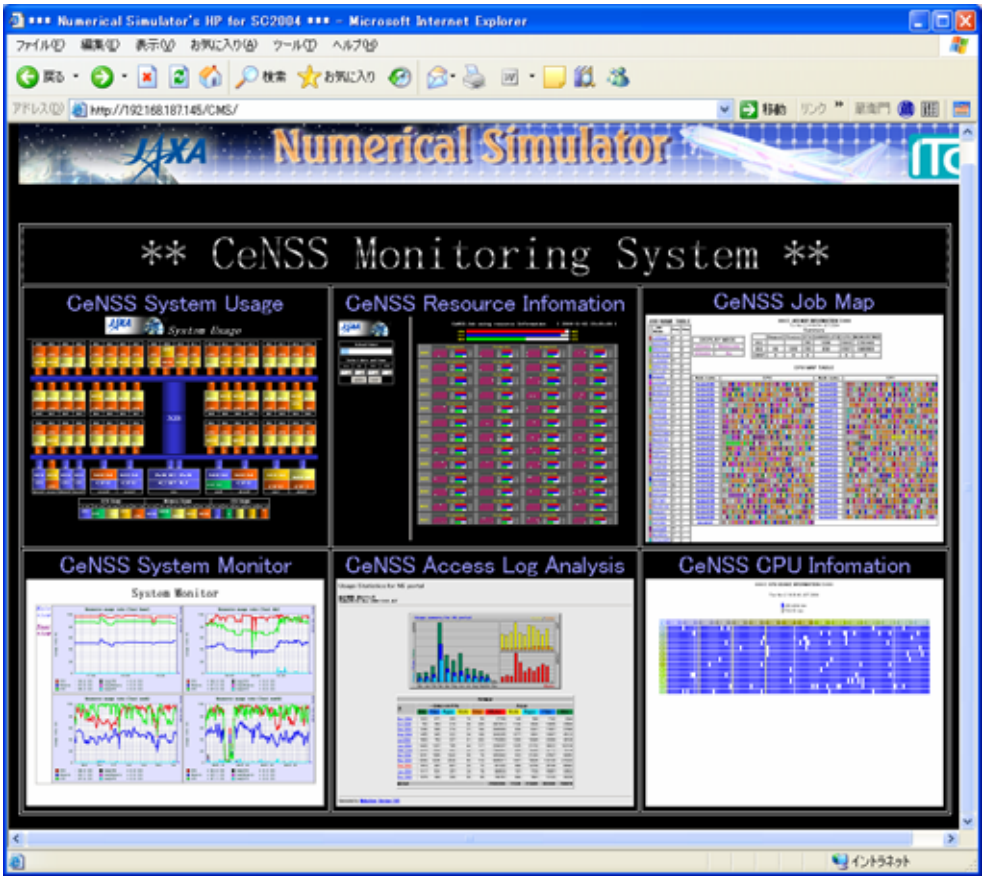

図 4.8　CeNSS Monitoring System のメニュー

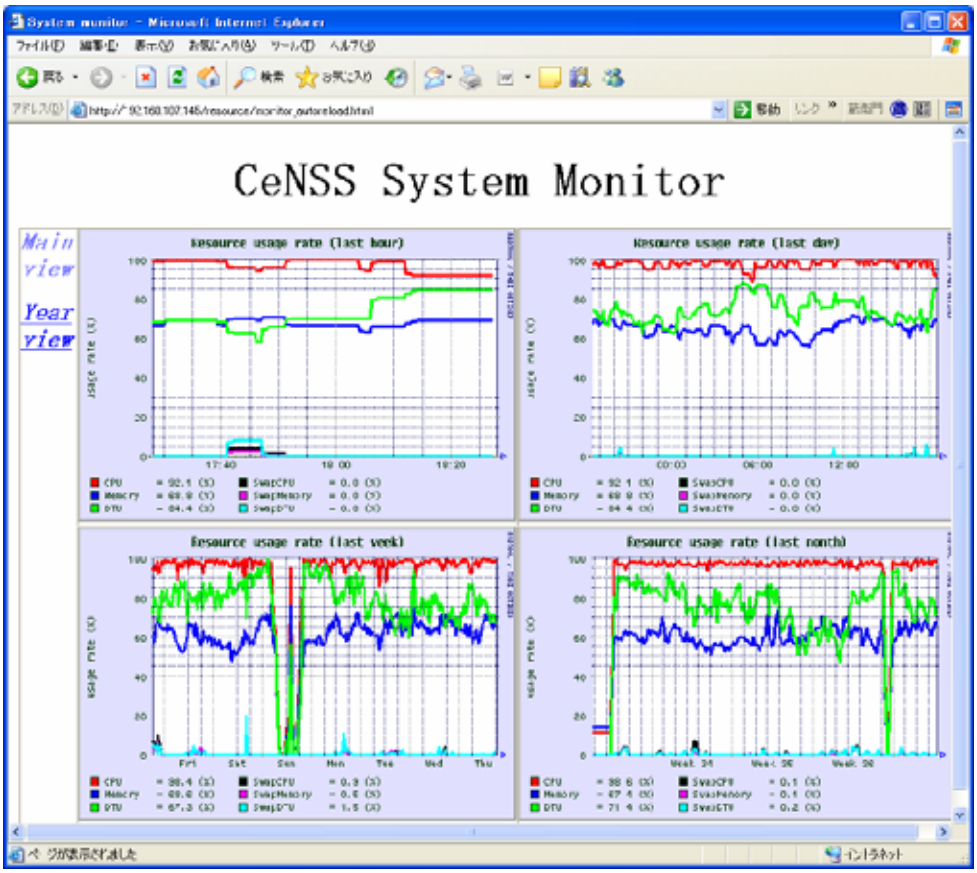

図 4.9　CeNSS System Monitor

