計算機アーキテクチャ　147−4
ハイパフォーマンス
コンピューティング　89−4
（2002．3．7）

メタコンピューティング環境における対話型アプリケーションHMMerのスケジューリング

池辺貞郎[1)]　中西恒夫[1)]　福田晃[2)]

[1)] 奈良先端科学技術大学院大学 情報科学研究科
[2)] 九州大学大学院システム情報科学研究院 情報工学部門

概要

分子生物学の分野では，蛋白質のデータを分類，整理する目的で，Profile HMM を用いるアプリケーション HMMer が広く用いられている．しかし，HMMer は実行時間を要するアプリケーションであり，その高速化は重要なテーマである．本稿では，メタコンピューティング環境を用いてHMMer アプリケーションを高速化する．メタコンピューティング環境においては計算機資源の割当や通信の遅延が変動し，その予測が困難であるため，従来のスケジューリング手法を用いた場合に十分な並列度が得られない．提案するスケジューリング手法は複数のプロセスに同一の仕事を割り当てることによりこの問題を回避する．実装による評価の結果，提案手法を用いることによって，セルフスケジューリングを用いた場合と比較して，試行による実行時間の変動が少なく，実行時間においても約 36%の改善がみられた．

A Scheduling Algorithm of an Interactive Application for Metacomputing Environment

Sadao Ikebe[1)]　Tsuneo Nakanishi[1)]　Akira Fukuda[2)]

[1)] Graduate School of Information Science
Nara Institute of Science and Technology
[2)] Graduate School of Information Science and Electrical Engineering
Kyushu University

Abstract

In the field of molecular biology, HMMer is a widely-used software for protein sequence analysis. Since it requires a lot of computational time, improvement in execution time is seriously needed. The purpose of this paper is to accelarate HMMer using metacomputing environment. This paper proposes a scheduling algorithm that manages some unpredictable characteristics degrading execution efficiency in the metacomputing environment and implements a parallelized version of HMMer. Experiments are performed in a simulated Grid environment with three clusters and grid-enabled schedulers for each site. The results demonstrate that the proposed version of HMMer runs 36% faster than the self-scheduled one.

1　はじめに

近年，ポストゲノム情報科学の分野では，大量にある蛋白質のデータベースを用いた構造や機能の予測が重要なテーマとなっている．これに向けて，現在，大量にある蛋白質のデータを分類，整理して，その中から法則性を見いだすというアプローチがさかんに行われている．

このとき，いくつかのアプリケーションを用いることになるが，その中でも，蛋白質のデータを分類，整理する目的で，HMMer [1] と呼ばれるアプリケーションが広く使われている．しかし，HMMer は実行

時間のかかることが問題となっており，現在は研究者自身の計算機でHMMerを実行することは非常に難しい．このような実行時間の長いアプリケーションにおいては，高性能計算機にインストールし，Webによるインターフェースを介して利用するといった利用形態が主流になっている [2]．しかしながら，Webによるインターフェースは，作業効率や機能の面で問題が多いため，作業効率や機能の面から考えると研究者自身の計算機でHMMerを実行することが望まれ，HMMerの高速化は重要なテーマである．

HMMerにおいては，モチーフであるProfile HMM (蛋白質の共通点を隠れマルコフモデルでモデル化したもの) [3] [4] と，系列をマッチングする際に，Viterbiアルゴリズムが使用されるが，Viterbiアルゴリズムをはじめとする動的計画法のアルゴリズムは，一般に計算量と計算結果の精度が比例する．このため，本研究では，計算量を減らさずとも実行時間を改善できるよう，並列計算による高速化を検討している．

本稿では，メタコンピューティング環境を用いたHMMerの高速化について検討する．メタコンピューティング環境とは，分散コンピューティングの一形態であり，ノードはヘテロな計算機から構成されている．この環境を対象としてHMMerの高速化を図る．

