2007－EVA－22 (1)
2007／8／2

バッチ処理型プログラム実行環境における資源制約を考慮したキュー選択の評価

川並秀観[†] 義久智樹[††] 金澤正憲[††]

近年, 多くの企業や研究機関において, 高速かつ高精度な計算環境を提供できるスーパーコンピューターが広く利用されている. スーパーコンピューターでは, 多くの場合, バッチ処理方式によってデータ処理を行っている. ユーザは一連の実行プログラムをジョブとしてスーパーコンピュータのキューに投入し, スーパーコンピュータのバッチ処理システムがポリシーに基づいてジョブをスケジューリングすることで, 計算資源を有効に利用している. しかし, 従来のスーパーコンピュータのバッチ処理環境では, キューごとに資源が制限されているため, 複数のユーザが同時間帯に同じキューへジョブを投入すると, 計算時間の増加といった問題が生じる. 本研究では, システム側でジョブのキュー選択を自動最適化し, 上に挙げた問題を解消することで, システムスループットの増加を目的としている. 本稿では, 実在の環境を基にした数学モデルを用い, モンテカルロシミュレーションによって自動キュー選択手法の性能を評価した.

An Evaluation for Queue Selection Policy Considering Resource Constraint on Batch Mode Processing Environment

Hozumi Kawanami,[†] Tomoki Yoshihisa[††] and Masanori Kanazawa[††]

Recently, computer simulations and numerical solutions are important for various research fields. Accordingly, many companies and research institutions use super computers, that have high performances. Super computers generally adopt batch-process policy and the users submit a sequence of programs as a job to one of queue on the super computer. By scheduling jobs according to the system policy, the super computer realizes effective performances. However, when users submit jobs to a same queue, the system performances dramatically decreases. In this research, by selecting queues automatically so as to get better performances, we improve them of the super computer. In this paper, we use numerical models of the system and evaluate the performance using Monte-Carlo simulations.

1. はじめに

近年, 様々な分野において, 大規模な数値計算やシミュレーションが実行されるため, 高速な処理能力を持つ計算機が必要とされている. このため, 多くの研究機関や大学, 企業でスーパーコンピュータが導入されているが,[1)3)4)5)6)]. 現在, このような大型計算機システムにはHPC(High Performance Computing) と呼ばれる概念が広く取り入れられており, 複数のコンピュータを相互結合網によって結合したクラスターと呼ばれるシステムが主流となっている[10)11)].

† 京都大学大学院 情報学研究科
Graduate School of Informatics, Kyoto University
†† 京都大学 学術情報メディアセンター
Academic Center for Computing and Media Studies, Kyoto University

大規模な問題を扱うスーパーコンピュータでは, 多くの計算資源を必要とするジョブが同時に複数投入されるような状況が起こりえる. このとき, 投入されたそれらのジョブを, リアルタイム処理のような対話的手法によって同時に処理すると, 以下のような問題が生じる.

- プロセス並列数が大きいジョブによってCPUが長時間占有される. 当該ジョブによってCPUが占有されている間, 他のジョブは実行されず, 実行再開まで長時間待つことになる.
- 物理メモリを大量に使用するジョブによってスワッピングが発生する. 他のジョブが使用できるメモリ量が少なくなり, 更にスワッピングが発生し, 実行速度が遅くなる.

こうした問題を解消し, 限られた計算資源を効率的に利用するための手法が必要となる. その例として,

NQS(Network Queuing System) 等の Batch Queuing System(BQS) と呼ばれる手法が広く知られている[9)12)]. BQS はリソースやジョブの管理を行うシステムであり, 限られた計算資源をユーザ間で公平に利用するため, バッチ処理方式を用いてプロセッサにデータ処理を実行させる. すなわち, ユーザは実行プログラムとなるジョブをスーパーコンピュータに投入し, スーパーコンピュータがジョブをバッチスケジューリングして実行する. 2 章で詳述するが, 通常, バッチ処理型プログラム実行環境には複数のキューが存在し, ユーザはジョブを投入するキューをその中から選択しなければならない. それぞれのキューには「CPU の同時並列数」や「CPU 時間」, 「同時実行可能ジョブ数」に異なる制限値が与えられている. このため, 複数のユーザが同時間帯で偶発的に同じキューを選択すると, 他のキューに空きがあるにも拘らず, 実行開始を待たされるジョブが発生する可能性がある.

