2007－HPC－111 (8)
2007／8／1

GridRPCアプリケーションのためのインタラクティブスケジューリングシステム

孫 顯 † 合田 憲人††

本稿では，GridRPCアプリケーションのためのインタラクティブスケジューリングシステムを提案し，その実装と性能評価結果について述べる．従来のGridRPCプログラムの実行ではユーザが自らジョブを割り当てる計算機を決定する必要がある．これに対して，提案システムではユーザがシステム上に実装されている複数のスケジューリングアルゴリズムの中から1つを選択することにより，自動的にスケジューリングが行われる．またユーザはアプリケーション実行中にスケジューリングアルゴリズムを変更することも可能である．性能評価では提案システムの有効性を議論する．

Interactive scheduling system for GridRPC applications

Hao Sun † and Kento Aida††

In this paper, we propose an interactive scheduling system for the GridRPC application, and the performance evaluation result of the implemented system is described. In GridRPC programming model, a user needs to decide which computer will execute the calculation libraries. On the other hand, in our proposal system, scheduling is automatically done by selecting one of the scheduling algorithms implemented in our system by the user. Moreover, it is also possible that the user changes the scheduling algorithm while executing the application. The effectiveness of the proposal system is discussed in the performance evaluation.

1. はじめに

グリッド上でのプログラミングモデルとしてGridRPCが普及しつつある．GridRPC[4]はリモート計算機上に実装されている計算ライブラリをユーザのクライアント計算機から呼び出すためのプログラミングモデルであり，クライアントプログラムのためのAPIが標準化されている．

ただし，GridRPCによるプログラミングでは開発者にさまざまな前提知識を要求する．ユーザ，すなわちアプリケーション開発者にさまざまな前提知識を要求する．例えば，ユーザは計算ライブラリを実行する計算機を事前に決定しておかなければならない．またクライアント計算機から呼び出される計算ライブラリは事前にリモート計算機にインストールされている必要がある．特に時々刻々変化する計算資源を利用するGridRPCアプリケーションの開発において前者は初心者にとっては容易なことではない．

† 東京工業大学
Tokyo Institute of Technology
†† 国立情報学研究所
National Institute of Informatics

これらの問題を解決するためにさまざまな研究が進められている．たとえばNinf-G[5]プロジェクトではNinf-Gをベースとする TFM(タスクファーミング)[1]の実装がなされた．TFMには計算を実行する計算機を自動的に決定するスケジューリング機能があるほか，計算機の障害発生時に計算の停止を回避する耐故障機能も備えている．また，Condor[2]ではユーザの要求要件を満たす計算機をMatchmaking手法により探索し計算を割り当てることができる．Relis-G[3]では，GridRPCの計算ライブラリをリモート計算機上に自動的に配備する機能を提供している．

しかしながら，これらの従来システムでは，柔軟なスケジューリングを実現する上での問題が残されている．ユーザがスケジューリング手法をカスタマイズしたい場合，TFMでは用いるアルゴリズムをユーザが実現する必要があり，CondorではClassAdを用いて記述しなければならない．また，アプリケーション実行中に計算機の状況を見てスケジューリングアルゴリズムを変更することができない．

本稿では，GridRPCアプリケーションにより柔軟なスケジューリングを実現するためのインタラクティブスケジューリングシステムを提案する．提案システ

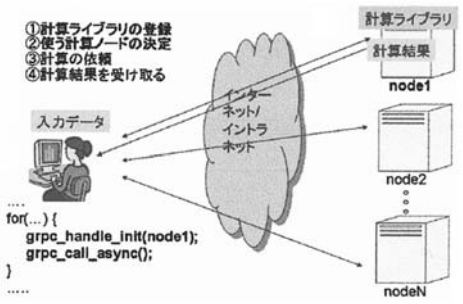

図 1 GridRPC プログラムの実行

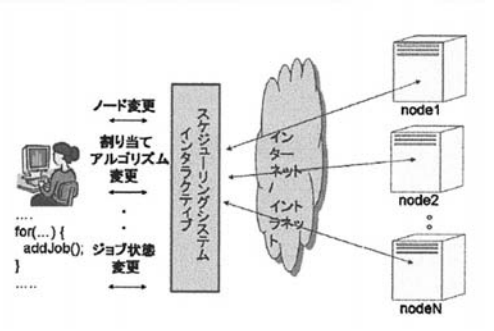

図 2 提案システム構成図

ムでは，複数のスケジューリングアルゴリズムがアドバイザという形で実装され，ユーザは利用したいアドバイザをシステム上で選択することにより利用するスケジューリングアルゴリズムを決定する．また，アドバイザはアプリケーション実行中に変更することも可能である．またユーザは適切なアドバイザを開発し本システムに組み込むことで，タスク・関数ごとに割り当て順序や冗長投入に関する設定ができる．デフォルトでは「計算性能優先アドバイザ」，「通信性能優先アドバイザ」，「実行実績に基づくアドバイザ」を提供している．ユーザは自分のアプリケーションの特性にしたがってアドバイザを選び，開発することが可能になる．

