社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2006－HPC－107 (14)
2006／8／1

タスク並列スクリプト言語処理系における広域分散実行方式

高木祐志[†] 西川雄彦[†] 大野和彦[†]
佐々木敬泰[†] 近藤利夫[†] 中島浩[††,☆]

我々は，メガスケールコンピューティング向けの並列プログラミング言語として MegaScript を開発している．MegaScript は，独立した逐次/並列の外部プログラムをタスクとして扱い，複数のタスクを並行/並列に実行する．また，タスクの標準入出力を接続し合うことでタスク間の連携を実現している．従来の MegaScript 処理系はマスター/スレーブモデルによる実装が行われていたが，ホスト数増加によるマスターホストの負荷集中が問題となっていた．そのため我々は現在，階層型動作モデルによる実行方式を採る MegaScript 処理系の実装を進めており，上記問題の解決を図っている．しかし，階層型処理系のタスク間通信手法が広域分散環境に適していないために，タスクの接続方法によっては，大幅な通信オーバヘッドが発生し，MegaScript の実行性能を低下させてしまう問題を抱えている．本論文では，広域分散環境における効率の良いタスク間通信手法の提案を行う．予備評価では提案手法を用いることによるオーバヘッドの削減が十分に期待できるものであることを確認した．

Widely-Distributed Implementation of Task Parallel Script Language MegaScript

YUJI TAKAGI,[†] TAKEHIKO NISHIKAWA,[†] KAZUHIKO OHNO,[†]
TAKAHIRO SASAKI,[†] TOSHIO KONDO[†]
and HIROSHI NAKASHIMA[††]

We are developing a task parallel script language named MegaScript for megascale computing. MegaScript regards independent sequential/parallel programs as tasks, and executes them in parallel.

The current implementation of MegaScript is based on the hierarchical model to distribute the load of the master host. However, its inter-task communication overhead is still large. So we propose a new communication scheme for widely-distributed environment.

1. はじめに

遺伝子解析や環境シミュレーションなどの分野では高精度で複雑な数値計算を要するため，近年では Pflops 以上の計算能力が期待されている．Pflops を超える性能を得るには，100 万台規模のプロセッサを用いた並列計算が不可欠である．このためコモディティな技術を用いた「低電力化とモデリング技術によるメガスケールコンピューティング」の研究が行われており，我々は，メガスケールコンピューティング向けの開発言語としてタスク並列スクリプト言語 MegaScript[1] を開発している．

MegaScript は逐次や並列の外部プログラムを実行単位 (タスク) とする．各タスクの標準入出力をつなぎ合わせることでタスク間の連携を可能とし，各タスクを複数の計算機上で同時実行することで並列実行処理を実現している．

ホストの管理/制御には従来，集中管理型のマスター/スレーブモデルを採用していた[2)3)]．しかし，スケジューリング負荷や，計算ホストの制御負荷がマスターホスト 1 台に集中し，スケーラビリティに欠けてしまう問題があった．このため，我々は階層型動作モデルによる実行方式を採る処理系の提案を行い[4)5)]，実装を進めている．

この実装により，計算ホスト数の増加に対するスケーラビリティの問題は解消可能であると考える．一方で，ユーザが定義するタスクの接続方法によっては，大幅な通信オーバヘッドが発生してしまう問題が明ら

† 三重大学
Mie University
†† 豊橋技術科学大学
Toyohashi University of Technology
☆ 現在，京都大学
Presently with Kyoto University

かになった．この問題はMegaScriptの実行性能を大幅に低下させる危険性を持つものであり，問題解決には階層型処理系のタスク間通信の実装を改良する必要がある．

そこで，本稿では広域分散環境における効率の良いタスク実行処理手法の提案を行い，予備評価により提案手法の性能について確認した．

以後，第2章でMegaScriptの概要を述べ，第3章でMegaScriptの基本機能が処理系でどのように実装されているかを説明する．第4章で問題点を具体例を挙げて説明し，第5章でその問題を解決する手法の提案を行い，第6章で予備評価を行い，最後にまとめる．

2. タスク並列スクリプト言語MegaScript

2.1 言語の概要と基本設計

MegaScriptはメガスケールの並列処理を目的とするプログラミング言語である．ユーザは処理の主要部分を別プログラムとして用意し，MegaScriptプログラム中でこれを計算タスクとして生成・実行する方式をとる．

