R3Qによる進化型計算の中粒度Gridスケジューリング

松浦芳樹[†1] 大倉和博[†2] 松村嘉之[†3]
藤本典幸[†4] 萩原兼一[†4]

本稿では，進化型計算の隔世代交替モデルを Grid Computing に実装することを試みる．隔世代交替モデルでは，世代交替の際に各計算機の同期を世代ごとに行う必要がある．しかし，Grid は動的かつヘテロな計算機環境であるため，低速計算機がボトルネックとなり，同期処理が効率良く行えない．さらに，分散可能なタスクが中粒度タスクゆえに，ネットワーク環境における通信遅延が問題となり，分散計算の効果が低下してしまう．本稿では，これらの問題に対し，Grid スケジューリングの立場から対処する．まず，同期処理に対し，タスクスケジューリングアルゴリズム RR（list scheduling with Round-robin order Replication）を適用する．さらに，通信遅延に対し，RR における強制終了の効率を考慮したスケジューリング手法 RWQ（Remote Work Queue）を提案し，RR と RWQ を併用する R3Q を提案する．いくつかの手法を比較した計算機実験の結果より，R3Q により通信遅延時間に左右されず高い並列化効率が得られることを確認した．これより，R3Q により EC の隔世代交替モデルにおいても，進化ダイナミクスを変えることなく，効率良く Grid に実装できることを示した．

A Grid Scheduling for an Evolutionary Computation Using R3Q

Yoshiki Matsuura,[†1] Kazuhiro Ohkura,[†2]
Yoshiyuki Matsumura,[†3] Noriyuki Fujimoto[†4]
and Kenichi Hagihara[†4]

A computational method for implementation of Evolutionary Computation (EC) optimization methods in grid computing environments is discussed. In general, EC methods implemented in a grid environment need jobs to be synchronized each generation. In addition, EC problems incur a significant communication overhead, because of the fact that they are what we refer to as “medium-grained” tasks. We deal with these problems by using a dynamic grid scheduling method without prediction. In this paper, list scheduling with Round-robin order Replication (RR) is adopted to reduce waiting times due to synchronisation. However, RR is suitable only for coarse-grained tasks. For medium-grained tasks, we propose a new technique for reducing the communication overhead, called the Remote Work Queue (RWQ) method. This is able both to reduce communication time and obtain efficient forced termination of tasks. We then define R3Q (Round-robin Replication Remote Work Queue) as RWQ with RR like task replication. In order to evaluate R3Q, computer simulations are conducted by using a test problem that requires a constant number of computations, but where varying computer resources in the grid nodes result in varying execution times. Our results show that R3Q can reduce both synchronous waiting time and communication time, and provides efficient forced termination of tasks in comparison with other methods.

1. はじめに

進化型計算（EC: Evolutionary Computation）[1,2]は最適化問題に対して有効な結果を示し，その適用範囲は多岐にわたっている．しかし，EC は解の探索に膨大な時間を要し，この問題点は実問題を解く際により顕著になる．これに対し，本稿では Grid Computing [2]（以下 Grid と呼ぶ）を適用し，複数の異なる計算機による処理の並列・分散化により実行時間の短縮を図る．Grid 環境は各計算資源の性能やアーキテク

†1 神戸大学大学院自然科学研究科
Graduate School of Science and Technology, Kobe University
†2 広島大学大学院工学研究科
Graduate School of Engineering, Hiroshima University
†3 信州大学繊維学部
Faculty of Textile Science and Technology, Shinshu University
†4 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University

チャが異なるヘテロな環境であり，かつ他ユーザやネットワーク回線の影響により環境が動的に変化する特徴を持つ．そのため，隔世代交替モデル（Generational Reproduction）を採用した EC を Grid に実装するには，世代交替を行うために計算機の同期処理が必要となり，その際に低速計算機がボトルネックとなる．さらに，EC で取り扱う問題がマルチロボット問題や蛋白質立体構造解析問題などの場合，広域ネットワーク上の通信遅延時間より大きい計算時間を要するが，通信遅延時間を無視できない程度の計算量のタスク（中粒度タスク）となり，Grid には本来不向きとされている．

そのため，EC を Grid に実装するにあたり，EC の連続世代交替モデルを非同期型に改良する[9),11)]，島モデル型にすることにより通信回数を減らす[1),8),13)]，といった方法が研究されている．しかし，世代交替モデルの違いにより進化ダイナミクスが変化し，問題によっては解の探索が困難になる場合がある．また，島モデル型で並列数が島数分になる場合，高い並列性が得られない．そこで，本稿では EC の隔世代交替モデルの実装を試み，EC の個体評価の並列・分散計算を Grid に効率良く実装する手法を，Grid タスクスケジューリングの立場から対処する．

タスクスケジューリングには様々な手法が提案されているが，Grid 環境中の計算機の計算負荷が時々刻々と変化することから，負荷変動を予測することは本質的に困難と考え，動的かつ予測情報を用いないスケジューリング手法に着目する．特に，EC における同期待ち時間の短縮という観点から，非予測型動的スケジューリングアルゴリズム RR（list scheduling with Round-robin order Replication）[3),6)] を適用する．この RR はタスクの複製と強制終了により，低速計算機のボトルネックを解消することができる．しかし，現状，RR は通信遅延時間が無視できるほどの粗粒度タスクにおいてのみ，その有効性が示されている[4)]．そのため，EC の中粒度タスクを並列化する際のもう 1 つの問題である通信時間によって，計算機の CPU 待ち時間が増大する問題が露になる．

