修士論文

題目

大規模環境を想定した
階層型ワークフロー処理系の
効率的な通信方式

指導教員

大野　和彦　講師

平成24年度

三重大学大学院 工学研究科 情報工学専攻
計算機アーキテクチャ研究室

仲　貴幸 (411M513)

内容梗概

近年，大規模な並列アプリケーションを作成する手法としてワークフローへの関心が高まっている．そこで，我々は大規模ワークフローを容易に記述できる並列プログラミング言語として MegaScript を開発している．従来の MegaScript 処理系は集中管理型のマスター・スレイブモデルによる実装が行われていたがスケーラビリティに欠ける問題があった．そのため階層型の動作モデルによる処理系の提案を行い，実装を進めている．しかし従来の通信方式を階層化した処理系に適用すると，ファイアウォールによる通信の阻害や不要なホストを中継するといった問題が生じて効率的な通信を行うことができない．

そこで，本論文では階層化した MegaScript 処理系に適した通信方式の提案を行う．提案手法では既存の通信効率化手法であるマルチキャスト，バッファリング，ルーティング最適化を適宜組み合わせ，処理系に適用する．マルチキャストでは一対多のホスト間通信を行うときに適用し，代表ホストを通じて個々のホストにメッセージを送信することで通信効率の向上を図る．バッファリングではいくつかのメッセージをまとめて送信することで通信回数を削減する．ルーティング最適化では送信先のホストを選択する．ホスト間通信が発生したとき，処理系実行時のホスト構成やファイアウォールによる対象ホスト間の直接通信の可否といった静的情報を参照し，直接通信もしくは可能な限り中継ホスト数を削減し処理系全体の通信効率を向上させる．

性能評価のために，提案した通信方式と従来の通信方式においてそれぞれ一対多，多対一の通信パターンのワークフローを実行し，実行時間を比較して評価を行った．その結果，ワークフローの実行時間を削減することができた．

三重大学大学院　工学研究科

Abstract

We are developing a parallel programming language MegaScript for mega-scale computing. MegaScript regards independent sequential/parallel programs as tasks and executes them in parallel. The current implementation is based on a centralized control architecture which consists of a master host and slave hosts. However, the architecture lacks scalability because the master host will be the bottleneck for the large number of slave hosts. To solve this problem, we are developing an architecture which is based on a hierarchical model.

In this paper, we propose an efficient communication scheme for the hierarchical model. It applies exitsing communication methods: multi-cast, buffering, and route selecting to the runtime adaptively. Multi-cast is applied when the number of receiver hosts is large. A sender host sends messages to their parent hosts and then each parent host sends the messages to receivers. Buffering is applied when the message size is small. Multiple messages are merged to a single message and sent it to the destination host. Route selecting determines routes to reduce the number of relaying hosts.

To evalate the proposed scheme, the execution time of the proposed scheme was compared with the conventional scheme on the hierarchical model. As a result, the proposed scheme could reduce the exection time.

三重大学大学院　工学研究科

目 次

三重大学大学院　工学研究科

図 目 次

表 目 次

1　はじめに

近年，大規模計算の需要が増大する一方で，単一プロセッサでの性能向上は頭打ちになりつつあり，並列処理への期待が高まっている．特にコストパフォーマンスやスケーラビリティの面で大きな利点があるため，PC クラスタの利用に注目が集まっている．

大規模な並列アプリケーションを作成する手法の一つとして，複数の独立したプログラムをタスクとして組み合わせるワークフローが使われている [1, 2, 3]．ワークフローは，各タスクの独立性が高いため既存のソフトウェアを再利用しやすいことやタスク間のデータフローを非循環有向グラフ (DAG) の形で記述できる等の利点を持つ．また，ワークフローでは一般にタスクやタスク間通信の粒度が大きく，大規模な計算資源を安価に供給できる広域分散環境との相性がよい．このため，天文学・物理学等の科学技術の分野において，大規模な問題を解く手段として利用されている [7, 8]．