各タスクは複数の計算機上で並行／並列に動作する．また，各タスクの標準入出力をストリームと呼ばれる通信路を用いることでタスク間のデータの中継を行う．

ストリームには入力部と出力部が存在し，それぞれ複数のタスクを接続することができる．入力部に複数のタスクを接続した場合，メッセージは行単位で非同期にマージされ，出力部に複数のタスクが接続されたときはそれぞれのタスクに対してマルチキャストが行われる (図1)．

2.2 スケジューラ

定義したタスクやストリームをどのホスト上に配置・生成するかは，ユーザが意識しMegaScriptプログラム中に明示する必要はない．これらは処理系に内蔵されているスケジューラが自動的に配置戦略や生成タイミングを決定し，効率の良い実行処理を行う．

2.3 プログラミング

MegaScriptはオブジェクト指向言語Ruby[6]の拡張言語であり，Rubyで提供されている基本機能はそのまま使用出来る．このため，必要に応じて複雑な処理をMegaScriptプログラム中に記述することが出来る．

MegaScriptでは，タスク・ストリームを表すクラスを提供している．ユーザがタスク・ストリームを扱う際にはこれらのクラスのインスタンスを作成し，メソッドを呼び出すという形で行う．作成したインスタンスはスケジューリングキューに登録した後に，スケジューラを呼び出すことで，自動的に計算ホストへの割り当てが行われ，各タスクの実行が開始される．

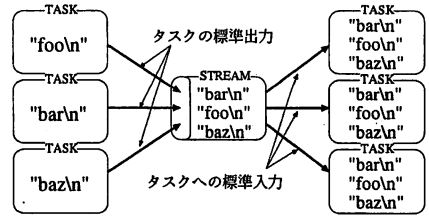

図1 ストリームの振る舞い
Fig. 1 Behavior of Stream

また，各タスクの実行特性（タスクの計算量や通信発生量，通信パターンなど）をメタプログラムとして追加記述することにより，これらの特性を考慮したスケジューリングが行われ，実行効率を向上させることができる．

3. MegaScript処理系

3.1 動作モデル

従来の処理系では計算ホストの管理/制御に集中管理型のマスター・スレーブモデルが採用されていた．しかし，計算ホスト数が増加するにつれスケジューリングを担当するマスターホストの計算量が増加してしまい，スケーラビリティに欠けるという問題があった．このため，我々は階層型動作モデルによる処理系の実行方式を提案し[5]，実装を進めている．

現在実装を進めている処理系では，タスクの実行処理に使用する計算ホストをドメイン単位でグループ化し (以後ホストグループと呼ぶ)，ホストグループを階層的に接続して管理を行う階層型動作モデルを採用している (図2)．ユーザはあらかじめ各ホストグループにそれぞれ1台代表ホストを決定し，グループ内の計算ホスト情報と下階層に接続されるホストグループの代表ホストを記述したホストファイルを作成しておく．処理系はこのホストファイルを読み込み，MegaScriptプログラム実行ホストを頂点とする階層型の処理系制御環境を構築する．

接続関係を持つ処理系間のネットワーク回線は実際にはそれぞれ異なる通信性能を持つが，処理系では，ホストグループ内は高速なLAN，ホストグループ外は低速なWANとして2種類に分類して扱うことで単純化を行っている．

3.2 階層型スケジューラ

実装中の処理系では，各代表ホスト上でそれぞれスケジューラが動作する．個々のスケジューラのスケジューリング対象ホストは，自らが所属するホストグ

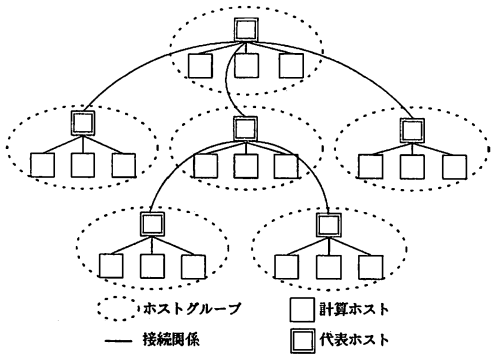

図 2 処理系の動作モデル
Fig. 2 Hierarchical Model of MegaScript Runtime

ループ内の計算ホストと，下階層に接続されているホストグループである．

図 2 の頂点のホストから生成されたタスクのうち，一部を頂点のホストグループ内で処理し，残りのタスクは下階層のホストグループへ処理を依頼する．依頼されたホストグループ上では上記同様の処理が行われ，タスクが次々と下層まで伝搬される．