本稿の構成は以下のとおりである．第2節では，HMMerアプリケーションを分析し，内部構造，および処理内容を並列／分散処理の観点から分析し，高速化の方針を明確にする．第3節では，分析結果を踏まえて，並列／分散化の手法を提案する．第4節では，提案手法を用いて並列化したHMMerの実装を行い，従来の手法と比較評価を行う．第5節では関連する研究について述べ，第6節で結論を述べる．

2 HMMerアプリケーション

HMMerでは，蛋白質の共通点をモデル化したものであるProfile HMMという基準を用い，実験データの蛋白質シーケンスとProfile HMM間でマッチングをとってスコアづけを行いファミリに分類する，いわゆるモチーフ検索に使われるアプリケーションである．

HMMerの実行時間は，Pentium III 1GHz SMPの個人用の計算機を用いてPfam [5] データベースと単一の系列のマッチングを行う場合において，約15分程であるが，試行錯誤を何度も繰り返す使われ方をされるため，実行時間を大幅に短縮する必要がある．

2.1 内部構造

HMMerの中での，データベースの検索を行う部分は，モチーフとクエリーのマッチングを行うという，依存関係のない多数のジョブを並行に実行している．このような処理形態は，Parameter Sweepアプリケーションと呼ばれ，自明な並列性をもつ．そのため並列化自体は容易であるが，メタコンピューティング環境上で効率よく実行するためには，計算機資源，通信路の遅延などの環境の変化への対応等を考慮したスケジューリングが必要となる．

2.2 計算量

各ジョブの行う処理は，Viterbiアルゴリズムによる単一のマッチング処理であるため，計算量は事前に予測することができる．

計算量の少ないジョブが実行される場合は，計算結果として単位時間当たりに通信路に流れるデータ量が増加し通信路に与える負荷が大きくなるが，ジョブの実行時間は短く，そのためセルフスケジューリングを使用した場合に各ノードでの計算の終了時刻を揃えやすく，並列度が上げやすい．

NR, Pfamなどのデータベースを検索する場合においては，計算量の小さいジョブ (e.g. モデル長が短い) の実行時間は小さく，計算結果の到着までの通信路の遅延が大部分を占めることになる．

2.3 並列・分散処理の観点からの分析

HMMerで使用されているセルフスケジューリングは，計算機資源等の変化がほとんどないホモジニアスな並列計算機においては有効に機能する．しかし，メタコンピューティング環境においては，計算機の利用率や通信路の遅延など予測不可能な要素があるため，各ノードにおけるタスクの終了時刻にずれが生じ，並列度が下がるため，有効ではない．

また，各ノードは停止したり，あるいは停止しないとしても外部のサイトから投入されたジョブに計算機資源を割り当てる優先順位が低くされるなど，サイト毎のスケジューラのポリシーによって突然，遠隔ノードにリクエストしたジョブにCPU資源が割り当てられなくなることがある．実行時間が短いアプリケーションにおいては，各ノードでのタスクの終了時刻のずれが実行時間に与える影響が非常に大きく，そのためスケジューリングの重要度が高い．

以上のような状況においても，大幅に実行時間が延びることなく，ある程度の実行時間で処理を完了することが本研究の目的となる．

3 スケジューリング法

本節では，HMMerアプリケーションにおけるスケジューリング法を提案する．アプリケーションで

は，一つの親プロセスが，子プロセスに対してジョブの配分を行い，子プロセスから返却された計算結果を集積している．このモデルは並列計算においても有効であり，本節においてもこのモデルを用いた場合のスケジューリング手法を提案する．

3.1 用語の定義

“ジョブ j” とは，計算の最小単位であり，本スケジューリング法ではこの単位ごとに，計算の割り当てを行う．M を全マッチング処理の数としたとき，$0 \leq j < M$ となる．

“親プロセス” は，子プロセスに対してデータベース検索のジョブを配分し，子プロセスから返却された計算結果を集積する．

“子プロセス c_i” は，親プロセスから配分されたジョブを実行し，計算結果を親プロセスに返却する．全子プロセス数を N としたとき，$0 \leq i < N$ となる．