そこで我々は大規模ワークフローを容易に記述できるタスク並列スクリプト言語 MegaScript の開発を進めている [9, 10, 11, 12, 13]．従来の MegaScript 処理系では集中管理型のマスター・スレイブモデルによる実装が行われていたがスケーラビリティに欠ける問題があった．そのため，階層型動作モデルによる処理系の提案を行い実装を進めている [14]．しかし従来の通信方式を階層化した MegaScript 処理系に適用するとファイアウォール等による通信の阻害や不要なホストを中継するといった問題が生じて効率的な通信が出来ず，性能が大幅に低下することがある．

そこで，本論文では階層化した MegaScript 処理系に適した通信方式の提案を行う．提案手法では既存の通信効率化手法であるマルチキャスト，バッファリング，ルーティング最適化を利用し，処理系で発生する一対一や一対多，多対一といった通信パターン毎に適切な手法を組み合わせ適用する．

以下，2 章で MegaScript の概要について述べ，3 章で MegaScript 処理系について述べる．4 章では提案する階層化に対応した通信方式について説明を行い，5 章で提案手法の実装について述べる．6 章で評価を行い，最後に 7 章でまとめを行う．

三重大学大学院　工学研究科

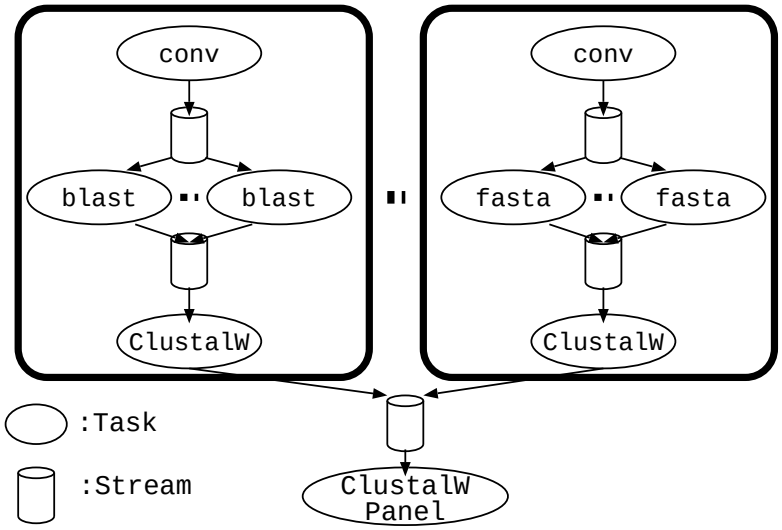

図 2.1: ワークフローの例

2 背景

2.1 タスク並列スクリプト言語 MegaScript

MegaScript はワークフローモデルに基づく粗粒度並列言語である．ユーザは処理の主要部を別プログラムとして用意し，MegaScript プログラム中でタスクとして生成・実行する．各タスクは PC クラスタ等の複数の計算ホスト上で並行・並列に実行され，ストリームと呼ばれる仮想的な通信路を介してタスク間の連携を実現している．MegaScript プログラムでは，タスクやストリームから構成されるワークフローやタスクの実行に必要な情報等を記述する．MegaScript 処理系はこの情報を解析し，スケジューリング結果に従って各タスクを指定された計算ホストで実行する．

ここで未知の病原菌の DNA 配列群を既知の病原菌データと比較して特定を行い，またそれらの相互比較を行うワークフローの例 (図 2.1) を示す．まず未知の DNA 配列群に対し $BLAST$ [4] や $FASTA$ [5] を使用し既知のシーケンスデータベースとの相同性検索を行う．その結果を利用して $ClustalW$ [6] においてマルチプルアライメントを行い，樹形図を得る (図 2.1 の太枠内)．これを複数のシーケンスデータベースで行い，シーケンスデータベースごとに得られた樹形図を $ClustalW\,Panel$ [6] で読み込み相互比較を行う．