以下では，2 節で提案システムの概要を説明し，3 節では提案システムの設計と実装を説明する．4 節では実験を通して提案システムの有効性と問題点を考察し，5 節ではまとめと今後の課題を述べる．

2. 提案システム

2.1 従来システムとの比較

図 1 と図 2 を比較しながら，本システムの構成を説明する．

従来の GridRPC プログラミング手順

GridRPC プログラミングには基本的に以下のに三つのステップがある．

（1） 計算ライブラリの登録

（2） 計算ノードの決定

（3） プログラムの実行（計算依頼，結果の受け取り）

これに対して提案システムでは，ユーザが計算ノードを指定する必要はなく，インタラクティブスケジューリングシステムがユーザの選択したアドバイザに従って計算ノードを自動決定する．

2.2 プログラミングのインタフェース

従来 GridRPC プログラミングではユーザは図 1 左下のようにリモート計算機の計算ライブラリを呼び出すための関数ハンドルを生成し，呼び出しのためのコードをプログラム中に記述する．これに対して提案システムでは，図 2 左下のように提案システムにジョブを投入するためのコードを記述する．現在の実装では，本システムは JAVA により実装されているため，クライアントプログラムも JAVA に限定されているが，JavaVM により標準サポートされている JNI(Java Native Interface) を利用することで C 言語への対応も可能である．以下にプログラム例を示す．

JAVA の API

```
double result[N];
Scheduler sched=new Scheduler(configFile);
for(i=0;i<N;i++) {
    sched.addJob(" callFuncName ",arg1,
                              arg2,result[i]);
}
If(sched.AllJobfinish) {
    for() { result[i] の処理； }
}
```

ここでは，結果を格納する配列 result を先に生成し，スケジューラのハンドル sched を作成する．そして，sched ハンドルを利用しリモートにあるライブラリ callFuncName を引数 arg1，arg2 で呼び出す．計算が正しく終了した場合，結果が result[i] に格納される．最後に，すべてのジョブが終了してから計算結果の処理をする例を示している．現在の実装では，任意のジョブの終了判定ができていない．今後 AnyJobfinish のような API を追加し，対応する予定である．

3. システムの設計と実装

3.1 提案システムの内部構造

図 3 には提案システムの内部構造を示す．本システ

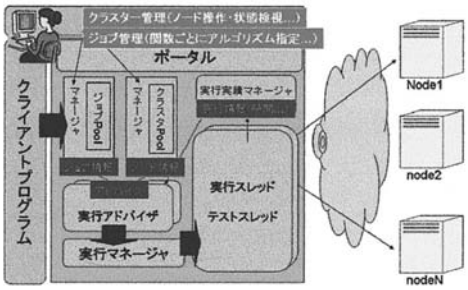

図 3 提案システム構成図

ムは以下の構成要素を含む.

- ポータル：
  提案システムのユーザインタフェース．ユーザがポータルを利用しクラスタ，ジョブの管理をする．
- ジョブプール：
  ジョブ情報を保存する場所．ジョブプールは 3 種類存在し，詳しくは後述する．アドバイザはジョブプールマネージャを通してこのジョブプールから計算ノードに割り当てるジョブを取り出す．
- ジョブプールマネージャ：
  ジョブプール中のジョブを管理する．具体的にはジョブのプールへの追加，取り出しを行う．ジョブプールマネージャはポータルにジョブ状況を反映する．
- クラスタプール：
  ジョブの実行に利用可能なクラスタ候補に関する情報が保存される．アドバイザがクラスタプールマネージャを通してこのプールからクラスタの探索や取り出しを行う．
- クラスタプールマネージャ：
  クラスタプールを管理する．具体的にはクラスタの追加，取り出しを行う．クラスタプールマネージャはポータルにクラスタの状況を反映する．
- 実行実績マネージャ：
  ジョブの実行結果を管理する．具体的にはジョブの実行時間に関する情報の追加，取り出しを行う．
- アドバイザ：
  スケジューリング，即ちジョブを割り当てるクラスタを決定する．実行マネージャに割り当ての候補を提供する．
- 実行マネージャ：
  アドバイザからのアドバイスに従いジョブを実行させる．