そこで，本稿では，RR の複製・強制終了の操作を有効に利用しつつ，CPU 待ち時間をなくすスケジューリング手法として，RWQ（Remote Work Queue）を提案する．そして，EC の中粒度タスクを Grid に実装する際の同期処理と通信時間の問題を，RR と RWQ を併用したスケジューリング手法 R3Q により対処する．性能テスト問題として，EC の隔世代モデルの一手法である ES（Evolution Strategies）による実関数最適化問題に，中粒度程度の計算粒度となるよう LU 分解を負荷として与え，R3Q の有効性を検証する．

本稿は，以下のように構成されている．まず，2 章では，進化型計算の概要を述べ，本稿で適用した $(\mu+\lambda)$-ES のアルゴリズムを示す．3 章では，タスクスケジューリングにおける関連研究について述べ，その中で本稿が同期処理の問題に対し適用した RR のスケジューリングについて述べる．4 章では，EC のような中粒度タスク向けに本稿で提案するスケジューリング手法 R3Q の計算手順を示す．そして，5 章では，計算機実験について述べ，R3Q の有効性を検証する．最後に，6 章では，本稿のまとめを記す．

2. 進化型計算 EC

EC[12)] は生物進化（選択淘汰，突然変異）に着想を得て考え出された解探索手法である．EC は基本的に対象とする問題の解候補の生成と適応度に基づく解候補の更新を用いた多点探索法である．確率的探索を行うため，解候補として個体が定義でき，なんらかの評価を行うことができれば適用可能であり，対象とする問題の評価関数の連続性や微分情報を必要としないという特徴がある．EC の手法は，1960 年代に開発された遺伝的アルゴリズム（Genetic Algorithms: GA），進化戦略（Evolution Strategies: ES），進化的プログラミング（Evolutionary Programming: EP）の 3 つの主な領域があり，目的変数の表現形式・突然変異・組み換え・選択などにより特徴づけられる．この中で，本稿で適用した $(\mu+\lambda)$-ES のアルゴリズムを以下に示す：

(1) 世代 $g = 0$ とし，μ 個体からなる初期集団をランダムに生成する．
(2) 親個体の適応度を目的関数 f を基に計算する．
(3) 1 個体の親から平均して λ/μ 個の子孫を突然変異によって生成し，合計 λ 個体の子孫を生成する．
(4) 子孫の個体の適応度を f を基に計算する．
(5) μ 個の親と λ 個の子孫を適応度の高い順に並べ，良いものから μ 個を次世代の親として選択する．
(6) 終了条件を満たさないときは，$g = g+1$ とし，(3) に戻る．

ここで，$(\mu+\lambda)$-ES において独立計算可能な部分は個体の適応度計算であり，マルチロボット問題などのような中粒度タスクを扱うと広域ネットワーク上の通信遅延時間を無視できない．また，親個体の選択と子個体の生成はすべての個体の適応度計算が終了した後に行うため，各計算機の同期をとる必要がある．なお，この同期のための処理を以降では，同期処理と呼ぶ．

3. タスクスケジューリング

3.1 動的スケジューリング

タスクスケジューリングには，スケジューリングに関するすべての決定をスケジュールの実行前に行う静的スケジューリング，一部またはすべての決定を実行中に行う動的スケジューリングがある．Grid 環境中のタスクを実行する複数の計算機（ワーカ）の計算負荷は時々刻々と変化するため，ユーザが使用する計算機（マスタ）でのスケジューリングには，動的スケジューリングが適している．この動的スケジューリングはさらに，実行中のワーカの CPU 負荷などの性能状況をモニタリングすることで，性能予測を行う予測型と，モニタリング情報を用いない非予測型に分類できる．Grid 環境は他ユーザなどの影響による突発的な負荷変動が起こりうることから，基本的に予測は困難であると考え，本稿では予測情報を用いない非予測型スケジューリングに着目する．

非予測型の古典的手法に WQ（Work Queue）[7] がある．WQ はマスタが利用可能になったワーカに順次タスクを割り与える手法であり，クラスタのようなホモな環境において有効性を示す．しかし，Grid のようなヘテロな環境では低速計算機がボトルネックとなり，効率良くタスクを分散できない．この低速計算機のボトルネックを解消する手法として，WQR（Work Queue Replication）[10] がある．WQR は同期処理の際に，すでにタスクを完了したワーカが未完のタスクを複製することで，低速計算機のボトルネックを解消する．しかし，WQR ではどれだけタスクを複製すればよいかは未知である．そこで，タスクの複製をラウンドロビン方式で行い，さらにタスクを完了すれば，実行中の同一タスクを強制終了する RR [3),6)] がある．RR は粗粒度タスクにおいて，理論的に性能が保証されており [3),5)]，予測誤差が 30%以上になると予測型スケジューリング手法よりも性能が良くなることがシミュレーション実験において示されている．本稿では，EC の同期処理の問題に対し，この RR を適用する．

3.2 RR による同期処理

本稿で用いる RR のアルゴリズムの流れを示す：

(1) 全タスクをマスタのキュー Q に入れる．
(2) Q の先頭 M 個のタスクを取り出して，M 台の各ワーカに 1 つずつタスクを割り当てる．
(3) ワーカへ割り当てられたタスクのうち，マスタへ計算結果が 1 つ返ってくるのを待つ．
(4) Q の先頭タスクを 1 つ取り出して，計算結果を返したワーカに割り当てる．

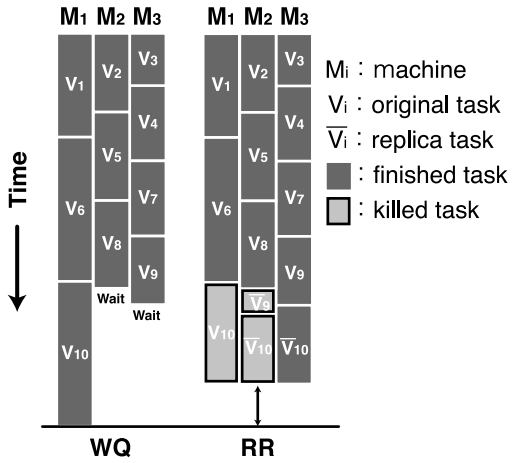


図 1 タスク処理の流れによる WQ と RR の比較
Fig. 1 Comparison of WQ and RR by task flow.