3.3 ストリームの扱い

ストリームの実装概要を図 3 に示す．処理系では MegaScript プログラム中に定義された 1 つのストリームに対し，入力端と出力端をそれぞれ生成する．入力端は入力側に接続されているタスクと同じホストに配置され，出力端は出力側に接続されているタスクと同じホストに配置される．

出力端のうち 1 つは代表者として処理系内で扱われる．ストリームを流れるメッセージは一旦代表出力端に集められてメッセージのマージが行われる．代表出力端には，代表者以外の出力端の設置先ホストが処理系より与えられており，その情報をもとにストリームメッセージの転送を行う．以上の実装により，処理系内でマージ/マルチキャストが実現されている．

3.4 通信機構

近年の研究室や企業などのネットワーク環境では，外部ネットワーク上の端末からローカル内へのアクセスが可能な端末やポートを制限するなど，セキュリティに対する対策を施していることが多い．そのため，ホスト同士が直接通信を行えない可能性があり，通信が可能となる経路を自動で判断し，複数ホスト間を経由してメッセージ転送を行う通信機能が求められる．

これらの要求に対し通信ライブラリに Phoenix[7] を用いることで解決を図っている．Phoenix は広域分散環境対応のメッセージパッシングライブラリであり，

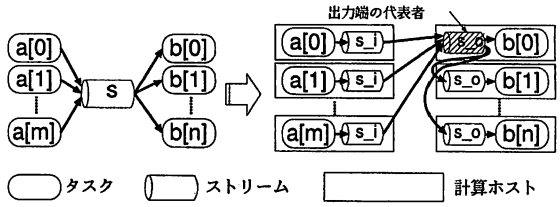

図 3 ストリームの実装手法
Fig. 3 Implementation of Stream

通信経路情報を独自に調査し，必要に応じ自動でメッセージの経由を行う機能を有する．

4. 現在のランタイムの問題点

4.1 ストリームのマルチキャスト

現在，処理系が抱えている問題点について，タスクとストリームが図 4 のように定義され，スケジューラによって図 5 のようにタスク配置が決定した例を用いて説明する．この例では，n 個の Task b のうち，ホストグループ (以下 HG)1 に b[0] から b[i-1] までの i 個のタスクを，HG2 に b[i] から b[j-1] までの j-i 個のタスクを，HG3 に b[j] から b[n-1] までの n-j 個のタスクを，それぞれ割り当てている．また，出力端の代表者は HG1 のホスト H_{1,2} に，Task a は HG2 のホスト H_{2,2} に割り当てている．

図 5 中の Task a から出力されたメッセージは出力端の代表者に収集されるため，H_{2,2} から H_{1,2} への WAN 通信が発生する．その後，出力端の代表者は出力端を保持する全てのホストへメッセージの転送を行うため，HG1 の LAN 通信が i-1 回，HG1 から HG2 への WAN 通信が j-i 回，HG1 から HG3 への WAN 通信が n-j 回発生することになる．これらの通信は Task a から出力されたメッセージであるので，すべて同じ内容のデータである．さらに，HG3 で発生したメッセージを同グループ内の出力先タスクに届けるために，一旦外部のネットワークを往復するという無駄な通信オーバヘッドが発生してしまう．

以上をまとめると，図 4，5 の例におけるマルチキャスト処理の問題点は以下の 3 つである．

- ホスト H_{1,2} の LAN/WAN 通信負荷の集中
- データが HG3 に到達するまで 2 度の WAN 経由
- HG2 内の出力端への不要な WAN 経由

なお今回の例では，問題点の説明を行いやすくするため代表出力端を HG1 に配置した．だが，Task a が配置されている HG2 内に代表出力端を配置するほうが第 3 項目の問題点である不要な WAN 経由の発生を防ぎ，一般に効率が良いのではないかと考えること

