2007－HPC－111 (9)
2007／8／1

複数の特性の異なるネットワーク経路を持つ環境における ストリーミングデータ処理のためのタスクスケジューリング

吉永一美
九州工業大学大学院
情報工学研究科

小出 洋
九州工業大学情報工学部
知能情報工学科

概 要

本研究では, 複数の特性が異なるネットワークが存在するような広域並列分散環境上で, ストリーミングデータ処理を含む並列アプリケーションと, 通常のアプリケーションの双方を効率的に実行できるようなタスクスケジューリング手法を提案し, 評価実験を行った. タスクの通信形態に適したネットワークを利用し, ネットワークの負荷変動を考慮した手法を利用することで, 全体の実行時間の短縮が図られ, 動的にネットワーク性能が変動している環境でも, 実行時間のばらつきが抑えられた. この結果はより複雑なタスクスケジューリング手法と組み合わせる場合に有用な特性である.

A New Task Scheduling Method for Processing of Streaming Data in an Environment Including Several Networks Having Different Characteristics

Kazumi YOSHINAGA
Graduate School of Computer
Science and Systems Engineering
Kyushu Institute of Technology

Hiroshi KOIDE
Faculty of Computer Science of
Systems Engineering
Kyushu Institute of Technology

This paper proposed and evaluated a new task scheduling method for parallel and distributed applications in an environment including several networks having different characteristics. The proposed method can schedule both streaming applications and non-streaming applications well at a same time, since it selects the most suitable networks for the communications of tasks and considers the changing loads of the networks. In the experimental results, the proposed method reduces the total execution time of a practical streaming application. The dispersion of the execution time is also suppressed even if the network bandwidth is dynamically changing. This characteristic is very useful when this method combines more complicated task scheduling methods.

1 はじめに

近年, 「マルチメディアデータ処理」や「実験装置から連続的に大量に生成されるデータの解析」など, ストリーミング処理が含まれたアプリケーションに対する要求が高まっている. だが, そういったストリーミング処理が含まれている分散プログラム (以下, ストリーミング型分散プログラム) は, 現在までのところ, 分散コンピューティング環境で効率的な実行を行うことができない.

その最大の理由は, 従来から良く研究されているワークフロー型 [1, 2] やタスクファーミング型の分散プログラム向けのタスクスケジューリング手法 [3] がそのままでは適用できず, 分散プログラミング環境の計算機資源を効率良く割り付けることができないからである.

そこで我々は, ストリーミング型分散プログラムを分散コンピューティング環境上で実行しても, 効率良く動作させることができ, その実行時間を短縮できるタスクスケジューリング手法を研究している [4]. 本稿では, 複数の特性の異なるネットワークが存在するような環境において, ストリーミング型分散プログラムを効率よく動作させることができるタスクスケジューリング手法を提案する. ネットワークには複数の経路が存在するが, 現在一般的に使われているネットワーク経路制御プロトコルでは, 経路を選択する指標として静的に設定されたコストが使用されており, 複数の経路を選択して利用することは難しい. しかしながら,

複数経路表を用いた方法 [5] など, 複数の経路を動的に切り替える経路制御の研究がなされており, 本研究はそのような環境を前提としている.

広域ネットワークの性能はその普及とともに飛躍的に向上しているが, 依然として計算機内部のデータ転送速度と比較するとかなり遅く, タスク間のデータ転送がボトルネックとなり実行速度の低下に直結する. また, ネットワーク性能のばらつきも大きく, 一般に他人と共有された資源であるため, 実効性能が時間的に変動する.

ストリーミング処理を含むタスクは通常のタスクと異なり, データの入力, 処理, 出力を繰り返し行うため, 長時間に渡ってタスク間で大量のデータ転送が発生すると考えられる. 一方通常のタスク処理では入出力は一度だけしか行わず, ストリーミング処理と比較すると通信量は一般的に小さい.

