PCクラスタにおけるジョブ管理

Job Management for PC Cluster

知的システムデザイン研究室　管理者委員会
吉田 純一

ver.1.0　　2000.05.12

1　はじめに

ジョブの管理は，ロードシェアリングやロードマネージメントとしても知られている．まずは，ジョブ管理システム（Job Management System : JMS）の基礎的な事項をまとめ，いくつかの代表的な JMS を紹介する．

2　ジョブ管理システムの役割

Saphir らは NASA の The Numerical Aerodynamic Simulation (NAS) parallel system やクラスタについて分析し，ジョブ管理システムの役割には次の３つの側面があるとした．

- **ユーザーサーバ（userserver）**

  ユーザはキューにジョブを追加したり，ジョブに対するリソースを要求したり，キューからジョブを削除したり，ジョブやキューの状態を尋ねたりする．ジョブ管理システムはこのようなユーザの要求の窓口にならなくてはならない．

- **ジョブスケジューラ（job scheduler）**

  ジョブ管理システムは，ジョブの種類，リソースの要求，リソースの利用可能性，スケジューリングのポリシーなどを考慮した上で，ジョブのスケジューリングを行わなくてはならない．

- **リソース管理（resource manager）**

  ジョブ管理システムは，リソースを監視し，スケジュール通りにリソースを割り振らなくてはならない．

3　JMS による管理と使用

JMS は，クラスタの全てのノードのジョブを管理することから，分散システムとして認識されることがあるが，実際にはマスターノードで一元的に管理しており，きわめて中央集権的である．したがって，全ての設定やログファイルなどは１カ所で管理される．JMS を使うと，PVM や MPI を用いた並列プログラミングを行わなくても，ジョブを適当に分散してくれる．

JMS は，実行中のジョブへの影響を最小限に押さえて，ジョブの設定を更新する必要がある．そのため，ジョブを実行する前にスクリプトが起動し，セキュリティのチェックや，クリーンアップをおこなう．また，ユーザは，いつでも自分のジョブを停止または中止することができる．なお，プロセスを停止させたり，中止した際にはクラスタのリソースを解放することが重要である．

クラスタにおけるジョブ管理には様々な側面がある．主なものを Table 1 にまとめた．以下では，これらについて順に解説する．

4　ジョブの種類

クラスタ上で，実行されるジョブはノードの利用方法やジョブの性質から，いくつかに分類することができる．まず，必要とするノード数で２つに分類が可能である．１つのノードで実行されるシングルジョブと，複

Table 1　ジョブ管理における問題点とその解決策

問題点	対応策	解決すべき問題
ジョブのプライオリティ	静的な処理 動的な処理	優先度の高いジョブの実行の遅れ 通信のオーバーヘッド．実装が困難
リソース要求	静的な処理 動的な処理	ロードバランスが悪い 通信のオーバーヘッド．実装が困難
リソースの共有	Dedicated Space Sharing Time Sharing	単純だが，効率が悪い タイリング問題，　large job問題 プロセスベースのジョブコントロールが必要
スケジューリング	Independent Gang	単純だが，効率が悪い 実装が困難
ローカルジョブとの競合	stay migration	ローカルジョブの実行の遅れ 移住先の選択，移住時の通信オーバーヘッド

数のノードで実行されるパラレルジョブである．ジョブの性質からもインタラクティブジョブとバッチジョブの２つに分類することができる．

- **インタラクティブジョブ（intaractive job）**

  ターミナルへの出力や，ユーザによる入力などが必要となるジョブ．素早いレスポンスが要求されるが，一般に計算機のリソース要求は小さい．

- **バッチジョブ（batch job）**

  バッチジョブは大規模なメモリ空間や，長い CPU タイムなど巨大なりソースを要求する．しかし，素早いレスポンスはあまり必要ではなく，ジョブはキューに投入され，リソースが空くの待ってから実行される．

インタラクティブジョブとバッチジョブは JMS によって同時に管理することが可能である．

また，JMS の管理外で実行されるジョブも存在する．これは foreign job と呼ばれる．これらは主に，クラスタの各ノードで独自に実行されるようなジョブである．これらはクラスタとしてのジョブ（クラスタジョブ）に対してローカルジョブと呼ばれる．

5　ジョブスケジューリング

ジョブのスケジューリング問題はプロセッサのスケジューリングによく似た問題であり，大変難しい問題である．スケジューリングには，時間をもとにスケジューリングを行うカレンダスケジューリング（calender scheduling）とイベントの発生によってスケジューリングを行うイベントスケジューリング（event scheduling）がある．スケジューリングに関する問題には様々なフェイズが存在する．表にこれらをまとめた．