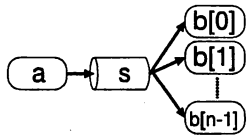

図 4 マルチキャスト型のタスクネットワーク
Fig. 4 Multicast-type Task Network

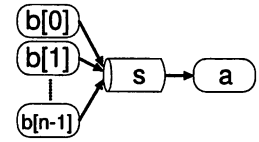

図 6 マージ型のタスクネットワーク
Fig. 6 Merge-type Task Network

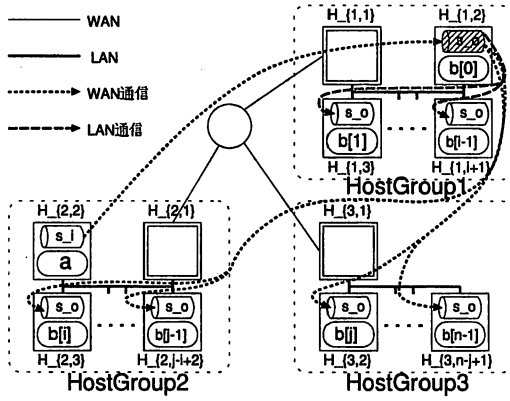

図 5 マルチキャストの従来実装手法
Fig. 5 Conventional Implementation of Stream Multicast

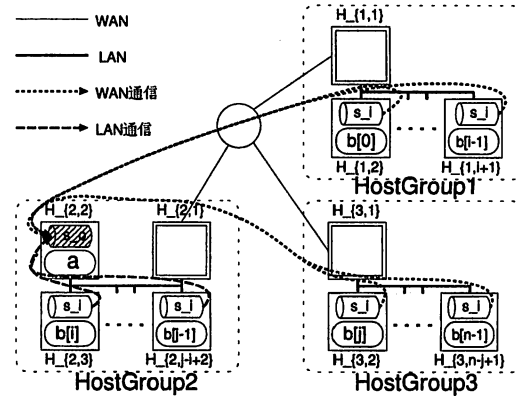

図 7 マージの従来実装手法
Fig. 7 Conventional Implementation of Stream Merge

もできる．しかし，発生する WAN 通信回数を比較すると，代表者を HG2 に設置した際は n-j+i 回であり，代表者を HG1 に設置した際は n-i+1 回である．よって，i や j の値によって大小関係が変化するため，代表者をどのホストに配置すれば良いかは状況によって変化する．

4.2 ストリームのマージ

次にマージについても同様に考えることが出来る．タスクとストリームが図 6 のように定義され，スケジューラにより図 7 のように配置が決定されたと考える．ストリーム出力端の代表者については，ストリーム出力端が 1 つしか存在しないので必然的にホスト H_{2,2} に配置される．

このとき，各 Task b からデータが出力されるたびに，ホスト H_{2,2} に対する通信が発生し，HG3 の WAN 回線に集中的な通信負荷が発生する．これについてはマルチキャスト時の問題とは異なり，メッセージの内容は個々に異なるものである．また，ホスト H_{2,2} は次々と到着するメッセージの処理を行わなければならず，受信データのバッファリングやメモリへのコピーなどに多くの計算負荷が発生し，Task a の実行処理に影響を与える可能性があると考えられる．

以上まとめると，図 6，7 におけるマージ処理の問題点は以下の 2 つである．

- HG2 への WAN 通信の頻発
- ホスト H_{2,2} の受信処理の負荷集中

5. ストリーム通信機構の改良

4 章で挙げた問題点を解決するには，メッセージの送信経路を変更し，WAN を経由する冗長な通信を抑える制御が必要となる．そこで，これらの問題の解決案として，マルチキャストとマージを実現する新たな通信方式を提案する．

5.1 ストリームのマルチキャスト

今回提案する MegaScript ストリームのマルチキャスト処理手法のアルゴリズムは以下の通りである．

(1) タスクから出力されたメッセージを自グループ内の代表ホストへ転送する
(2) さらに，他ホストグループでそのメッセージを必要としているホストが 1 台以上あれば，そのホストグループの代表ホストにメッセージのコピーを送信する
(3) 各代表ホストは，ホストグループ内でそのメッセージを必要としているホストへメッセージの転送を行う

上記のアルゴリズムを 4.1 節と同様の実行環境例である図 8 を用いて説明する．まず (1) より，Task a より出力されたメッセージはグループ内代表ホストへ