これらの特徴に基づき, タスクに適したネットワークを選択することで, ストリーミング処理タスクおよび通常の処理タスクの双方の実効時間を短くするタスクスケジューリング手法を提案し, 評価を行った.

2 提案するタスクスケジューリング

2.1 設計指針

本タスクスケジューリング手法は, ストリーミング型分散プログラムを分散コンピューティング環境上で効率良く実行することを目的としている. また, ストリーミング処理以外の処理が混在した場合にもそれぞれに適した資源割り当てを行い, 効率的に実行するように設計する.

本研究で対象としている環境は, 異なる特性を持った複数のネットワークが利用できるような広域分散コンピューティング環境である. 分散コンピューティング環境には様々な性能の計算機やネットワークが接続されており, 性能の格差が大きい. また, 複数の他人により共有されることが多いため, 負荷の変動, 特に広域分散コンピューティング環境ではネットワークにおける負荷変動が大きい.

そこで, ネットワーク特性の差異と性能の変動に着目し, スケジューリング手法を設計した.

2.2 ストリーム処理タスクの割り付け

ストリーミング型分散プログラムはストリーミングデータ処理を行うタスクを含んでおり, こうしたタスクを含まない通常のアプリケーションと比較すると, 一般にタスク間で長時間に渡り多くのデータ通信を行うという特徴がある.

この特徴を考慮し, 出来る限りバンド幅の大きなネットワークを利用することでデータ通信を効率良く行うことができ, 効率的な実行が可能になると考えられる.

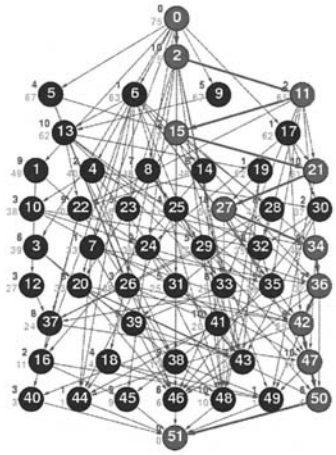

図 1: 分散プログラムのタスクグラフの例.
Fig. 1: An Example of Task Graph.

分散プログラムをタスクグラフ (図 1, [6]) として表すとすると, ストリーミングデータ処理を含むタスクとしては,

1. ストリーミングデータの入力がある
2. ストリーミングデータの出力がある
3. ストリーミングデータの入出力ともに持ち合わせている

の 3 種類がある. 今回提案するタスクスケジューリング手法では, ストリーミングデータの入力があるタスク, つまり 1 と 3 のタスクに着目して, 2.2.1 で述べるような割り付けを行う.

2.2.1 ワーカの選択

スケジューリング手法が実際に実行可能なタスクを割り付ける際, タスク t_x を割り付けるものとすると, t_x に対してストリーミングでデータを送信するタスクを $t_{s_1}, \cdots, t_{s_n}$ とし, それらのタスクが割り付けられたワーカを $w_{s_1}, \cdots, w_{s_n}$ とする. このとき,

$$\min\{bw\_pred(w_{s_1}, w_x), \cdots, bw\_pred(w_{s_n}, w_x),\}$$

の値が最大になる w_x を，タスク t_x の割り付けるべきワーカとする．ここで，$bw\_pred(w_m, w_n)$ は，ワーカ m とワーカ n の間のバンド幅の予測値である．

2.2.2 タスク・マイグレーション

2.2 で述べたように，ストリーミングデータ処理を含むタスクは長期間に渡りデータを流し続ける．広域分散コンピューティング環境の特性上，2.2.1 で決めたワーカが長時間に渡り良いパフォーマンスを維持するとは考え難く，性能が低下した場合の対策を講じる必要がある．