“到着時刻 t_{ij}” とは，ある一つの子プロセス c_i から，親プロセスに一つのジョブ j の計算結果が返却された時刻である．プロセス c_i から親プロセスに到着した最後のジョブを k としたとき，$t_i = t_{ik}$ とする．

“到着間隔 d_i” とは，ある一つの子プロセス c_i から親プロセスに一つのジョブの計算結果が到着した時刻と，同じ子プロセス c_i に割り当てられた次のジョブの計算結果の到着時刻との間隔を指す．

“期待到着間隔 $E(d_i)$” は，ある子プロセス c_i における，到着間隔の期待値を指す．

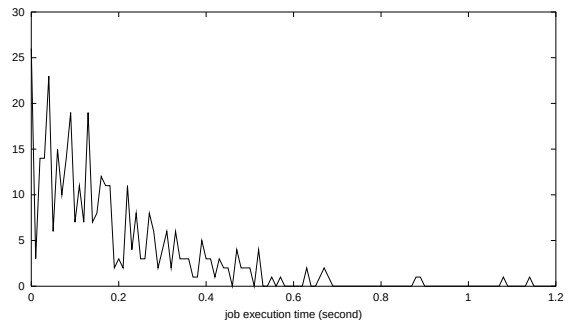

図 1: ある子プロセスにおける計算結果の到着間隔 (度数分布)

3.2 スケジューリング法の概要

本節において提案するスケジューリング手法の基本的な概念は，親プロセスからの視点で，子プロセスの振舞を観測して統計を作成し，統計をもとにジョブの実行時間を推測し，推測した実行時間をもとにスケジューリングを行うというものである．

図 1 に，ある一つの子プロセスに等しい計算量のジョブを複数投入した場合の到着間隔を計測し，その度数分布を示す．図 1 に見られるように，子プロセスから返却される計算結果の到着間隔には大きなばらつきがある．

到着間隔が 0 の事象が多く見られるが，実際にはジョブの計算時間が 0 ということはなく，通信路の事情により，計算結果が連続して到着したものと思われる．また，到着間隔が非常に長い事象も確認できた．

こうした実験結果から，本アプリケーションに関しては計算機のスケジュール，あるいは通信路の遅延が到着間隔に与える影響が大きく，ジョブの計算量が等しくても，計算結果の到着時刻は大幅に変動することがわかる．このばらつきは，子プロセスに割り当てられた計算機資源の変動であったり通信路の遅延であったりすると考えられるが，広域ネットワークを用いるメタコンピューティング環境においては，計算機資源の変動や通信路の遅延などの事象は予測が困難である．

本稿では，これらの予測が困難な要因に対処するため，同一のジョブを複数のノードに割り当て，先に返却された計算結果を採用することによって，最悪の場合のジョブ実行時間を短縮するという考え方に基づくスケジューリング手法を提案する．

3.3 危険度

本節では，一つのプロセスにおけるジョブの到着時刻の最悪値を考える．メタコンピューティング環境においては，通信路の断線，計算機の停止といったことも起こりうるため，確実に保証できる最悪値は考えられないが，プロセス c_i におけるジョブ j の計算結果の到着時刻 $t_{ij} < T$ となる確率を考えることは可能である．

上記のように，あるプロセスの最後のジョブの到着時刻が，時刻 T 以前であるという仮定を行う際に，この仮定が誤りである確率という意味での “危険度” という概念を導入する．

定義 任意の危険度 $P(0 < P < 1)$ を与えるとき，プロセス c_i の到着間隔の統計情報をもとに，プロセス c_i に割り当てられたジョブの到着間隔は，確率 P で，t より大きい値になると推定できる．このとき，t を，“プロセス c_i における，危険度 P での予測到着間隔 $E_P(d_i)$” とする．(i.e. プロセス c_i の到着間隔 d_i は，確率 $1-P$ で，t よりも小さい値をとる．)