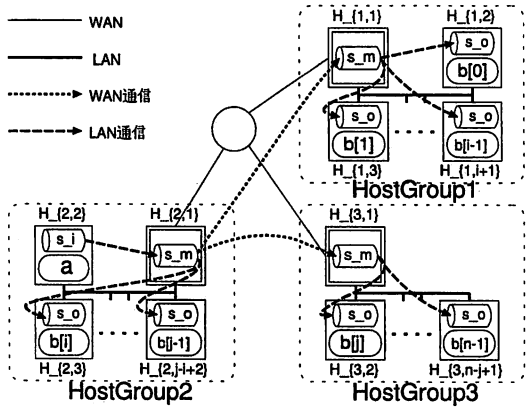

図 8 マルチキャストの提案手法
Fig. 8 Proposed Implementation of Stream Multicast

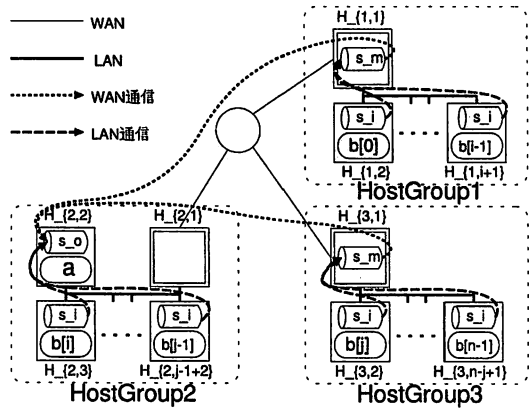

図 9 マージの提案手法
Fig. 9 Proposed Implementation of Stream Merge

と転送される．次に (2) より，HG2 の代表ホストは HG1 と HG3 の代表ホストへメッセージの転送を行い，その後 (3) より，各代表ホストからグループ内の出力端へメッセージが転送される．

以上の操作により，すべての Task b に対してメッセージが配信されることになる．

5.2 ストリームのマージ

MegaScript ストリームのマージ処理手法の提案アルゴリズムは以下の通りである．

(1) 出力端が同じホストグループ内に存在すれば，出力端へ直接送信を行う．存在しなければホストグループ内の代表ホストへ送信する．

(2) メッセージを受信した代表ホストは，メッセージを送信バッファに追加格納し，特定のタイミングで送信バッファのメッセージをパックし，必要としているホストに対して直接送信する．

(3) パックされたメッセージを受け取ったホストは，メッセージの解凍を行い，タスクに対してデータを受け渡す

上記のアルゴリズムを 4.2 節と同様の実行環境例で図 9 を用いて説明する．まず (1) より，HG2 内の Task b メッセージは直接 H_{2,2} に対して送信する．HG1，HG3 の Task b メッセージはそれぞれ HG1，HG3 の代表ホストへと集められ，(2) により特定のタイミングで送信バッファのパッキングを行い，ホスト H_{2,2} に対してパックされたメッセージを送信する．ホスト H_{2,2} は受信したメッセージの解凍を行い，Task a に対してメッセージを受け渡す．以上の操作により，Task b 出力データがすべて Task a に配信される．

5.3 既存手法との適応的な切り替え

4.1 節で説明した従来のマルチキャスト例では総通信回数は n 回であり，内訳は LAN 通信が i-1 回，WAN 通信が n-i+1 回となっている．最長経由経路は WAN2 回である．一方，5.1 節で説明した提案手法によるマルチキャスト例では総通信回数は n+3 回であり，内訳は LAN 通信が n+1 回，WAN 通信が 2 回となっている．最長経由経路は LAN2 回+WAN1 回である．これらを比較すると，従来手法では出力端代表者のホストグループ以外に配置されるタスク数が増加すると，通信コストが非常に大きくなることがわかる．

一方で，WAN 回線が LAN 回線と比較して性能差がそれほど大きくない実行環境では，発生するオーバヘッドは通信回数が重要な要素と考えられる．そのため，上記条件では従来手法に比べて，総通信回数やホスト経由の最大ステップ数が多い提案手法のほうが性能が低下してしまうことが予想できる．

効果的な性能向上を目指すには，各ストリームに対して従来手法と提案手法のどちらが適しているかを自動で判断し，適応的に通信手法を切り替えることが必要である．

切り替え判断に参考となる情報は以下のものが考えられる．

- タスク配置情報
  比較的多数のタスクが各ホストグループに均等に配置されているほうが提案手法に適していると予想される．
- 通信回線情報
  バンド幅が小さく，遅延時間の大きいものほど提案手法に適していると予想される．
- メタプログラム情報
  通信量が大きく，通信頻度が高いものほど提案手法に適していると予想される．

表 1 評価環境
Table 1 Evaluation Environment

	三重大	豊橋技科学
CPU	Pentium4 2.8GHz	PentiumIII 1.0GHz
Memory	512MB	256MB
Network	1000Base-T	——
OS	Linux 2.4.31	Linux 2.4.26