そこで，ネットワーク性能が低下してきた場合にタスクを他のワーカに移動する (マイグレーションする) ことで，性能の低下を回避するようにした．具体的には，入力ストリームの流量を監視しておき，その値が減少してかつそのストリームが利用しているネットワークのバンド幅の測定値が低下していた場合に，そのネットワークが混雑していると判断し，タスクの実行を中断して別のワーカへ移動する．その際にどのワーカにタスクを移動するかについては 2.2.1 で述べた方法を用いる．移動した先のワーカでは，ストリームの再接続など必要な処理を行った後，中断した時点から実行を再開する．

2.3 通常タスクの割り付け

ストリーミングデータ処理を含まないような通常のアプリケーションは，ストリーミング型分散プログラムと比較してデータ通信量が少ない．データ通信に要する時間 t を，バンド幅 bw，通信データサイズ d，遅延 t_l を用いて表すと，

$$t = \frac{d}{bw} + t_l$$

となり，通信するデータのサイズが小さいほどネットワーク遅延の影響を大きく受ける．そこで，通常のタスクを割り付ける際には遅延の小さなネットワークを利用するように配置することで，実行の高速化を図る．

3 評価実験

3.1 実験に用いたハードウェア環境

実験に用いた計算機とネットワークの構成を図 2 に示す．8 台の同一性能の計算機が接続されており，1 台がスケジューラ，残りの 7 台がワーカの役割を果たす．計算機のスペックは表 1 の通りである．

各計算機が利用できるネットワーク経路は 2 種類存在する．経路 A は 100Base-TX のネットワークで，HUB を介して直接通信を行う経路であるため遅延が非常に小さい (0.1ms 程度)．経路 B は 1000Base-T のネットワークであり，ルータを介した通信が行われる．また，途中で通過するネットワークエミュレータにて 10ms の遅延を発生させているため，経路 A と比較するとバンド幅は大きいが遅延も大きいネットワークになっている．

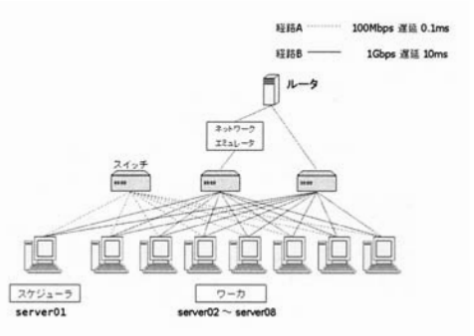

図 2: 実験に用いた計算機とネットワーク．
Fig. 2: The Networks for the Experimentations.

表 1: 実験に用いた計算機のスペック
Table 1: The Specification of machines.

CPU	AMD Athlon™64 X2 3800+
メモリ	1GByte(512Byte × 2) DDR 400
NIC	nVidia Corporation MCP51(100Mbps)
	Intel Corporation 82541PI Gigabit Ethernet Controller(1Gbps) × 2
その他	Java JDK 1.5.0_11
	iperf 2.0.2

本研究が対象にしている環境は，多数の利用者が使用しており，負荷が動的に変動する分散コンピューティング環境である．実験で用いた環境では，iperf を複数起動して通信を行うスクリプトを用いて，他の利用者により負荷がかかったような状態を模擬した．スクリプト内の処理は以下の通りである．

1. 帯域制限するホストを，`server03`〜`server07` からランダムに 2 つ選択する
2. 帯域制限する時間 l_t を，60〜120 秒の間でランダムに設定する
3. 1 で決定した 2 ホスト間で，iperf を用いて経路 B を通る通信を l_t 秒間行う
4. l_t 秒経過後，1 に戻る．

3.2 評価用アプリケーション

3.2.1 ストリーミング型プログラム

ストリーミングアプリケーションとして，現在研究室内で開発している分散型ネットワーク監視システムの一部を用いた．このシステムではネットワークパケットを監視し，セキュリティチェックやコンテンツの分析等を行う．

今回利用したのは，コンテンツのキャッシュを行う分散プログラム部分である．これはセキュリティチェックを行うために必要不可欠な処理であり，ネットワーク利用の効率化を図るためにも有用なものである．

実験に用いたシステムの構成を図 3 に示す．

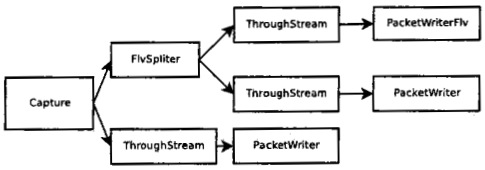

図 3: 評価用アプリケーション．
Fig.3: The Application for an Evaluation.