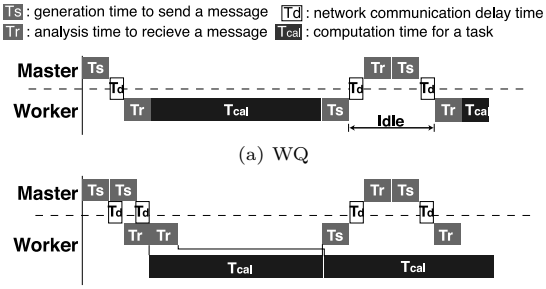
(5) Q が空になるまで，(3) から (4) を繰り返す．
(6) Q が空になった時点での，M 台の各ワーカで実行中のタスクの名前の集合を R とする．
(7) ワーカへ割り当てたタスクのうち，マスタへ計算結果が 1 つ返ってくるのを待つ．
(8) 完了したタスク S の他のインスタンスが，もしあれば，すべての計算機で強制終了する．
(9) R から S を削除する．
(10) R 内のタスクをラウンドロビン順で 1 つ選び，タスクを実行中でないすべての計算機に複製する．
(11) R が空になるまで，(7) から (10) を繰り返す．

ここで，RR は全タスクのうちの $(M-1)$ 個のタスクだけを重複して複数の計算機で実行することに注意されたい．この重複実行はタスク複製と呼ばれる．Grid を構成する各計算機は性能が異なり，かつ Grid 計算以外のジョブの実行により，その計算負荷は時々刻々と変化する．このため最初に割り当てられたタスク（オリジナルタスク）より複製タスクの方が早く実行完了する場合がある．これにより，RR では WQ における同期待ち時間を高速計算機により解消することが可能になる（例，**図 1** の M_3）．しかし，これらは通信遅延時間が無視できるほどの粗粒度タスクの場合に成立し，EC の適応度計算で中粒度タスクとなる場合は，RR の複製/強制終了の操作は時間短縮に十分貢献しないことが多い．

4. EC における Grid スケジューリング

4.1 通信遅延時間の隠蔽

EC における中粒度タスクを Grid に実装する際，



(a) WQ

(b) Slide Transfer Method and RWQ

図 2 WQ，スライド転送，RWQ における CPU 利用
Fig. 2 CPU utilization with WQ, Slide Transfer and RWQ.

RR を含め前節で述べたどの動的スケジューリング手法を用いた場合も同期処理以外のもう 1 つの問題点となる通信遅延時間は無視できない．これは**図 2** (a) に示すように，タスクの計算時間に対し，転送時間の割合が比較的大きいため，ワーカの CPU がアイドル状態になることに起因する．この問題に対し，通信遅延時間を隠蔽するスライド転送[9] が提案されている (図 2 (b))．スライド転送は GridRPC である Ninf を用い，ワーカに実行タスク以外に次のタスクを待機させておくことで，ワーカに絶え間なくタスクを実行させる．これにより，ワーカがアイドル状態になることなくタスクを実行でき，見かけ上通信遅延時間をほぼ隠蔽することが可能になる．

しかし，隔世代交代モデルの場合，スライド転送のようにタスクを待機させて通信遅延時間を隠蔽する手法では，転送タスク数が増加してしまうため，低速計算機のボトルネックはより顕著になり，同期待ち時間が増大してしまうことが予想される．

4.2 R3Q の提案

R3Q では，スライド転送と同様，ワーカに次のタスクを待機させ，通信遅延時間を見かけ上なくし，さらに，同期待ち時間増大の問題を RR を併用することで対処する．この際，RR の強制終了を行いやすくするため，タスクの待機方法にキューを用いた RWQ (Remote Work Queue) を提案する．スライド転送と異なる RWQ の特徴は，陽にマスタのスレッド数を多くする (マスタでのスレッド数が，スライド転送ではワーカ数に等しく，RWQ ではワーカ数の 2 倍になる) ことで，その分どのワーカにどのタスクを転送しているかをマスタで管理しやすくし，強制終了命令を送りやすくする点にある．このため，スライド転送は少ないスレッド数で済むため同期処理が不要な問題に向いており，RWQ は隔世代交代モデルを用いる場合のように，同期処理のために複製や強制終了を頻繁に行う場合に向いている．以上より，同期処理と通信時間の両方を改善する手法として RWQ に RR を併用する R3Q を提案する．

最初に，RWQ について説明する．RWQ はネットワーク上にあるワーカにキューを持たせ，タスクを溜めておく (このタスクを保持タスクと呼ぶ) ことで，ワーカの CPU アイドル状態をなくす．タスクが完了しても，マスタからの受信を待たずに次のタスクを実行することができ，スライド転送と同様に，ワーカがアイドル状態になることなくつねにタスクを実行し続けることができる (図 2 (b))．さらに，強制終了を保持タスクから行うことで，実行中のタスクの強制終了回数を抑制できる．RWQ のアルゴリズムは以下のとおりである：