本研究では, 実在のスーパーコンピュータ環境をモデル化し, モンテカルロシミュレーションによって自動キュー選択手法の性能評価を行った. ユーザによるキューの選択をシステム側で自動最適化することによって, システム利用の簡便性向上とシステムスループットの増加を実現できる手法の提案が研究の目的である.

筆者らはこれまで, バッチ処理型プログラム実行環境におけるキュー選択手法を評価してきた[7)]. 本稿は, キューの資源制約を考慮している点が異なる.

以下では, 2 章でバッチ処理型プログラム実行環境について簡単に説明し, 3 章で性能評価を行った最適化手法を説明する. 4 章では実験で用いたモデルとパラメータなどの設定を解説し, 5 章で実験結果を示した後, 6 章で考察を行い, 7 章でまとめる.

2. バッチ処理型プログラム実行環境

バッチ処理型プログラム実行環境では, 一定量または一定期間に集積したデータをまとめて処理するため, 特定のジョブが計算資源を占有することがない. また, キューに設定された制約にしたがって計算資源を割り当てるため, 物理メモリの枯渇によるスワッピングの発生を抑えることが可能である.

通常, バッチ処理型プログラム実行環境には複数のキューが存在しており, 最大プロセス数や同時実行可能ジョブ数, 最大 CPU 時間, 使用メモリ量といった計算資源にそれぞれ異なった利用制限が設定されている.

ユーザは利用するキューを何らかの手段でジョブ投入時に指定する必要がある. その手段として, ジョブの投入を行うコマンドのオプションでキューを指定する方法が広く用いられている. 多くの場合, 実行プログラムをジョブとして投入するにはシェルスクリプトを用いるため, キュー指定のオプションはスクリプトファイルに記述できる. 以上のように, バッチ処理型プログラム実行環境では, ユーザの意思によって利用するキューを明示的に選択することが原則である.

高度に並列化されたプログラムや, 長時間にわたって実行されることが自明であるプログラムなどは, キューに設定された条件をよく把握した上で投入するキューを判断・選択しなければならない. しかし, 大規模なジョブまたは特殊な条件を必要とするジョブの数が, 実行される全ジョブ数と比較して大幅に少ない場合, 多数のユーザが無作為にキューを選択することで, ジョブの実行リクエストが特定のキューに片寄ることが起こりえる. このとき, 他のキューでジョブの同時実行数に余裕があるにも拘らず待機状態になってしまうジョブが生じえる. これにより, 計算資源の利用効率低下, サービス時間の増加といった問題が生じると考えられる.

現在のバッチ処理型プログラム実行環境では, 人為的な計算負荷の偏りが生じてしまう場合があると考えられる. ジョブを実行するキューをスーパーコンピュータ側が自動的に選択することで, リアルタイムの稼動状況を考慮した最適な計算資源の利用とシステムスループットの向上を実現させることが目標である.

2.1 関連研究

NQS[12)](Network Queuing System) はバッチジョブのスケジューリングを行うツールとして NASA で開発された. 初期バージョンの Cosmic では複数キューの実装をサポートしており, それぞれのキューでは計算資源が制限されている. キューに投入されたジョブは FIFO(First In, First Out) の原則によって処理が行われる. 商用として開発された後期バージョンでは, 負荷分散およびより高度なリソース管理機能が追加された. シンプルで実装が容易であるため, 多くのスーパーコンピュータに実装されている. NQS は標準的なバッチ処理型プログラム実行環境を提供する標準プロトコルであるため応用が利きやすく, 様々な手法と組み合わせることができる.

また, ジョブの振り分けを行うシステムとして, Condor[12)] がある. アイドル状態のワークステーションやサーバから計算資源を収集し, 長期間稼動するバッチジョブへ割り当てることを目的として Wisconsin 大学が開発した. Condor の制御を行うデーモンは, プールされているホストの稼動情報を CPU の負荷や入出

力デバイスの状態で判断する．アイドル状態であると判断されたホストがあれば，他のコンピュータで待機状態にあるバッチジョブを移送して処理を行わせる．チェックポイント機能を実装しているため，アイドル状態のホストがビジー状態に遷移した際，他のアイドル状態であるホストへバッチジョブを転送してチェックポイントから処理を継続して行うことができる．