以下にそれぞれのタスクの簡単な説明を述べる．

Capture タスク： パケットをキャプチャし，HTTP 通信とそれ以外とに分割するタスクである．今回は各実験で同じデータを利用するために，リアルタイムにパケットをキャプチャするのではなく，tcpdump 形式で保存されたデータをファイルから読み込む仕様とし，実験では約 2.7GByte のパケットダンプデータを用いた．なお，このタスクは常に `server02` 上で実行される．

FlvSpliter タスク： HTTP 通信のパケットを受け取り，flv データ (動画ストリームデータ) のパケットとそれ以外のパケットに分割するタスクである．

PacketWriter タスク： パケットを受け取り，シーケンス番号に基づいてデータを再構築し，データをディスクに書き出す．このタスクは `server08` で実行される．

PacketWriterFlv タスク： PacketWriter とほぼ同じだが，こちらは HTTP ヘッダの解析を行い，flv データだけを取り出してディスクに書き出す．このタスクは `server08` 上で実行される．

ThroughStream タスク： 特に何も処理をせずストリームを通すだけのタスクであり，未完成であるファイル内のチェック処理や，動画変換処理などが挿入される部分に代わって配置した．

3.2.2 非ストリーミングプログラム

ストリーミング処理を含まないアプリケーションとして，早稲田大学笠原研究室で配布されている，Standard Task Graph set(STG)[6] を用いた．今回はタスク数が 50 個の STG を 180 種類利用し，タスク間では 10KByte のデータ転送を行うものとした．

3.3 並列分散処理プラットホーム

実験に用いたソフトウェアプラットホームの構成を図 4 に示す．これは大きく分けて，タスクスケジューリングシステムと資源情報サーバ (RIS) [7] のふたつの部分から構成されており，すべて Java を用いて実装されている．各モジュール間の通信には JavaRMI およびソケットライブラリが用いられている．

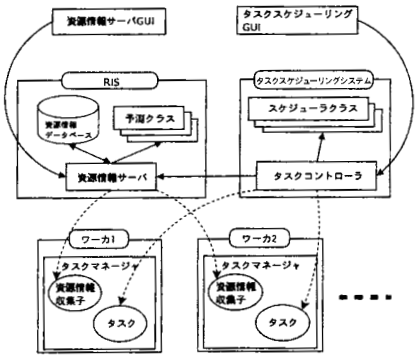

図 4: ソフトウェアプラットホーム．
Fig.4: Our Software Platform.

3.3.1 タスクスケジューリングシステム

タスクスケジューリングシステムは，タスクの組み合わせで定義されたユーザ・プログラムをスケジューラからの指示に基づいてワーカに割り付けることにより，並列分散コンピューティングを実現するものである．システムは，タスクコントローラ，タスクマネージャ，タスクスケジューリング GUI の 3 つのモジュールと，タスクコントローラが使用するスケジューラクラスから構成されている．各コンポーネントの役割は以下の通りである．

タスクコントローラ： ユーザにより投入されたユーザ・プログラムを構成するタスクを，ユーザが指定したスケジューラクラスに基づき，タスクマネージャに割り付ける．

タスクマネージャ： 各ワーカ上で動作し，タスクコントローラにより割り付けられたタスクを実行する．

タスクスケジューリング GUI： ユーザがタスクコントローラを制御するための GUI フロントエンドである．タスクを組み合わせてユーザ・プログラムを構成したり，使用するタスクスケジューラを選択して実行したりできる．スクリーンショットを図 5 に示す．