ジョブのスケジューリングは一般に，なんらかのプライオリティ（順番）に応じて行われる．プライオリティは，キューへ投入された時間や，リソースの要求，ジョブの種類，ユーザの権限などを考慮して決定される．

プライオリティの決定は，静的または動的に行われる．それぞれ，つぎのような特徴を持つ

- **静的プライオリティ（static priority）**

  静的プライオリティを採用した場合は，あらかじめ定めたとおりにジョブを実行する．キューに登録された順に，実行していく非常にシンプルな方法である．

- **動的プライオリティ（dynamic priority）**

  動的にプライオリティを決定する場合，ジョブの実行は，他のジョブの進行や，新しいジョブの登録状況などに応じて変化する．また，昼間はインタラクティブジョブを優先し，夜間はバッチジョブを優先するといったポリシーを適用することもできる．動的プライオリティでは後述のタイリング（tiling）技術が重要であり，これによってシステムの性能は大きく左右される．

6 リソース管理

リソースの要求は，動的にも静的にも処理することができる．リソースの管理は，クラスタ全体のパフォーマンスを考える上できわめて重要である．並列のジョブを実行する場合にも，全てのノードが常に利用されているわけではない．このようなリソースについても有効に活用できることが望ましい．

静的リソース管理は，現在のクラスタの多くで利用されている．あらかじめ決定されたリソースをジョブが終了するまで固定し，ジョブを実行する．しかしながら，この方法では，リソースをうまく使うことは困難であり，ユーザ自身がロードバランスを考慮する必要がある．また，この方法ではノードのトラブルなどに対応できないという問題もある．

動的リソース管理では，ジョブの実行中に使用しないノードを解放することができる．ただし，これを実現するためにはジョブと JMS が密接に連携して動作する必要があり，実装は困難である．ジョブは JMS に対して非同期にリソースの解放や追加の要求を渡すため，JMS は常に全ノードのリソースを把握した上で，最適化を行わなくてはならない．

JMS とジョブとの高度な連携のためには，プログラミング言語やライブラリを改良しなければならない．並列化ライブラリである PVM や MPI には，これに必要となる機能が含まれている．

7 クラスタのノード共有

クラスタで，複数のジョブを効率的に事項するためには，ノードのシェアリングが重要となる．ノードのシェアリングのためには複数の方法がある．

7.1 Dedicated mode

Dedicated mode では，クラスタで同時に実行できるジョブは１つだけに限定する．また，各ノードには１つのプロセスが割り当てられる．最も単純な手法であるが，ジョブが完了し全てのリソースを完了するまでは次のジョブを実行できない．

7.2 Space sharing mode

Space sharing mode では，複数のジョブを複数のグループに分けて同時に実行することができる．一般に１ノードには１つのプロセスが割り当てられる．このときタイリング技術が重要になる．タイリングによる処理速度の向上のモデルを Fig. 1 に示す．

クラスタジョブのような大きなジョブを，ノードでの小さなジョブが妨げて実行できないという large-job 問題といわれる問題がある．この問題は前述の２つのノードシェアリング法では解決できない．そこで，これを解決するためには次に述べる time sharinig が必要である．

7.3 Time sharing

前述の２つの方法では，基本的に１ノードには１プロセスしか割り当てられない．一方，tme sharing mode では複数ユーザのプロセスを１つのノードに割り当てることが可能になる．Time sharing については複数のポリシーが存在する．

- **Independent scheduling**

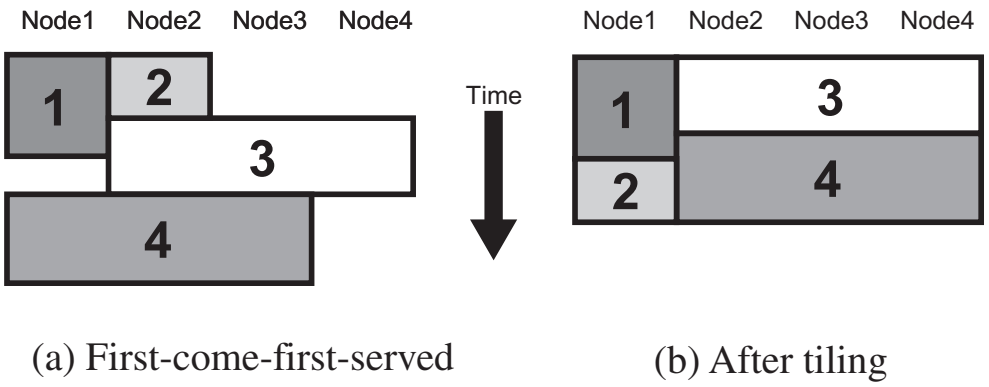

Fig. 1　タイリングによる処理効率の向上

もっとも単純な Time sharing 法としては各ノードの OS の機能を利用してスケジューリングを行う方法がある．これは，ローカルスケジューリングまたはインデペンデントスケジューリングと呼ばれる．ただし，この方法では並列のジョブに対するパフォーマンスがきわめて悪い．なぜなら，パラレルジョブの実行では，ノード間の協調が必要となるためである．