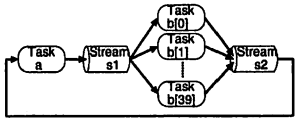

図 10 評価内容
Fig. 10 Evaluation contents

6. 予備評価

今回提案した手法がどの程度の効果をもたらすのかを予測するために，表 1 の評価環境で予備評価を行った．三重大学 (以下三重大) と豊橋技術科学大学 (以下豊橋技科大) の RoundTripTime は約 6.5ms である．

6.1 評価内容

図 10 の MegaScript プログラムを実行した際に発生する従来手法と提案手法のタスク間通信を擬似的に発生させる評価プログラムを作成し，通信開始から通信終了までの時間計測を行った．なお，s2 は Task b のメッセージ受信完了通知用のストリームであり，通信量は s1 と比較して十分小さいものである．評価プログラムは，通信ライブラリに Phoenix1.0 を用いて，gcc バージョン 3.2 でコンパイルを行った．

タスク/ストリームの各配置については，Task a は豊橋技科大に，Task b は三重大に 20 タスク，豊橋技科大に 20 タスク配置し，s1 の代表出力端は豊橋技科大に配置した例を用いる．また，実験用に計算機の台数を十分確保できなかったため，豊橋技科大側ではすべてのタスク (に相当する評価プログラム) を同一ホストで実行している．Task a の出力メッセージサイズと通信回数を変化させてそれぞれ計測を行った．

6.2 評価結果と考察

実験結果を表 2 に示す．各項目 5 回実行した平均時間である．今回の条件では，従来手法と比較して 1.7 倍から 2.6 倍の処理速度向上の結果が得られた．また，通信回数が 10 倍になると，通信時間もおよそ 10 倍程度となっていることから，通信回数は，両手法の切り替え判断の要素ではないことが推測できる．

表 2 評価結果
Table 2 Result of the Evaluation

通信サイズ＼通信回数	10	100	1000
1 kbyte(従来手法)	1.135s	10.647s	125.538s
1 kbyte(提案手法)	0.619s	6.2730s	66.598s
100 kbyte(従来手法)	6.291s	69.897s	696.994s
100 kbyte(提案手法)	2.781s	27.710s	276.803s

7. おわりに

本稿では，現在実装中である階層型処理系のタスク間通信時に発生する問題点を指摘し，その問題を解決するために従来の手法を改良した新たなストリーム通信処理方式の提案を行った．予備評価では，40 タスクという比較的小規模なマルチキャスト処理でも，提案手法により 2 倍程度の速度向上を得ることができた．

今後は，より多くのパラメータを用いて評価を行い，マージ/マルチキャストにおける切り替え基準の定式化を行う．その後，提案手法を階層型処理系に実装し．大規模環境で処理系の性能評価を行う予定である．

謝辞 本研究は，科学技術振興事業団・戦略的基礎研究「低電力化とモデリング技術によるメガスケールコンピューティング」による．

参考文献

1) 大塚保紀, 深野佑公, 西里一史, 大野和彦, 中島浩: タスク並列スクリプト言語 MegaScript の構想, 先進的計算基盤システムシンポジウム SACSIS2003, pp. 73–76 (2003).
2) 西里一史, 大野和彦, 中島浩: タスク並列スクリプト言語 MegaScript のランタイムシステムの設計と実装, 情処研報 2003-HPC-95, pp. 119–124 (2003).
3) 西里一史, 大野和彦, 中島浩: タスク並列スクリプト言語 MegaScript 向けランタイムシステム, 情処研報 2004-HPC-99, pp. 7–12 (2004).
4) 西川雄彦, 高木祐志, 大野和彦, 佐々木敬泰, 近藤利夫, 中島浩: タスク並列スクリプト言語 MegaScript ランタイムの広域分散化, 先進的計算基盤システムシンポジウム SACSIS2005, pp. 251–252 (2005).
5) 高木祐志, 西川雄彦, 大野和彦, 佐々木敬泰, 近藤利夫, 中島浩: 効率の良い広域分散対応のタスク並列スクリプト言語の実現, 情処研報 2005-HPC-103, pp. 157–162 (2005).
6) まつもとゆきひろ, 石塚圭樹: オブジェクト指向スクリプト言語 Ruby, ASCII (1999).
7) Kenjiro, T., Toshio, E., Kenji, K. and Akinori, Y.: Phoenix : a Parallel Programming Model for Accommodating Dynamically Joining/Leaving Resources, *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming(PPoPP 2003)*.