3.2 ポータル

提案システムでは，ユーザがポータルを通してスケジューリングを動的に制御することができる．具体的

図 4 インタフェースの利用例

には，ユーザは計算を割り当てる候補となる計算機クラスタおよびスケジューリングアルゴリズム（アドバイザ）をアプリケーション実行中でも変更することができる．本システムではポータルの実装には GWT(Google Web Toolkit)[6] を利用している．図 4 は，ユーザがポータル上で計算機クラスタのノード数を変更する画面の例を示している．このように，ユーザの操作がイベントとして捕らえられ，スケジューリングシステムに反映されることにより動的にスケジューリングの挙動を変えることができる．

3.3 ジョブプールマネージャ

提案システムでは，ジョブを管理するために三つのジョブプールを用いる．

- 実行待ちジョブプール
  ユーザから投入された実行待ちジョブが保存される．ジョブが実行状態に移る時，実行中ジョブプールに移されこのプールから削除される．提案システムでは，ジョブの実行を担当する実行マネージャがこのプールにジョブが存在するか否かを常に確かめている．ジョブが存在する場合，実行スレッドが生成される．
- 実行中ジョブプール
  実行中のジョブが保存される．ジョブが終了状態に移る時このプールから削除される．アドバイザがこのプールにあるジョブを制御することにより，アドバイスを修正することが可能になっている．
- 再実行ジョブプール
  実行に失敗したジョブが保存される．何らかの理由で実行に失敗したジョブが実行中ジョブプールからこのプールに移される．実行マネージャが実

行待ちジョブプールと同じようにこのプールにあるジョブを監視する．

3.4 クラスタプールマネージャ

クラスタプールマネージャは各クラスタの情報を格納するクラスタプールを管理し，さらにポータルと通信するためのインタフェースを備えている．実装にはGWT の API を使った．

3.5 実行実績マネージャ

提案システムでは，ジョブの実行情報を分類し（例：データ転送時間，ジョブの計算時間，関数ハンドル作成時間），データベースに格納している．アドバイザが必要な情報を実行実績マネージャから取得し，適切なアドバイスを作成する．

3.6 実行アドバイザの設計と実装

アドバイザの種類

提案システムでは現在，典型的なスケジューリングを実現する以下の 3 つのアドバイザが実装されている．

- 計算性能優先アドバイザ
  本アドバイザは計算機の CPU の性能を優先的に考え，計算性能の最も高いと予想される計算機クラスタにジョブを割り当てる．計算指向のアプリケーションに有効なアルゴリズムとなる．
- 通信性能優先アドバイザ
  本アドバイザは通信性能を優先的に考え，クライアントとの通信性能の最も高いと予想される計算機クラスタにジョブを割り当てる．通信指向のアプリケーションに有効なアルゴリズムとなる．
- 実行実績に基づくアドバイザ
  本アドバイザはアプリケーション実行中各計算機クラスタ上のジョブの実行履歴を保存しておき，実行履歴より実行時間が最短となることが予想される計算機クラスタにジョブを割り当てる．本アドバイザはアプリケーションが計算指向か通信指向かが不明な場合に有効と考えられる．しかし，実行履歴がある程度獲得できないと効果的に動作しない可能性がある．

アドバイザの実装

図 5 はアドバイザの内部構造を示している．任意のアドバイザがアドバイスを作成するとき，タスクとクラスタの事前選択フェーズがある．これはアドバイザが提供できるアドバイスが異なるため，アドバイスを出せないジョブ，クラスタを選ばないようにするための仕組みである．ジョブ，クラスタ数が多い時，この仕組みによりメモリが節約でき，スケジューリングシステムの全体の性能を向上させる．アドバイザがジョブとクラスタに関する情報と実行情報の基にアドバイスを作成する．その後，実行マネージャがアドバイスを受け入れ，ジョブの実行を管理する．

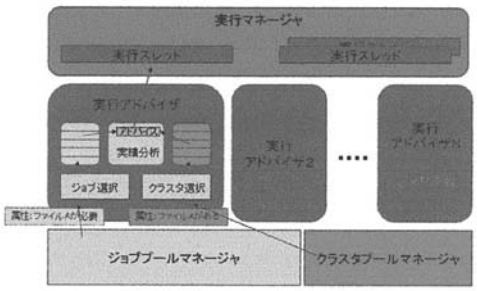

図 5 実行アドバイザの内部構造

3.7 実行マネージャ

実行マネージャはジョブプールマネージャから実行待ちジョブを取得し，ユーザが指定した実行アドバイザからアドバイスを取得する．その後，アドバイスに従い，新たに実行スレッドを生成しそのジョブの実行を管理する．