- **Gang Scheduling**

  Gang Scheduling ではパラレルジョブの実行のために，全てのノードのプロセスに対して同時にスケジューリングを行う．つまり，パラレルジョブの一つのプロセスがアクティブであれば，すべてのプロセスはアクティブである．この方法は，ホモなくラスタにおいてはパフォーマンスを向上させるといわれている．この方法の問題は,「同時に」という点にある．クラスタにおいては全てのノードで完全に同期がとれているわけではなく，システムクロックにずれがあることもある．Gang Scheduling を行った場合このシステムクロックのずれがクラスタ全体のパフォーマンスを低下させることになる．したがって，非同期なクラスタや，ワークステーションをつないだだけのヘテロなクラスタには向かないといえる．

7.4　ローカルジョブとの競合

クラスタのノードでローカルジョブが実行されている場合，スケジューリングはより複雑になる．このときローカルジョブは，クラスタジョブよりも優先度が高いとすると，このときにとるべき方法は以下の 2 つである．

- **stay scheme：** クラスタジョブが，ローカルジョブが終了するまで待つ

  この方法では移住のコストは内が，ローカルジョブ，クラスタジョブともにパフォーマンスが低下するという問題がある．

- **migrate scheme：** クラスタジョブを他のノードに移す

  同期待ちなどで，システム全体のパフォーマンスが低下する．また，移住先の状況を把握しておく必要がある．

8　ジョブマネージメントシステムの紹介

現在代表的な JMS として市販のもの，フリーのものを併せて 20 種類以上が存在する．Baker らは，代表的な JMS を調査し，その性能を分析した．

JMS を導入する際の注意点としては，MPI のサポートを考慮する必要がある．またジョブのスケジューリング，FIFO にするのか，ラウンドロビンを適用するのか，単一ジョブ実行時間の上限，登録ジョブ数の上限，Alerm（busy）によってキューを一時的に利用負荷にするロードの程度などをどの程度いじれるのかを確認し

なくてはならない．また，チェックポインティングシステムやプロセス移住の機能についても同様に考慮する必要がある．

- **DQS（Distributed Queuing System）**

  Florida State University によって開発されたフリーの JMS．MPI はサポートされているが，HPF には対応していない．商用バージョンは Codine と呼ばれ，ドイツの GENIAS GmbH が提供している．ちなみに，ここで紹介するものの中では唯一，Debian GNU/Linux 用のパッケージが公式に用意されている．

  `http://www.scri.fsu.edu/ pasko/dqs.html` (DQS)

  `http://www.scri.fsu.edu/ pasko/dqs.html` (Codine)

- **LSF（Load Sharing Facility）**

  カナダの Platform Computing Corporation によって提供されている商用パッケージであり，世界中で 20000 ライセンス以上が購入されている．”Scarable Parallel Computing [1)] ”には主にこれが紹介されている．

  `http://www.platform.com/`

- **Condor**

  University of Wisconsin によって開発されたフリーのパッケージである．信頼性のあるチェックポイントシステムやプロセス移住をサポートした最初のシステムである．リモートからの I/O にも対応している．ただしオーバーヘッドが大きいために，動作は遅い．PVM をサポートしている．

  `http://www.cs.wisc.edu/condor/`

- **NQS / Generic NQS / The Connect:Queue**

  使い勝ってがシンプルである．また NQS は JMS のデファクトスタンダードである．また多くの JMS が NQS との互換性（NQS のコマンドのサポート）を謳っている．Stirling Software によって 1980 年代初期に開発された NQS はさらに The Connect:Queue と呼ばれる商用パッケージに発展した．これをフリーなパッケージとして実装されたのが Generic NQS である．これはイギリスの University of Sheffield によって運営されている．

- **PBS（Portable Batch System）**

  NASA によって開発，配布されている．BSD licence のもとで自由に利用できる．感じとしては，Generic NQS をパワフルにしたようなものである．

  `http://pbs.mrj.com`

参考文献

1) Kai Hwang and Zhiwei Xu,『Scalable Parallel Computing』, WCB/McGraw-Hill (1998)