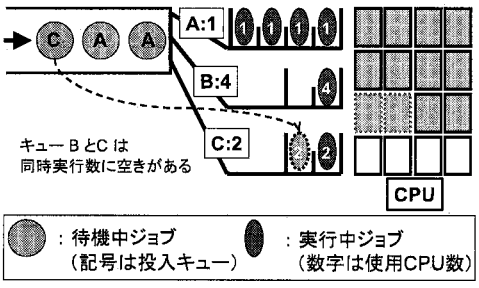

図 1　基本モデルの概要
Fig. 1　an overview of our model

3. 実験に用いた最適化手法

この章では，シミュレーションで用いた自動最適化手法について説明する．以下で説明する手法とキューを指定する通常の手法の性能をシミュレーションによって比較・検討した．

ここで通常手法の様子を図 1 に示す．

左の三つの円は待機中ジョブを示し，円中のアルファベットは，投入先となるキューを示している．キューAの最大利用可能 CPU 数はA:1と記述しているようにジョブあたり 1 個であり，キューBは 4 個，キューCは 2 個である．

キューAではすでに 4 個のジョブが実行されており，それぞれの使用 CPU 数は 1 個である．キューBでは 1 個のジョブが実行されており，使用 CPU 数は 4 個，キューCでは 2 個となる．合計で 10 個の CPU が使われることになるため，図中に示す 16 個の CPU のうち，10 個は使用中という意味で灰色を用いて示している．ここで，キューAは最大ジョブ数が 4 個であり，実行中のジョブも 4 個であるため，これ以上キューAのジョブを実行できない．このため，待機中のジョブのうち，Cが先に投入され，2 個の CPU を使って実行されることになる．

以上のように，全てのジョブはいずれかのキューを指定しており，ジョブの同時実行数に余裕がない場合には枠が空くまでそのキューで待機し続ける手法を指している．この場合，他のキューに空きがあってもジョブの移動は行われない．

本稿で検討する最適化手法では，実行可能キューのうち，ジョブの同時実行可能数の上限にもっとも余裕があるキューを選択する．これにより，特定のキューへジョブが片寄ることを回避し，無駄に待機状態となるジョブが減らせると考えられる．また，残りの実行可能数が最大である実行可能キューが複数ある場合には，CPU の制限が最も厳しいキュー，つまり，CPU の同時利用可能数が少ないキューや CPU 時間の制限が厳しいキューを選択するものとする．このポリシーによって，CPU 並列数などの資源が豊富なキューの実行可能枠を無駄に消費することが削減できると期待される．つまり，一つのジョブで 128 個まで同時に CPU を使用できるキューが 1CPU しか使用しないジョブで埋め尽くされるような場合を減らせると考える．以降では，この最適化手法を**最大実行可能残数選択法**と呼ぶ．

4. ジョブ投入システムのモデル化

4.1 HPC2500 の概説

モデルの説明に先立って，モデルの参考にしたスーパーコンピュータ，HPC2500 について簡単に紹介する．HPC2500 は京都大学大型計算機センターに設置されたクラスター型のスーパーコンピュータである．フロントエンドプロセッサが 1 ノード，バックエンドプロセッサが 11 ノードの合計 12 ノードで計算処理を行う．12 台の計算ノードの他に，ディスク装置と接続された I/O ノードが 1 台存在し，これら 13 台のノードは高速光インターコネクトによって相互に接続されている．ユーザはフロントエンドプロセッサに ssh プロトコルを利用してログインすることでシステムの資源を使用できるようになる．また，外部ネットワークとの接続を持つのはフロントエンドプロセッサのみである．

表 1 は HPC2500 に設置されたキューの名称と，それらのキューに設定された資源制約の一部である．この他の資源制約として，ラージページメモリ量，スレッド並列数，プロセス並列数などがある．

次に，ジョブ投入の詳しい手順を紹介する．

バッチジョブの投入は，多くの場合，バッチスクリプトと呼ばれる一種のシェルスクリプトを利用する．スクリプトファイルの中に，所定の形式に沿って指定するキューや利用したい計算資源の種類，数量を指定する．

表 2 はバッチスクリプトの一例であり，(1) でジョブの投入キューとして “d32” を指定している．表 1 に示

表 1　HPC2500:キュー名と制限量
Table 1　queue names and the resource constraints of HPC2500

キュー名	最大 CPU 数	経過時間
s8	8	336 時間
ss8	8	2 時間
s128	128	336 時間
d32	32	336 時間
d128	128	2 時間
d512	512	336 時間
fss8	8	2 時間
fs8	8	336 時間
fs32	32	336 時間