(1) 全タスクをマスタのキュー Q に入れる．ワーカはワーカ 1，ワーカ 2，$\cdots$ ワーカ M の M 台とする．
(2) ワーカ k は自身の CPU 数だけ実行スレッドを起動する ($k \in \{1, 2, \cdots, M\}$)．
(3) マスタはワーカ k へ (CPU 数 $\times 2$) 個のタスクを転送し，ワーカ k が持つキュー $Q_{r,k}$ へ挿入する ($k \in \{1, 2, \cdots, M\}$)．
(4) ワーカ k の実行スレッドは $Q_{r,k}$ のタスクを実行し，結果をマスタへ返信する ($k \in \{1, 2, \cdots, M\}$)．
(5) マスタはワーカ k から返信されると，Q の先頭タスクをワーカ k の $Q_{r,k}$ に溜める．
(6) マスタの Q が空になるまで，(4) から (5) を繰り返す．

RWQ ではワーカ k に (CPU 数 $\times 2$) 個のタスクを転送することで，ワーカ k の 1 つの CPU につき，1 つのタスクがキュー $Q_{r,k}$ に溜まる．ワーカ k の実行スレッドは，マスタから転送されるタスクを待たずに $Q_{r,k}$ から次のタスクを取り出し実行できる．これにより，RWQ では 1 タスクの計算時間の最小値が通信遅延時間以上である場合，見かけ上通信遅延時間を無視できる．

次に，RWQ に RR を併用する際，RR の強制終了を保持タスクから行えるように工夫する．すなわち，タスクをワーカ k の $Q_{r,k}$ から取り除くだけにする．ここで，M 台の計算機ですべて 1 つの CPU を持つ場合を考えると，RWQ ではマスタから全ワーカの総 CPU 数の 2 倍 ($= 2M$) のタスクが転送され，そのうちの実行中のタスクは CPU 数 (M) で，残りの CPU 数 (M) が保持タスクとなる．タスクは転送さ

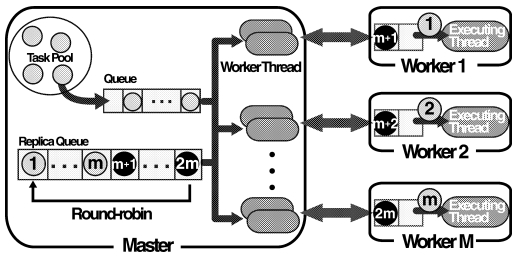


図 3 R3Q Grid システム
Fig. 3 R3Q Grid System.

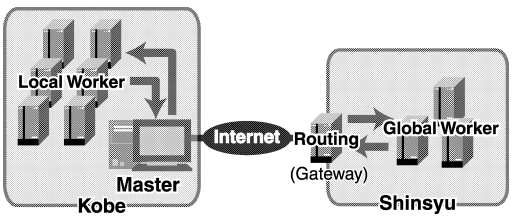


図 4 実験で利用した計算機環境
Fig. 4 Grid environment used in our experiment.

れた順に実行されるため，保持タスクは後から転送された M 個のタスクとなる．この保持タスクを複製することで，強制終了の対象となるタスクを保持タスクにできる．よって，RWQ に RR を併用する際，タスク複製はマスタが転送した逆順にラウンドロビン方式で行うこととし，この併用スケジューリング手法を R3Q と呼ぶ（**図 3**）．

5. 計算機実験

5.1 計算機システム

最初に，本実験で用いた計算機（神戸大学 7 台，信州大学 4 台）のスペックとテスト問題として使用した 3 種類のタスク（M1 で 100[ms]，500[ms]，1,000[ms] の時間がかかるタスク）を計算させた場合の計算時間とその速度比を**表 1** に示す．なお，この速度比の合計は Grid の最大速度上昇比となる．次に，計算機実験で用いたネットワーク環境を**図 4** に示す．これは神戸大学にある M1 の計算機をマスタとし，L1–L6 が神戸大学のローカルエリア内のワーカ計算機，R1 が信州大学のグローバル IP を持つ計算機，W1–W3 が信州大学内のローカルエリア内のワーカ計算機である．ここで，神戸大学から信州大学のローカルエリアの計算機は直接通信できず，ルーティングの役割を果たす R1 を通して，W1–W3 へタスクを割り当てている．また，本実験では，通信時間として CPU の空き時間を計測するため，計測可能なように複数 CPU を持つ計算機もシングル CPU と仮定し，各計算機が一度に実行するタスクは 1 つとした．本稿では，通信プロトコルにすべて SOAP を用いて実装した．マスタでは Java のマルチスレッドプログラミングにより，スレッド（WorkerThread と呼ぶ）を生成し，WorkerThread が各ワーカと GLUE SOAP によりデータを送信する．ワーカでは Tomcat サーバと Axis によりマスタからの送信データを受信し，タスクを実行する．実行結果は Axis SOAP により返信する．

5.2 問題設定

提案手法 R3Q により同期待ち時間と通信時間が減少するかを検証するため，計算粒度が一定の関数最適化問題をテスト問題として用いた．テスト関数は以下の 30 次元 Ridge 関数最適化問題を用いた．

$$f = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2 , (-100 \leq x_j \leq 100) \quad (1)$$