定義 プロセス c_i に割り当てられている残りのジョブ数を n として，$T_i(P) = t_i + n \times E_P(d_i)$ に

よって定義される $T_i(P)$ を，“プロセス c_i における，危険度 P での予測完了時刻” とする．

定義 $T(P) = \max_i(T_i(P))$ によって定義される $T(P)$ を “危険度 P での予測完了時刻” とする．

実際には，危険度 P を充分小さな値にして上記の予測を行い，下記のスケジューリングアルゴリズムの指標とする．

3.4 アルゴリズム

提案するスケジューリング手法は，充分小さな P において，前節で述べた危険度 P での予測完了時刻を最小にして，アプリケーションの終了時刻を保証することが目標である．しかし，危険度 P が小さい場合の予測到着間隔は，通常の到着間隔とはかけ離れた値であり，危険度 P での到着間隔をスケジューリングの指標として用いると十分な並列度が得られない可能性がある．

そのため以下では，まず期待到着間隔 $E(d_i)$ を用いてスケジューリングを行う．その後，同一のジョブ j を複数のプロセスに割り当てることによって危険度 P での予測完了時刻を短縮するというアプローチでメタコンピューティング環境特有の予測困難な要因に対処するという手法を提案する．

ジョブの冗長投入

プロセス c_x に割り当てられている最後のジョブを j とする．c_x における危険度 P での予測完了時刻 $T_x(P)$ は，定義より $T_x(P) = t_x + n_x \times E_P(d_x)$ であるが，ここで，ジョブ j を c_y にも重複して投入することを考える．すると，$T'_x(P) = t_x + (n_x - 1) \times E_P(d_x) + E_A(d_x)$，$T'_y(P) = t_y + n_y \times E_P(d_y) + E_B(d_y)$ となる．ここで，プロセス c_x, c_y に割り当てられている残りのジョブ数はそれぞれ n_x, n_y であり，A および B は，それぞれ $AB = P$ を満たす危険度である．

$A, B, P < 1$ であるので，一般に $E_P(d_x) \leq E_A(d_x)$ となる．そのため，$T'_x(P) < T_x(P)$ が成立する．このように，プロセス c_x の最後のジョブをプロセス c_y にも重複して投入する操作により，$T_x(P)$ を短縮できる可能性がある．

3.5 アルゴリズムの詳細

提案スケジューリング手法のアルゴリズムを以下に示す:

1. 任意の危険度 P を設定する．
2. 各プロセス毎に，ジョブの計算結果の到着間隔 d_i を収集する．(4, 5 の間も収集は継続する)
3. 各プロセス毎の d_i の統計情報を用いて，$E(d_i)$ を計算する．
4. (スケジューリングイベント) プロセス c_i から親プロセスに到着した最後のジョブを j とし，c_i に割り当てるジョブ数を n として，$T_i = t_{ij} + n \times E(d_i)$ を計算する．
5. 各プロセスに関して，4. で計算した T_i の最大値と最小値の差が最小になるようにジョブを投入する．
6. 4 および 5 のスケジューリングイベントを適宜実行し，残りのジョブ数が十分少なくなった時点で，以下の手法でのスケジューリングに切り替える．
7. 最大の $T_i(P)$ をとるプロセス c_x と，最小の $T_i(P)$ をとるプロセス c_y を考える．プロセス c_x に割り当てられている最後のジョブを c_y に重複して割り当てる．
8. T(P) が改善されなくなるまで，7. の操作を繰り返す．
9. 残りのジョブの完了を待つ．

4 実装と評価

実装は，広域計算システム型のメタコンピューティング環境として，最も標準的な，Globus [6] 上での動作をターゲットとた．実際には，Globus のユーザー認証や通信機構を利用した，MPICH-G2 [7] を用いて実装を行った．

Pfam データベースを検索するために用いる，`hmmpfam` コマンドを，提案スケジューリング手法を用いて MPICH-G2 上で実装し，評価を行う．

また，従来の `hmmpfam` コマンドは Globus 環境上で動作しないので，従来の `hmmpfam` コマンドと同様のスケジューリング (セルフスケジューリング) を用いたものを MPICH-G2 上で実装し，比較対象とする．

4.1 実験環境

評価はシミュレーションで行っている．シミュレーションでは，メッセージパッシング機構を用いた並列計算機が動作している 3 つのサイトで構成される，Grid 環境を想定している．