4. 評　　価

この節では提案したスケジューリングシステムの有用性を示すために以下の実験を行った．

アドバイザ切り替えによる実行時間短縮の検証

システムに提供される三つのアドバイザと四種類のテストプログラム（計算指向なプログラム，通信指向なプログラム，計算と通信の両方を行うプログラム，以上の 3 種類を組み合わせたプログラム）について実験を行った (表 1)．以降，実験結果を分析し各アドバイザの特性を示し，動的にアドバイザを切り替えた場合の有用性を示す．

以上の実験を行うために，以下のような特性を持つ実験環境を構築した．

4.1 評価環境

図 6 に示した通り，本実験では 2 台のローカルクラスタを利用した．gs 計算機クラスタ（以下 gs）には利用可能なノードが少なく各ノードの計算性能も比較的低い．gk 計算機クラスタ（以下 gk）には利用可能なノードが多く，各ノードの計算性能も比較的高い．ただし，gs と gk はともに同一 LAN 上に設置されているため，通信性能の差を出すために linux のカーネルに組み込まれたネットワークエミュレーションソフトウェア netem[7] を使った．今回の実験では gk に対

表 1　実験環境

テストプログラムの種類

useCPU	固定サイズの行列積を計算するプログラム
useNet	固定サイズのデータを転送するプログラム
useBoth	useCPU と useNet を合併したプログラム
useAny	useCPU，useNet と useBoth を一定の比率で合併したプログラム

アドバイザの種類

CPUAd	計算性能優先アドバイザ
NetAd	通信性能優先アドバイザ
ExpAd	実行実績に基づくアドバイザ
Roundrobin	ラウンドロビン（比較対象アルゴリズム）

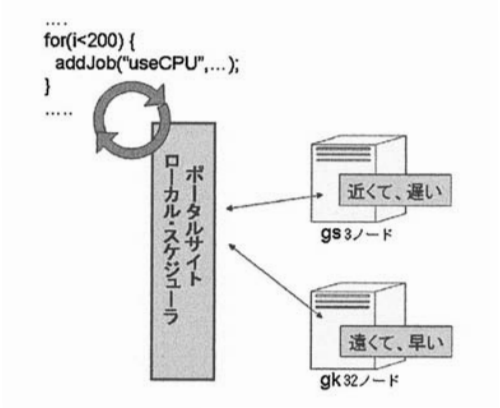

図 6　テストプログラムの実行

し一定時間のネットワークの遅延を発生させた．

4.2　実験のパターンと結果分析

以上のテストプログラムとアドバイザを組み合わせてテストすることにより，各アドバイザの特性を示す．さらにアドバイザ間で性能を比較し，動的にアドバイザを切り替える有用性を示す．今回の実験では，固定サイズのデータと固定サイズの行列計算プログラムを用いた．

4.2.1　useCPU プログラムのテスト結果と分析

表 2　useCPU プログラムのテスト結果

アドバイザ	Round robin	NetAd	CPUAd	ExpAd
実行時間	10:04	09:00	07:59	08:41
性能向上比	–	11%	21%	14%

計算指向な useCPU プログラムを 200 回実行し，ラウンドロビンアルゴリズムを用いた結果，10 分 04 秒を要した (表 2)．gs と gk に平均的に割り当てたラウンドロビンより，データ転送が少ない useCPU プログラムを gk に多く割り当てることで，全体の実行時間の短縮が図られる．計算性能優先アドバイザがジョブ実行情報から各クラスタでの実行時間を収集・比較した結果，gk に優先してジョブ割り当てたことで，実行時間が 21%短縮された．通信性能優先アドバイザは通信性能を重要視するためテスト開始当初は gs にジョブを優先して割り当てているが，アドバイザは各クラスタにおけるジョブの実行時間を常に収集・比較しているため，実行が進むにつれ gs における実行時間が gk における実行時間を上回っていると判断し，gk に優先して割り当てた．このため，ラウンドロビンよりも良い性能が得られた．このようなケースでは，ユーザが動的に計算性能優先を選ぶことでアプリケーションの実行時間を短縮できる．