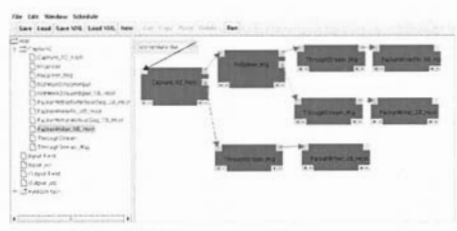

図 5: ユーザプログラムを定義するための GUI.
Fig.5: The GUI for User Programs.

タスクやタスクスケジューラは，あらかじめ用意されている Java で書かれたインターフェースクラスを実装することにより，ユーザが自由に作成することができる．タスク間のデータ交換としては，ひとつのタスクが終了する際に次のタスクに情報が受け渡される (返り値と引数) タイプの他に，タスクが終了せずにストリーミングデータとして次のタスクに情報を受け渡す (ストリーミング) タイプもサポートしている．前者は Java RMI を用いて実装されており，後者はソケットライブラリを用いて実装されている．なお，これらの通信はタスクマネージャ同士で直接行われる．本研究では，このタスクスケジューリングシステム上で動作するタスクスケジューラを開発し，ストリーミングデータ処理を含むアプリケーションを用いて評価を行った．

3.3.2 資源情報サーバ (RIS)

資源情報サーバ (RIS: Resource Information Server) は，各ワーカのプロセッサ負荷やネットワーク負荷を含む資源情報を計測・蓄積し，過去の情報の提示や予測の提供を実現している [7].

3.4 実験 1: 経路選択の評価実験

3.4.1 実験方法

タスクに適した特性のネットワークを選択することの有用性を評価するために，ストリーミング型プログラムと非ストリーミングプログラムを，

1. 全ての通信に経路 A を用いた場合
2. 全ての通信に経路 B を用いた場合
3. ストリーミング通信に経路 B，通常の通信に経路 A を用いた場合

の 3 種類の状況で実行し，それぞれのプログラムの実行時間を計測した．

なお，この実験は通信負荷を掛けずに行った．

3.4.2 結果と考察

表 2 は，ストリーミング型プログラムと，非ストリーミングプログラムを，それぞれ個別に動作させた場合の結果である．経路 A,B の両方を使い分けた場合，ストリーミング・非ストリーミング共に効率良く実行できていることが確認できる．

表 2: 各プログラムの実行時間
Table 2: Elapse Time of Each Program.

ジョブ ＼ 利用経路	A のみ	B のみ	A,B
ストリーミング [sec]	560	225	204
非ストリーミング [msec]	1660	2715	1660

表 3 は，両方のプログラムを同時に実行した場合の，それぞれの実行時間である．経路 A,B を使い分けることにより，効率的な実行が出来ていることが確認できる．個別に動作させた場合と比較すると，経路 A のみを利用した場合の結果が大きく悪化しており，高速に実行できていた非ストリーミングプログラムの実行も，非常に遅くなっている．これは，ストリーミング型プログラムに大きく帯域を圧迫されたことが原因だと考えられる．

表 3: 各プログラムの実行時間 (同時実行)
Table 3: Elapse Time of Each Program (Job Level Parallel Execution).

ジョブ ＼ 利用経路	A のみ	B のみ	A,B
ストリーミング [sec]	601	270	277
非ストリーミング [msec]	6671	2692	1865

3.5 実験 2: マイグレーションの評価実験

3.5.1 実験概要

負荷変動に対するマイグレーションの有用性を評価する実験として，タスクマイグレーション実装/非実装

の2種類のスケジューラを用い, 3.1 で示した通信負荷を加えて実験を行った. 動作させたアプリケーションはストリーミング型プログラムのみで, 平均実行時間および実行時間の標準偏差を用いて評価する.