表 2　バッチスクリプトの例：“test.sh”
Table 2　an examle of batch script

```
#@$-q d32                              (1)
#@$-lT 200:00:00                       (2)
#@$-lP 4                               (3)
#@$-lp 8                               (4)
mpiexec -n 4 -mode limited ./test.out  (5)
```

すように d32 を指定すると，最大 32 個の CPU を利用でき，336 時間までジョブを連続で実行できる．(2), (3), (4) はそれぞれ CPU 時間，プロセス並列数 (CPU 利用数)，スレッド並列数を指定している．そして，(5) では “test.out” という名前の MPI(Message Passing Interface)[2] プログラムを (1)-(4) の条件の下でジョブとして実行するよう要求しているコマンドである．この例ではプログラムを一個しか指定していないが，複数個指定し，一連のプログラムを連続して実行できる．

表 2 のようなスクリプトを作成すると，これをジョブとして処理してもらえるよう，スーパーコンピュータに投入しなければならない．スクリプトファイルをジョブとして投入する際には，ssh クライアントなどの遠隔操作用端末を利用し，この端末上でスクリプトファイルを引数としてジョブ投入用のコマンドを実行する．例えば，HPC2500 におけるジョブの投入コマンドは “qsub” である．従って，

```
[user@hpc user]$qsub test.sh
```

の様にコマンドを実行すればよい．本研究では，キューの資源制約を越えていると，qsub で投入されたジョブが待機状態になると想定している．図 1 や図 2 の左端の矢印は，qsub によりジョブが投入されたことを示している．

表 3 は HPC2500 のを構成するコンピュータ 1 ノードあたりのスペックを示している．

4.2　HPC2500 のモデル化

モンテカルロ法によるシミュレーションを行うにあたって，プログラムの挙動を定義するため，実際のバッチ処理型プログラム実行環境である HPC2500 をモデル化した．図 1 に HPC2500 のジョブマネジメントに関する基本動作モデルを示す．

表 3　HPC2500:簡易スペック表
Table 3　a brief specification of HPC2500

		フロントエンド	バックエンド
CPU	個数	128	128
	クロック	2.08GHz	1.56GHz
	一次キャッシュ	128KB × 2	128KB × 2
	二次キャッシュ	4MB	2MB
メモリ		512GB	512GB
理論演算性能 [GFLOPS]		1064.96	798.72

ユーザはシェルスクリプトに実行プログラムの情報と使用するキューなどの実行条件を記述し，これをジョブとしてゲートウェイノード（フロントエンドプロセッサ）に投入する．投入されたジョブは，システムによって実行可能であるかを判定され，可能であれば指定されたキューで CPU を割り当てられ，処理が開始される．制約条件からジョブを実行できない場合には，計算資源が新たに解放されるまで待機状態となる．

図 1 では，CPU の総数 16 個に対して 3 種類のキュー A, B, C が用意されている．キュー A, B, C におけるジョブの同時実行可能数はそれぞれ 4, 2, 2 である．また，ジョブ一つあたりが使用可能な CPU 数の上限はそれぞれ 1, 4, 2 としてある．図 1 は，キュー A について待機中のジョブが二つあるところにキュー C へ新たなジョブが到着した状態である．すでに二つのジョブがキュー A について待機中であるが，キュー C では実行中のジョブ数が上限に達していないため，すぐに CPU が割り当てられ，処理が開始される．この場合，キュー B には資源の空きがあるが，待機中のジョブが存在している．

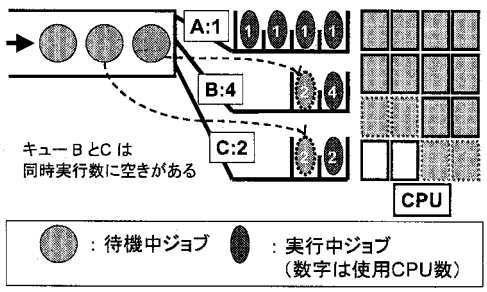

図 2　最適化後のモデル概要
Fig. 2　an overview of of our model under automatical allocation

図 2 では, ジョブは実行キューを指定されず, システムによって自動的にキューが選択される場合を示している. この場合, 空いているキューを自動的に選択しているため, 待機中のジョブは先着順に実行可能キューへ投入される. 従って, ジョブの平均待機時間が減少し, 資源の利用率が向上していると言える.