4.2.2　useNet プログラムのテスト結果と分析

表 3　useNet プログラムのテスト結果

アドバイザ	Round robin	NetAd	CPUAd	ExpAd
実行時間	08:32	09:26	10:27	10:49
性能向上比	–	-11%	-22%	-27%

通信指向な useNet プログラムを 200 回実行し，ラウンドロビンアルゴリズムを用いた結果，8 分 32 秒を要した (表 3)．useNet プログラムはデータ転送処理が中心のため，ネットワーク上において近くに設置された gs に割り当てると実行時間が短縮されると考えられる．実際に gs におけるジョブ単体の実行時間は gk における実行時間を下回ったため，いずれのアドバイザも useNet プログラムを gs に優先して割り当てた．しかし，本実験環境では gs の計算ノード数が少ないため，過剰に割り当てられたジョブの再実行が必要になり，ラウンドロビンと比較して性能が低下した．この問題の回避方法は現在検討中である．

4.2.3　useBoth プログラムのテスト結果と分析

表 4　useBoth プログラムのテスト結果

アドバイザ	Round robin	NetAd	CPUAd	ExpAd
実行時間	10:13	08:17	08:12	08:25
性能向上比	–	19%	20%	18%

useBoth プログラムを 200 回実行し，ラウンドロビンアルゴリズムを用いた結果，10 分 13 秒を要した (表 4)．useBoth プログラムは通信指向であり，かつ計算指向なアプリケーションでもある．ただし，計算処理が通信処理より多く時間を費やしている．このプログラムの gk でのジョブ実行時間が gs より短いた

め，ジョブを計算ノードの多い gk に多く割り当てることで，全体実行時間の短縮が図られる．その結果，計算性能優先アドバイザが通信性能優先アドバイザより若干よい性能を示し，ともにラウンドロビンよりよい性能を示した．実行実績に基づくアドバイザはジョブの性質の分析処理の影響により上記の二つのアドバイザによる結果よりも多少時間を要している．

4.2.4 useAny プログラムのテスト結果と分析

表 5 useAny プログラムのテスト結果

アドバイザ	Round robin	NetAd	CPUAd	ExpAd
実行時間	10:58	08:54	08:17	08:35
性能向上比	–	19%	24%	22%

useAny プログラムは useBoth プログラム 1/2, useCPU プログラムを 1/3, useNet プログラムを 1/6 の確率でランダムに実行させるものである．useAny プログラムを 200 回実行し，ラウンドロビンアルゴリズムを用いた結果，10 分 58 秒を要した (表 5)．useAny プログラムはデータ転送処理より計算処理を多く行うプログラムであるため，計算性能優先アドバイザにより最も良い性能が得られた．4.2.3 節と同様，実行実績に基づくアドバイザはジョブの性質の分析処理に多少時間を要しているが、通信性能優先アドバイザによる結果よりもよい結果が得られた。

5. まとめと今後の課題

本稿ではユーザインタラクティブなスケジューリングシステムを提案し，その実装と性能評価結果を示した．実験を通してアドバイザによるスケジューリングに基づく実行時間が単純なスケジューリングによる実行時間よりも短縮され，その有用性を確認した．今後の課題として，アドバイザによる計算機クラスタの評価時に計算ノード数を考慮するような機構の開発や，多重実行機能の追加，他のスケジューリングを実現するアドバイザの実装と評価が挙げられる．

参 考 文 献

1) 谷村 勇輔: GridRPC を用いたタスクファーミング API ライブラリの設計 (http://ninf.apgrid.org/event/ninfg06/slides/tanimura.pdf).
2) Raman,R., Livny,M. and Solomon,M. : Matchmaking: Distributed Resource Management for High Throughput Computing, *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, pp.28–31(1998).
3) Watanabe,H., Honda,H., Yuba,T., Tanaka,Y. and Sato,M.: Relis-G : Remote Library Install System for Computational Grids(Grid Systems), *Transactions of Information Processing Society of Japan, Information Processing Society of Japan (IPSJ)*, pp.196–206(2004).
4) A GridRPC Model and API for End-User Applications (http://www.ogf.org/documents/GFD.52.pdf).
5) Tanaka,Y., Nakada,H., Sekiguchi,S., Suzumura,T. and Matsuoka,S.: Ninf-G: A Reference Implementation of RPC-based Programming Middleware for Grid Computing, *Journal of Grid Computing*,Vol.1,No.1,pp.41-51(2003)
6) Google Web ToolKit (http://code.google.com/webtoolkit/).
7) S. Hemminger: Network Emulation with NetEm (http://developer.osdl.org/shemminger/LCA2005_netem.pdf).