ワーカの選択手法は, どちらのスケジューラも 2.2.1 で示した方法である. RIS のネットワークの予測値には, 過去 1 分間のネットワーク帯域幅の平均値を用いた. 高精度予測手法 [7] を用いなかった理由は, 実験環境が普段あまり利用されてない環境で, 予測に必須となる有効なログが無い上に, 加えた負荷が完全にランダムであるため, 有効な予測が出来ないと判断したからである. マイグレーションを実装したスケジューラではデータ流量を 10 秒毎に測定し, その値と RIS によるネットワーク性能の実測値の和が 15MByte/s を切った場合にマイグレーションを行うように設計した.

3.5.2 結果と考察

実験結果を表 4 に示す. マイグレーションを行うことにより, 平均実行時間が短縮できていることが確認できる. また, 実行時間の標準偏差が小さくなっている, つまり実行時間のばらつきが抑えられていることが確認できる.

実行時間のばらつきが小さくなると, ジョブの終了時刻の予測精度が高まり, それにより資源情報の予測精度も向上する. その結果, 他のジョブやタスクに効率の良い資源割り当てを行うことが可能となり, 実行時間の短縮が図れる.

動的に負荷が変動している環境上において, このような安定した性能が残せたことは, 非常に有用な特性であるといえる.

表 4: マイグレーションの効果

Table 4: Effect of Task Migration.

マイグレーションの有無	有	無
平均実行時間 [sec]	376.9	530.9
最短実行時間 [sec]	298	284
最長実行時間 [sec]	461	936
実行時間の標準偏差	51.3	217.0

4 おわりに

本研究では, ストリーミングデータ処理を含む分散プログラムと通常の分散プログラムを共に効率的に実行できるようなタスクスケジューリング手法を提案し, 評価実験を行った. その結果, 通信形態に適した特性を持つネットワークを利用することで, 全体の実行時間が短縮できることを確認した. また, タスクマイグレーションを行うことで, 動的変動がある環境上で安定して高い性能を維持することが確認できた. 今後はネットワーク負荷やプロセッサ負荷の予測手法との連携や, マイグレーションを行う閾値の自動設定方法の検討を行い, 効率化を図っていく予定である.

謝辞 本研究の一部は, 文部科学省科学研究費補助金 (課題番号 : 18500056,18100001) および総務省の援助を受けている.

参考文献

[1] Kwok, Y. and Ahmad, I.: Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors, *ACM Computing Surveys*, Vol. 31, No. 4, pp. 406-471 (1999).

[2] Koide, H. Hirayama, H., Murasugi, A. Hayashi, T. and Kasahara, H.: Meta-scheduling for a Cluster of Supercomputers, *Proceedings of International Conference on Supercomputers Workshop, Scheduling Algorithms for Parallel/Distributed Computing - From Theory to Practice -*, 1999, pp. 63-69.

[3] Yamamoto, H., Kawahara, K., Takine, T. and Oie, Y.: Investigation and Fundamental Analysis of Process Allocation Scheme on Grid Computing Environment, *Techical Report of IECE*, IN2002-8, pp.43-48 (2002).

[4] 吉永一美, 小出洋, ストリーミング処理のためのタスクスケジューリング, 情報処理学会 HPC 研究会 2006-HPC-107-24, pp.139-144, (2006).

[5] 樋田博信, 池永全志, 尾家祐二, 複数経路表を用いたトラヒックエンジニアリング機構の実装と評価, 電子情報通信学会 技術研究報告, NS2003-308, IN2003-263, pp.65-70, (2003).

[6] Standard Task Graph Set: Kasahara Laboratory, Waseda University, `http://www.kasahara.elec.waseda.ac.jp/schedule`.

[7] 小出洋, 山岸信寛, 武宮博, 笠原博徳, 情報資源サーバにおける資源情報予測の評価, 情報処理学会論文誌:プログラミング, Vol. 42, No.SIG3(PRO 10), pp.65-73 (2001).