ただし，Ridge 関数最適化だけでは，1 個体の適応度計算はごくわずかであるため，EC の工学的最適化問題を想定した負荷として，一定量の LU 分解の計算を適応度計算の後に行う．LU 分解は，スパコンの性能テストでも利用されており，かつ行列の大きさを調整することで，容易に工学的最適化問題に相当する計算粒度を作り出すことができる．本稿では，中粒度の計算粒度として，表 1 のマスタである M1 で計算時間が約 100[ms]，約 500[ms]，約 1,000[ms] の 3 種類になるよう行列の大きさを調節した．

EC における同期処理と通信遅延の問題点を解消することに注目しているため，1 回の試行で 1 万回の通信を行うこととし，(30＋70)-ES を 100 世代まで実験を行った．100 世代の計算が終わるまでの総実行時間と同期待ち時間と通信時間を計測し，総実行時間から同期待ち時間と通信時間を除いた時間を計算時間とする．ここで，同期待ち時間はマスタのキュー Q にタスクがなく，WorkerThread がタスク待ち状態になってから，すべての WorkerThread に対してタスクの計算結果が返信されるまでの時間とする．ただし，RR と R3Q の場合は，同期待ちの際に複製タスクを転送するため，最初の複製タスクを転送してから，すべてのタスクが返信されるまでの時間とする．通信時間はワーカがタスクを実行していない時間のワーカ 1 台あたりの平均時間とする☆．そして，古典的手法である WQ，同期時間減少に適用した RR，通信時間減少に

☆ 複製を行う RR と R3Q では，同期待ちの際にも複製タスクを転送するため，同期待ち時間と通信時間が多少重複するが総量が少ないため，本質的な結果には影響を及ぼさない．

表 1 本実験に使用したコンピュータのスペックとタスクの計算時間（粒度=100，500，1,000[ms]）
Table 1 Average calculation time per one task (computational granularity=100, 500, 1,000) and performance ratio of each machine when the performance of a master machine (PentiumIII 800 MHz) is 1.0.

	Granularity	100[ms]	100[ms]	500[ms]	500[ms]	1,000[ms]	1,000[ms]
No.	Machines	time[ms]	speed	time[ms]	speed	time[ms]	speed
M1	PentiumIII 800 MHz	101	1.00	506	1.00	1,000	1.00
L1	Pentium4 2.4 GHz	42	2.40	175	2.89	229	3.37
L2	PentiumIII 800 MHz	95	1.06	506	1.00	996	1.01
L3	PentiumIII 550 MHz	148	0.68	717	0.71	1,300	0.77
L4	PentiumIII 500 MHz	163	0.62	783	0.65	1,395	0.72
L5	Celeron 466 MHz	184	0.55	1,204	0.42	2,477	0.41
L6	PentiumII 300 MHz	275	0.37	1,328	0.38	2,365	0.43
R1	Xeon 2.4 GHz	41	2.46	176	2.88	299	3.37
W1	Xeon 3.0 GHz	37	2.73	188	2.69	335	3.01
W2	Xeon 2.4 GHz	41	2.46	175	2.89	301	3.35
W3	Athlon MP 2400+	37	2.89	151	3.35	258	3.90
Total		-	**17.23**	-	**18.85**	-	**21.33**

提案した RWQ，RWQ と RR を併用した R3Q の 4 種類の手法を比較することで，R3Q の有効性を検証する．なお，実験は独立に 10 回試行し，その平均時間により評価を行った．

5.3 LAN 内における実験結果

最初に，通信の不確定要素の少ない LAN 環境のみで実験を行った．計算機は表 1 に示す神戸大学にある合計 7 台を用いた．M1 をマスタ，L1–L6 と M1 自身の合計 7 台をワーカとし，マスタは直接ワーカと通信できる．ここで，仮想的に WAN 環境を実現するため，通信遅延時間として約 10[ms]，約 30[ms]，約 50[ms] の 3 種類を付加した．これは神戸–信州大学間の通信遅延時間を基に設定し，スレッドはデータの送受信の前後に Sleep してからデータを転送する．

各計算粒度ならびに各通信遅延時間の設定の違いによる結果を**図 5** に示す．図 5 より，RR は WQ に比べ同期待ち時間を減少させ，総実行時間が減少していることが分かる．しかし，WQ と RR とも通信時間に中粒度タスクゆえの大きな時間的損失が見られる．これに対し，RWQ は通信時間を大きく減少させるが，同期待ち時間が増加している．計算粒度が細かい場合は，同期待ち時間が少ないため，RWQ は RR より総実行時間が短いが，計算粒度が粗い場合は，RWQ は総実行時間が長くなる．R3Q は RWQ と同様，通信時間を大きく減少させ，さらに，RWQ の同期待ち時間も減少させることで，結果的にすべての設定で総実行時間が最も短くなっている．

さらに，R3Q のグローバルな Grid 環境における通信遅延時間の関係を考察するため，横軸に通信遅延時間，縦軸に実行時間をとったグラフを**図 6** に示す．図 6 より，すべての計算粒度において，WQ と RR は通信遅延時間が大きくなるにつれ，総実行時間が増加することが分かる．これに対し，RWQ，R3Q は通信遅延時間に左右されにくい．特に，R3Q はどの計算粒度でも他の手法よりも総実行時間が減少している．これより，R3Q では WAN 環境のような通信遅延時間が大きい環境においても，通信遅延時間を意識することなく一定の効果が得られるといえる．