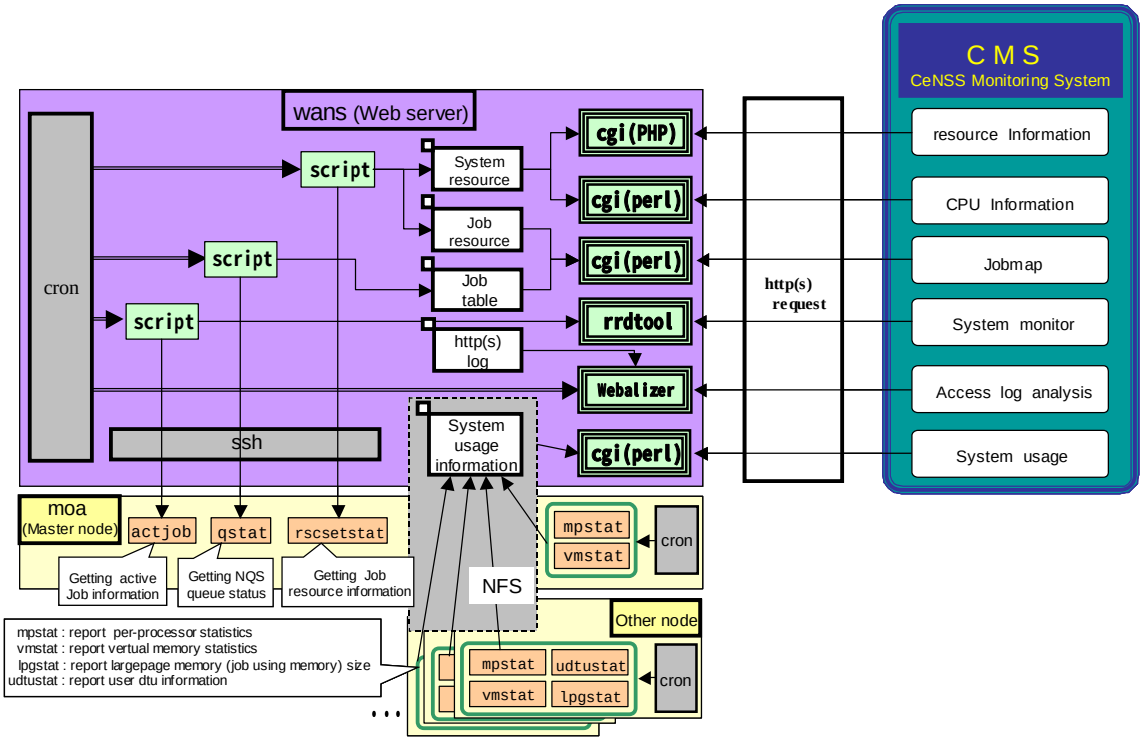

図 4.10　CeNSS System Monitor の処理の流れ

(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