実際に Grid 環境を構築し，時間の計測を実時間で行うが，外乱のほとんどないないクラスタ PC を

用いたため，各計算機資源および通信路資源に，以下に示す負荷をかけて計測を行った．

- ローカルサイト (8CPU)，通信，CPU に負荷等は与えない．
- リモートサイト (4CPU)，jobmanager-condor が使用されていることを想定，2–4 秒ごとに，計算機資源の割り当てを 0.1 − 2.0 秒間停止する．通信に負荷は与えない．
- リモートサイト (8CPU)，バックグラウンドで他のジョブを実行している．CPU 負荷は 1.0 程度で安定しているが，通信において遅延がかかる．通信における遅延は，ある実在のサイトまでの RTT を計測し，その結果を考慮して決定している．

4.2 実験条件

単一の蛋白質系列 ($L = 1405$) と Pfam DB の間でマッチングを行う試行を，セルフスケジューリングと，提案手法それぞれを用いる場合において 70 回ずつ行い，統計をとる．(危険度 $P = 0.01$ とした．)

4.3 実験結果

表 1 に，前節で述べた条件での各試行の実行時間を平均したものを示す．

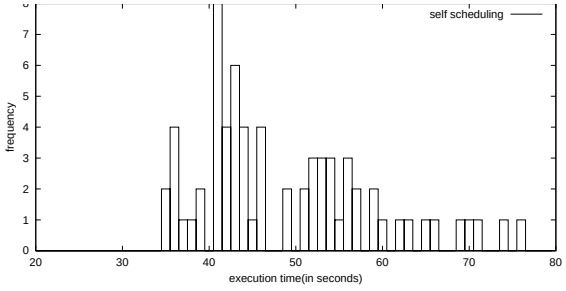

図 2: セルフスケジューリングを用いた場合の実行時間 (度数分布)

また，図 2 および図 3 に，各試行の実行時間の度数分布を示す．

表 1 から，提案手法では，セルフスケジューリングと比較して，平均実行時間において約 36%の改善がみられる．また，最悪実行時間においても，52%の改善など，著しい改善がみられる．

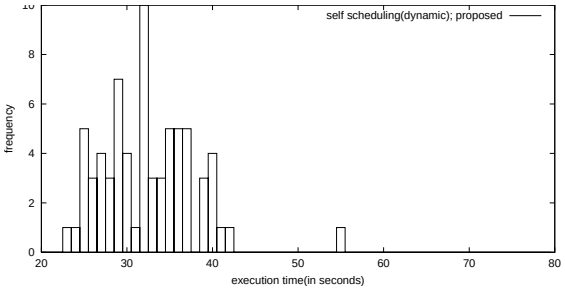

図 3: 提案手法を用いた場合の度数分布 (度数分布)

ジョブの冗長投入を行うことによって，実行時間の標準偏差が小さくなることがわかる．このことから，提案手法を用いることにより計算機の利用率や通信路の遅延など，予測不可能な要素がある状況の中でも，ある程度の実行時間で処理を完了できているといえ，本研究の目的が達成されている．

5 関連する研究

Parameter Sweep アプリケーションのスケジューリングに関するフレームワークが [8] において提案されており，本稿においても参考にしている．同時に提案されているスケジューリング手法に関しては，前提が異なるため採用していないが，提案されているスケジューリング手法を拡張した XSufferage と呼ばれるものが [9] において提案され，よい性能を示すことが報告されている．APST [10] は，Parameter Sweep アプリケーションに特化したスケジューリングを提供するミドルウエアである．

MW [11] は，Master-Worker の形で実行されるアプリケーションをメタコンピューティング環境上で並列化するフレームワークを提供している．

6 まとめと今後の課題

本稿では，HMMer を高速化するためのプラットフォームとして，メタコンピューティング環境を提案した．また，メタコンピューティング環境特有の問題を想定し，この問題を解決するためのスケジューリング手法などを提案した．