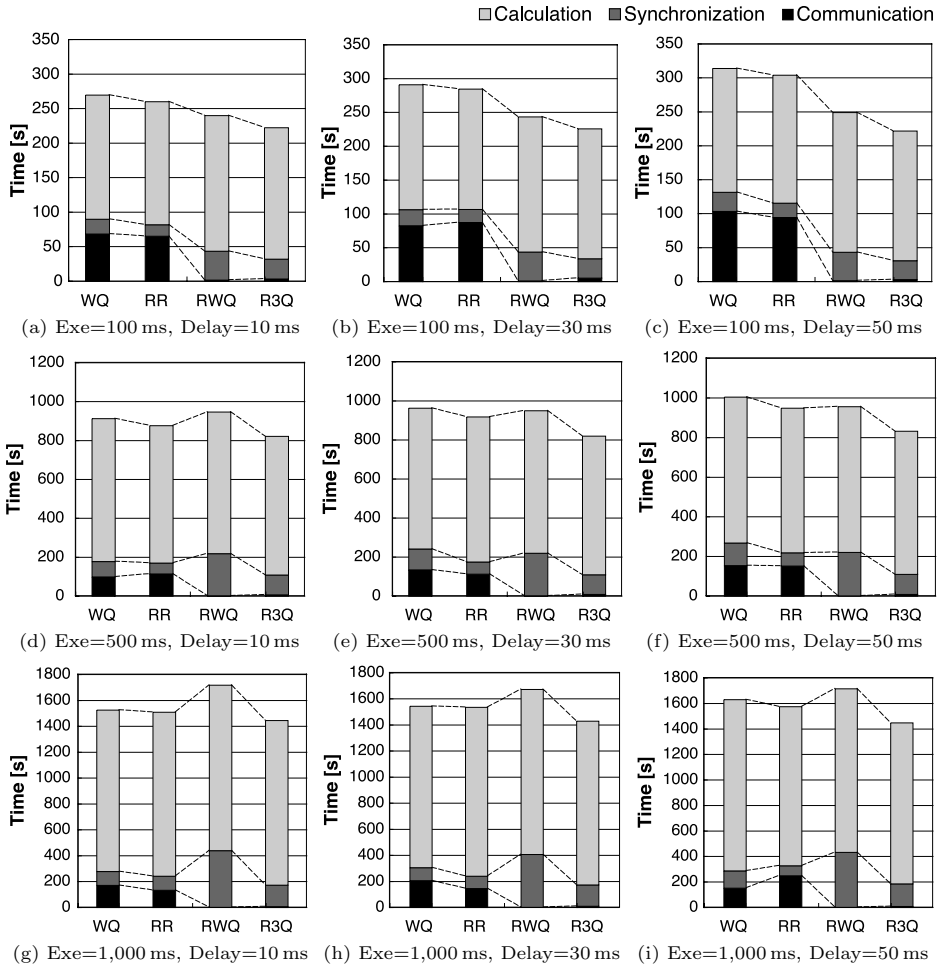
5.4 WAN 環境における検証

次に，スケジューリング手法の比較用に通信時間を一定にした仮想的な LAN 環境だけでなく，通信時間が変動する WAN 環境での各スケジューリング手法の総実行時間を，神戸大学と信州大学のすべての計算機を用いて計測した．

各計算粒度の設定の違いによる結果を**図 7** に示す．LAN 環境と同様，RR は WQ に比べ同期待ち時間を減少させるが，通信時間が増加する．これは WAN 環境が階層構造を持つため，タスクが完了しているワーカに強制終了命令が送られるような命令の行き違いが起こりやすくなり，ワーカにタスクが割り与えられない場合があったためと考えられる．また，計算時間の減少も見られる．これは WAN 環境の計算機が Dual CPU であったため，シングル CPU と仮定した 1 つのタスク実行と強制終了の処理が並列に行えたためと考えられる．その他は LAN 環境同様，RWQ は通信時間を減少させるが，同期待ち時間が増加する．そして，R3Q は通信時間と RWQ の同期待ち時間を減少させ，最も総実行時間が短い．

5.5 並列化効率の検証

本システムおよび R3Q の総実行時間の短縮効果を並列化効率の観点から検証する．**表 2** にすべてのタスクを M1 の計算機単体で計算した場合の総実行時

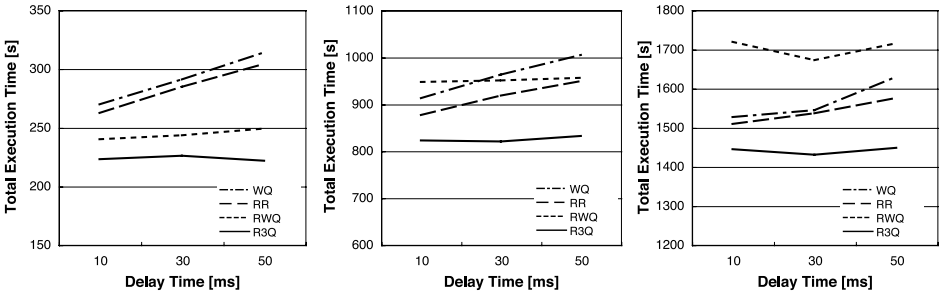


(a) Exe=100 ms, Delay=10 ms (b) Exe=100 ms, Delay=30 ms (c) Exe=100 ms, Delay=50 ms

(d) Exe=500 ms, Delay=10 ms (e) Exe=500 ms, Delay=30 ms (f) Exe=500 ms, Delay=50 ms