5. 評価実験

この節では, モンテカルロシミュレーションを用いて通常手法と最大実行可能残数手法の性能を比較・評価する. モンテカルロシミュレーションに用いる乱数は, メルセンヌ・ツイスター[8] を利用して生成している. 実験には図 2 のモデルを用いた. ジョブがシステムに到着する間隔として 10ms, 100ms, 1000ms の 3 通りの場合を考え, それぞれの場合におけるジョブの平均待機時間および待機状態に入る割合を調べた.

ここで, それぞれのジョブの要求する CPU 数や処理開始から完了までの処理時間といった様々なランダム値には, HPC2500 における実際のログデータから確率分布を用いて近似した値を用いている. 例えば, ジョブの処理時間は実際のログデータより最小二乗法を用いてパラメータを同定したパレート分布を用いた.

3 通りそれぞれの到着間隔において, 性質を乱数によって生成した 10000 個のジョブを一つずつ等間隔にシステムへ投入し, 投入した全てのジョブの終了までを一度のシミュレーションとした. 今回のシミュレーションでは時間による終了判定を行っておらず, ジョブが全て実行されると終了する.

手法の性能を評価する指標として, 投入したジョブが待機状態に入った割合と, 待機状態を経過したジョブの平均待機時間を用いる. 待機状態に入った割合が少ないほど, 多くのジョブが投入直後に実行されるため, この値は小さいほどシステムのスループットが高いといえる. また, 待機状態の時間が短いほど, 時間あたりに処理するジョブの数が多いことになるため, この値も小さいほど良いといえる.

図 3 に, ジョブの到着間隔とジョブが待機状態に入る割合の関係を示す. "Auto" が最大実行可能残数手法, "Normal" が通常手法を表している. 図 3 より, 到着間隔が 10ms の場合にはどちらの手法もほぼ同じ割合のジョブが待機状態を経て処理が行われていることが分かる. また, 間隔が 100ms, 1000ms の場合には, Auto の方が待機状態に入る割合が少ないということが見られる. これは, 10ms の場合は, スーパーコンピュータの容量に対して過剰な量のジョブが到着し, このため, 到着するジョブのほとんどが待機状態に入ってしまい,

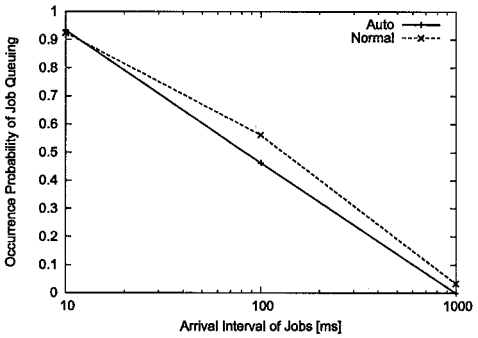

図 3 到着間隔に対してジョブが待機状態になる割合
Fig. 3 occurrence probability of job queuing and the arrival interval

100ms と 1000ms の場合にはその状態が緩和され, 最適化手法が性能を発揮できるようになったためであると考えられる.

更に, 1000ms から 10000ms, 100000ms, ... とジョブの到着間隔を増加させていくと, 最終的には到着間隔よりもジョブの処理時間の方が短くなり, ジョブが到着するたびに常に資源が空いている状態になると考えられる. このとき, ジョブは待機状態を経由せずに処理状態へと移行するため, 平均待機時間と待機状態に入るジョブの割合は, 到着間隔が増加するにつれて, 共に 0 に収束すると考えられる.

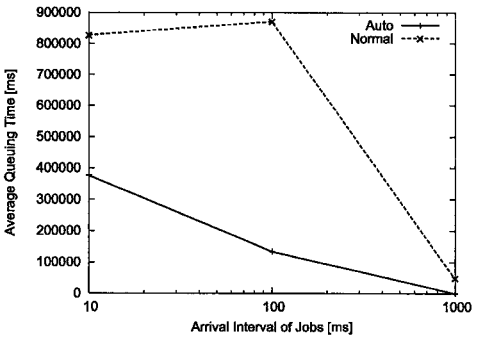

図 4 到着間隔と平均待機時間の関係
Fig. 4 average queuing time and arrival interval of jobs

図 4 に, ジョブの到着間隔と平均待機時間の関係を示す. これを見ると, 最大実行可能残数選択法の方が明らかに平均待機時間が短くなっていることが分かる. これは, 3 節で触れたように, 無駄な待機時間を過ごすジョブが十分に削減できているためであると考えられる.