計算機の利用率や通信路の遅延など，予測不可能な要素がある状況の中でも，大幅に実行時間が延びることなく，ある程度の時間で実行を完了することを保証できているといえ，本研究の目的が達成されている．

提案手法では，同じジョブを複数のノードに投入

表 1: 実行時間の計測結果

スケジューリング法	平均実行時間 (秒)	最悪実行時間 (秒)	標準偏差
セルフスケジューリング	51.38	88.96	15.46
提案手法	32.93	42.54	5.48

することにより計算量が増加しているが，このことは計算機資源の浪費につながる恐れがある．そのため，提案手法を，計算機資源の浪費の観点からも評価し，計算機資源の浪費を最小限に抑える手法を提案する必要がある．

また，本稿では，唯一の想定 Grid 環境でのみ評価を行ったが，実験結果は Grid 環境の設定条件によって大幅に変わる可能性がある．提案手法の，より正確な評価を行うためには，複数の Grid 環境を想定して実験を行う必要がある．

本稿において提案したスケジューリング手法は，比較的短時間で完了するタスクの終了時刻を揃え，並列度を高める手法であることから，OpenMP(Multi Processing) などによって展開されるループなど，細粒度なタスクのスケジューリングに適用できる可能性がある．近年の広域通信の基幹回線は，密結合な並列計算機などの通信路として使用されている myrinet あるいは Gigabit Ethernet といった通信路の帯域幅を大きく越えており，帯域幅の面では細粒度並列化が充分期待できる．

謝辞

本研究は，文部科学省平成 13 年度科学研究費・特定領域研究 (C)(2),「ゲノム情報学アプリケーションの自己適応的並列化／最適化に関する研究」の助成を受けています．また，本研究を進めるにあたって多くの助言を頂いた産業技術総合研究所生命情報科学研究センターの熊谷俊高氏に深く感謝します．

参考文献

[1] S. R. Eddy: “Profile hidden markov models,” *Bioinformatics*, Vol. 14, pp. 755–763, 1998.

[2] Y. Akiyama, K. Onizuka, T. Noguchi, and M. Ando: “Parallel protein information analysis (papia) system running on a 64-node pc cluster,” *Proc. of the 9th Genome Informatics Workshop (GIW'98), Universal Academy Press*, pp. 131–140, 1998.

[3] S. Eddy: “Hidden markov models and large-scale genome analysis,” *Trans. of the American Crystallographic Association*, 1997.

[4] R. Hughey and A. Krogh: “Hidden markov models for sequence analysis: extension and analysis of basic method.,” *Comp. Appl. BioSci*, Vol. 12, No. 2, pp. 95–108, 1996.

[5] E. Sonnhammer, S. Eddy, E. Birney, A. Bateman, and R. Durbin: “Pfam: multiple sequence alignments and hmm-profiles of protein domains,” *Nucleic Acids Research*, Vol. 26, pp. 320–322, 1998.

[6] I. Foster and C. Kesselman: “Globus: A metacomputing infrastructure toolkit,” *The International Journal of Supercomputer Applications and High Performance Computing*, Vol. 11, No. 2, pp. 115–128, Summer 1997.

[7] I. Foster: “A grid-enabled MPI: Message passing in heterogeneous distributed computing systems,” 1998.

[8] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund: “Dynamic mapping of a class of independent tasks onto heterogeneous computing systems,” *Journal of Parallel and Distributed Computing*, Vol. 59, No. 2, pp. 107–131, 1999.

[9] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman: “Heuristics for scheduling parameter sweep applications in grid environments,” *Heterogeneous Computing Workshop*, pp. 349–363, 2000.

[10] H. Casanova, G. Obertelli, F. Berman, and R. Wolski: “The apples parameter sweep template: User-level middleware for the grid,” *Supercomputing*, Nov. 2000.

[11] J. Goux, S. Kulkani, J. Linderoth, and M. Yoder: “An enabling framework for master-worker applications on the computational grid,” *Proc. of the Ninth IEEE International Symposium on High Performance Distributed Computing*, pp. 43–50, 2000.