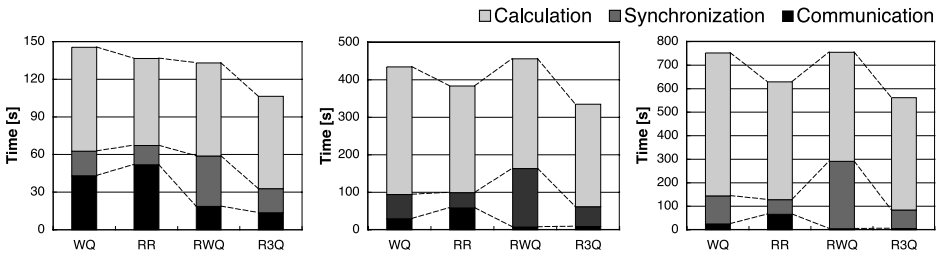
(g) Exe=1,000 ms, Delay=10 ms (h) Exe=1,000 ms, Delay=30 ms (i) Exe=1,000 ms, Delay=50 ms

図 5 各計算粒度と通信遅延時間における WQ，RR，RWQ，R3Q の実行時間計測結果
Fig. 5 Execution time of WQ, RR, RWQ and R3Q for each computational granularity (Exe) and communication delay (Delay) time.



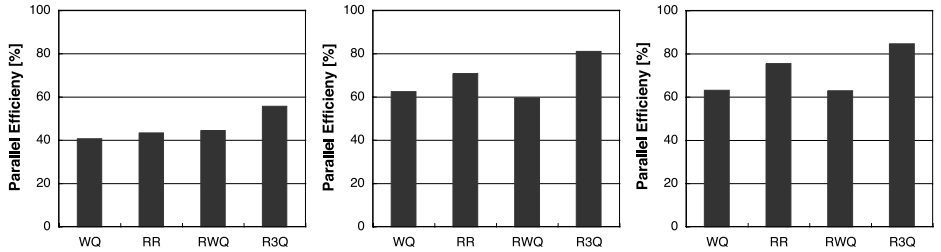
(a) Exe=100 ms (b) Exe=500 ms (c) Exe=1,000 ms

図 6 各通信遅延時間における WQ，RR，RWQ，R3Q の実行時間の比較
Fig. 6 Comparison of total execution time among WQ, RR, RWQ, and R3Q for each communication delay time (Delay) and each computational granularity (Exe).



(a) Exe=100 ms (b) Exe=500 ms (c) Exe=1,000 ms

図 7 WAN 環境での各計算粒度における WQ，RR，RWQ，R3Q の実行時間計測結果
Fig. 7 Execution time of WQ, RR, RWQ and R3Q for each computational granularity (Exe) in WAN environment.



(a) Exe=100 ms (b) Exe=500 ms (c) Exe=1,000 ms

図 8 WQ，RR，RWQ，R3Q の並列化効率
Fig. 8 Comparison of parallel efficiency among WQ, RR, RWQ and R3Q.

表 2 マスタ計算機の総実行時間とすべての計算機を用いた最小実行時間
Table 2 Todal execution time of all tasks on a master (PentiumIII 800 MHz) and a lower bound of ideal execution time on our grid.

computational granularity [ms]	total computational amount [s]	lower bound of ideal parallel time [s]
100	1,021	59
500	5,124	271
1,000	10,139	475

間と，表 1 の最大速度上昇比を基に計算したすべての計算機を利用した場合の最小実行時間（理想的並列実行時間の理論値，OPT と書く）の下界を示す．この下界を LB，スケジューリングアルゴリズム A を用いた場合の総実行時間を E とすると，並列化効率 $Efficiency$ の下界は以下の式で求められる．

$$efficiency = \frac{OPT(\sigma)}{E[A(\sigma)]} \times 100 \geq \frac{LB}{E[A(\sigma)]} \times 100[\%] \quad (2)$$

ここで，σ はすべてのタスク集合を表す．

式 (2) から計算した，各計算粒度の設定の違いによる並列化効率を**図 8** に示す．図 8 より，計算粒度が細かい場合は，それほど大きな並列化効率は得られていないことが分かる．これは SOAP のオーバヘッドが起因していると考えられる．この実験では，転送データが個体の遺伝子情報であり，double 型の配列を 30 個分転送する．そのため，SOAP メッセージの解析時間に約 20[ms] 程度かかり，計算粒度が 100[ms] の場合，少なくとも 20%の損失が存在する．また，WQ と R3Q を比べると，どの計算粒度においても 20%程度の改善効果が見られ，計算粒度が 500[ms] 以上で 80%以上の高い並列化効率が得られた．

6. おわりに

進化型計算の隔世代交替モデルにおいて，各個体の適応度計算を分散計算させて Grid に実装する際に生じる問題に対し，Grid タスクスケジューリングの立場から対処した．特に，非予測型動的スケジューリングの観点から，同期処理の問題に対し RR を適用し，通信時間の問題に対し RR の複製・強制終了を考慮した RWQ を提案し，RR と RWQ を併用する R3Q を提案した．計算粒度が一定のテスト問題において，WQ，RR，RWQ，そして R3Q の比較実験を行った．結果より，R3Q は RR の同期待ち時間の減少と RWQ の通信時間の減少の双方の利点を有効に利用することで，総実行時間を減少させ，さらに総実行時間が通信遅延時間に左右されないことを確認し，高い並列化効率が