また, 図 4 では, 到着間隔 10ms の場合における通常

手法の平均待機時間が, 到着間隔 100ms の場合よりも低い値を示している. これは, シミュレーションの試行回数が少ないために歪みが生じていると考えられる.

6. 考　　察

シミュレーションによる評価の結果, 今回検討した最大実行可能残数選択法は通常のキューマネジメント手法よりも平均の待機時間を抑えることができると分かった. 特に, 到着間隔が 10ms および 100ms の, システムに高い負荷が掛かっている場合には通常手法と大きな差が開いたことが図 4 より分かる.

しかし, 今回の評価方法では経過時間を評価概念に入れていないため, 到着間隔 1000ms の場合において両手法の優劣を明確にすることができていない. 図中では省略しているが, 到着間隔 10000ms の場合には, どちらの手法でも待機状態に入るジョブの割合が 0 であるため, 性能の比較・評価を行えなかった. これらのことから, ジョブ同士の到着間隔に十分余裕がある場合の性能評価を行うことができる評価指標を今回用いた評価指標とは別に採用する必要がある.

7. ま と め

本研究では, スーパーコンピュータを利用する際にジョブを実行するキューを手動で指定するという従来の手法によって生じる処理効率と性能の低下に注目し, これを回避する手法を検討した. 検討に際しては, 実在のスーパーコンピュータシステムである HPC2500 を基にした数学モデルを作成し, 実際のログデータから同定した確率分布を利用するモンテカルロシミュレーションによって, 検討手法の性能を比較・評価した.

実験の結果, 各ジョブの到着間隔が 10ms, 100ms の場合には, 検討手法である最大実行可能残数選択法が従来の通常手法よりも良い性能を示すことがわかった. しかし, 到着間隔が 1000ms 以上の, 十分余裕のある到着間隔の場合には, 待機状態になるジョブが存在しないため, 性能の比較・評価が行えなかった.

今後の課題として, 新たな性能評価手法の考案, 数学モデルの改善, より高度な最適化手法の考案が考えられる.

謝　　辞

本研究の一部は, 平成 19 年度 「魅力ある大学院教育」イニシアティブ「シミュレーション科学を支える高度人材育成」および, 文部科学省科学研究費補助金 (若手研究 (B))「選択型コンテンツの放送型配信に関する研究」(課題番号：18700085) の研究助成によるものである. ここに記して謝意を表す.

参 考 文 献

1) HPC Systems, http://www.hpc.co.jp/
2) Message Passing Interface (MPI) Forum Home Page, http://www.mpi-forum.org/
3) Usage of SCS MAFFIN, http://www.affrc.go.jp/ja/info/scs/index.html
4) 産業総合技術研究所, http://www.aist.go.jp/
5) 学術情報メディアセンター, https://web.kudpc.kyoto-u.ac.jp/lsc/
6) 東京工業大学, http://www.titech.ac.jp/home-j.html
7) 川並 秀観, 義久 智樹, 金澤正憲, “バッチ処理型プログラム実行環境におけるキュー選択方式に関するシミュレーション評価,” 情報処理学会研究報告 (2007-EVA20), p. 11-16, March 16 2007.
8) Mersenne Twister with improved initialization, ”http://www.math.sci.hiroshima-u.ac.jp/m-mat/MT/”
9) Karl Czajkowski, Ian Foster, Nick Karonis, Carl Kesselman, Stuart Martin, Warren Smith and Steven Tuecke, “A Resource Management Architecture for Metacomputing Systems,” in *Proc. IPPS/SPDP'98 Workshop on Job Scheduling Strategies for Parallel Processing*, page 4–18. IEEE-P, March 1998.
10) Yoav Etsion and Dan Tsafrir, “A Short Survey of Commercial Cluster Batch Schedulers,” Technical Report 2005-13, He-brew University, May 2005.
11) Louis H. Turcotte, “A Survey of Software Environments for Exploiting Networked Computing Resources,” Draft Report, Engineering Research Center for Computational Field Simulations, Mississippi State, January 1993.
12) Mary Papakhian, “Comparing Job-Management Systems: The User's Perspective,” *IEEE Computational Science & Engineering*, vol. 5, Issue 2, April 1998.