得られることを確認した．

今後は，複数大学をつないだ WAN 環境におけるグローバル Grid において，大規模マルチロボット問題を解き，R3Q の有効性を実証する．

謝辞 本研究は科学研究費補助金（16016262），（17500144），（17700233），21COE の研究助成を受けたことを付記し，謝意を表す．

参 考 文 献

1) Castillo, P.A., Arenas, M.G., Castellano, J.G., Merelo, J.J., Rivas, V.M. and Romero, G.: Optimisation of Multilayer Perceptrons Using a Distributed Evolutionary Algorithm with SOAP, *The 7th International Conference on Parallel Problem Solving from Nature* (*PPSN VII*), Lecture Notes in Computer Science, Vol.2439, pp.676–685, ISSN:0302-9743 (2002)
2) Foster, I. and Kesselman, C.: *The Grid: Blueprint for a new computing infrastructure*, Morgan Kaufmann Publishers, CA (1999)
3) Fujimoto, N. and Hagihara, K.: Near-Optimal Dynamic Task Scheduling of Independent Coarse-Grained Tasks onto a Computational Grid, *The 32nd Annual International Conference on Parallel Processing* (*ICPP'03*), pp.391–398, IEEE Press (2003)
4) Fujimoto, N. and Hagihara, K.: A Comparison among Grid Scheduling Algorithms for Independent Coarse-Grained Tasks, *Proc. SAINT2004 Workshop on High Performance Grid Computing and Networking*, pp.674–680 (2004)
5) Fujimoto, N. and Hagihara, K.: Toward Performance Guarantee of Dynamic Task Scheduling of a Parameter Sweep Application onto a Computational Grid, *High-Performance Computing Paradigm and Infrastructure*, pp.333–348, John Wiley & Sons, Inc. (2005)
6) 藤本典幸，萩原兼一：グリッド上でのパラメータ・スウィープ計算を対象として消費余剰計算力のをねらった動的タスクスケジューリングのための近似アルゴリズム，情報処理学会論文誌：数理モデル化と応用，Vol.46, No.SIG10 (TOM 12), pp.1–9 (2005)
7) Graham, R.L.: Bounds for certain multiprocessing anomalies, *Bell System Technical Journal*, Vol.45, pp.1563–1581 (1966)
8) Jing, T., Lim, M.H. and Ong, Y.S.: A Parallel Hybrid GA for Combinatorial Optimization, *Congress on Evolutionary Computing* (*CEC2003*), pp.1895–1902 (2003)
9) 小野 功，水口尚亮，中島直敏，小野典彦，中田秀基，松岡 聡，関口智嗣，楯 真一：Ninf-1/Ninf-G を用いた NMR 蛋白質立体構造決定のための遺伝アルゴリズムのグリッド化，先進的計算基盤システムシンポジウム SACSIS 2005, No.5, pp.143–151 (2005)
10) Paranhos, D., Cirne, W. and Brasileiro, F.: Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids, *Lecture Notes in Computer Science*, Vol.2790, pp.169–180, Springer-Verlag (2003)
11) 染谷博司：グリッド環境に適した遺伝的アルゴリズムによる最適化，統計数理，Vol.52, No.2, pp.381–391 (2004)
12) Spears, W.M., De Jong, K.A., Bäck, T., Fogel, D.B. and De Garis, H.: An Overview of Evolutionary Computation, *Proc. 1993 European Conference on Machine Learning*, pp.442–459 (1993)
13) Tanimura, Y., Hiroyasu, T., Miki, M. and Aoi, K.: The System for Evolutionary Computing on the Computational Grid, *IASTED 14th International Conference on Parallel and Distributed Computing and Systems*, pp.39–44 (2002)

(平成 18 年 1 月 26 日受付)
(平成 18 年 5 月 24 日採録)

松浦 芳樹

2004 年神戸大学工学部機械工学科卒業．2006 年同大学大学院自然科学研究科博士前期課程修了．同年（株）日立製作所システム開発研究所入社．進化型計算のためのグリッド計算に興味を持つ．

大倉 和博

1988 年北海道大学工学部精密工学科卒業．1990 年同大学大学院工学研究科情報工学専攻修士課程修了．同年（株）富士通研究所入社．1993 年神戸大学工学部助手．1998 年より 1 年間英国サセックス大学客員研究員．2000 年神戸大学工学部助教授．2006 年より広島大学大学院工学研究科教授．博士（工学）．EC, RL, AL, CAD/CAM, 生産システム等に興味を持つ．

松村 嘉之（正会員）

1998 年神戸大学工学部機械工学科卒業．2002 年同大学大学院自然科学研究科博士後期課程修了．博士（工学）．2000 年日本学術振興会特別研究員（DC1）．2002 年同研究員（PD）．同年東京大学客員研究員，英国バーミンガム大学客員研究員．2003 年より信州大学繊維学部講師．同年英国バーミンガム大学名誉研究員．EC，EANN，ER，Grid 等に興味を持つ．

藤本 典幸（正会員）

1992 年大阪大学基礎工学部情報工学科卒業．1994 年同大学大学院基礎工学研究科博士前期課程修了．1997 年同大学院基礎工学研究科博士後期課程単位取得退学．同年大阪大学大学院基礎工学研究科助手．2002 年より大阪大学大学院情報科学研究科助教授．工学博士．並列処理，組合せ最適化，近似アルゴリズム等に興味を持つ．

萩原 兼一（正会員）

1974 年大阪大学基礎工学部情報工学科卒業．1979 年同大学大学院基礎工学研究科博士課程修了．工学博士．同大学基礎工学部助手，講師，助教授を経て，1993 年奈良先端科学技術大学院大学教授．1994 年より大阪大学教授．1992 年より 1 年間文部省在外研究員（米国メリーランド大学）．現在，並列処理の基礎および応用に興味を持っている．