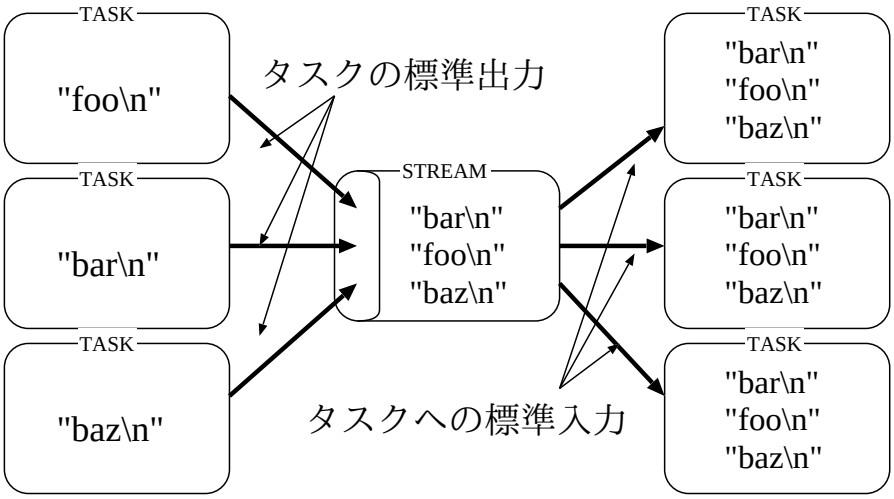

図 2.2: ストリームの振る舞い

2.2 タスクとストリーム

タスクは MegaScript の並列実行単位である．タスクは独立性の高い逐次・並列の外部プログラムであり，その内部処理に MegaScript は関与しない．

ストリームはあるタスクの標準出力を他のタスクの標準入力に流すための仮想的な通信路である．1 つのストリームの入出力端にそれぞれ複数のタスクを接続することで一対多，多対多等のタスク間通信を実現する．入力端に接続した入力側タスク群の出力が非決定的にマージされ，出力端に接続した出力側タスク群にマルチキャストされる (図 2.2)．現実装では標準入出力のテキストストリームのみ扱い，行単位でメッセージとしている．

またワークフローにおいて，パラメータスイープのように同じプログラムを異なる条件・引数で多数実行する場合，それらのタスクはワークフロー上で同一ストリームに接続されており，出力頻度など似た性質を持つことが推測される．

3 MegaScript 処理系

本章では，MegaScript 処理系の概要について説明する．MegaScript 処理系は使用するホスト上にそれぞれランタイムプロセスを起動する．ランタイムプロセスはそのホスト上で実行するタスクを子プロセスとして生成し，またストリームの管理やタスクとストリームのスケジューリングや別ホスト上のランタイムプロセスとの通信を行う．異なるホスト上のタスク間の通信は各ホスト上に起動したランタイムプロセスを介して行われる．

3.1 タスク間通信

図 3.3 の環境上で図 3.4 の多対多のタスク間通信を行うワークフローを実行する場合を例に説明する．このワークフローはタスク a[i] を入力側タスク，タスク b[i] を出力側タスクとしており，それらをストリームで接続している．ここでは簡単のため n=m=1 とし，タスクとストリームは既にスケジューリングされ各ホストに配置されているとする．ランタイムプロセスはタスクプロセスと双方向パイプを用いて接続する．入力側タスクを実行するとき，ランタイムプロセスはタスクの標準出力をパイプを通じて取得する (図 3.3(a))．逆に出力側タスクを実行するときは，通信により受け取ったメッセージをランタイムプロセスからパイプを通じてタスクに与える (図 3.3(b))．

次にストリームの実装の概要について説明する．MegaScript 処理系ではプログラム中で定義された 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．入力端は入力側タスクを実行するホスト毎に 1 つずつ配置され (図 3.3 の s_i)，出力端は出力側タスクを実行するホスト毎に 1 つずつ配置される (図 3.3 の s_o)．出力端のうち 1 つは処理系内で代表出力端として扱われる．

ストリームを流れるメッセージは一度代表出力端に集められメッセージのマージが行われる (図 3.3(c))．代表出力端には代表出力端以外の出力端の配置先ホストの情報が処理系より与えられており，それを基にメッセージの送信を行い，タスク間通信を実現している (図 3.3(d))．

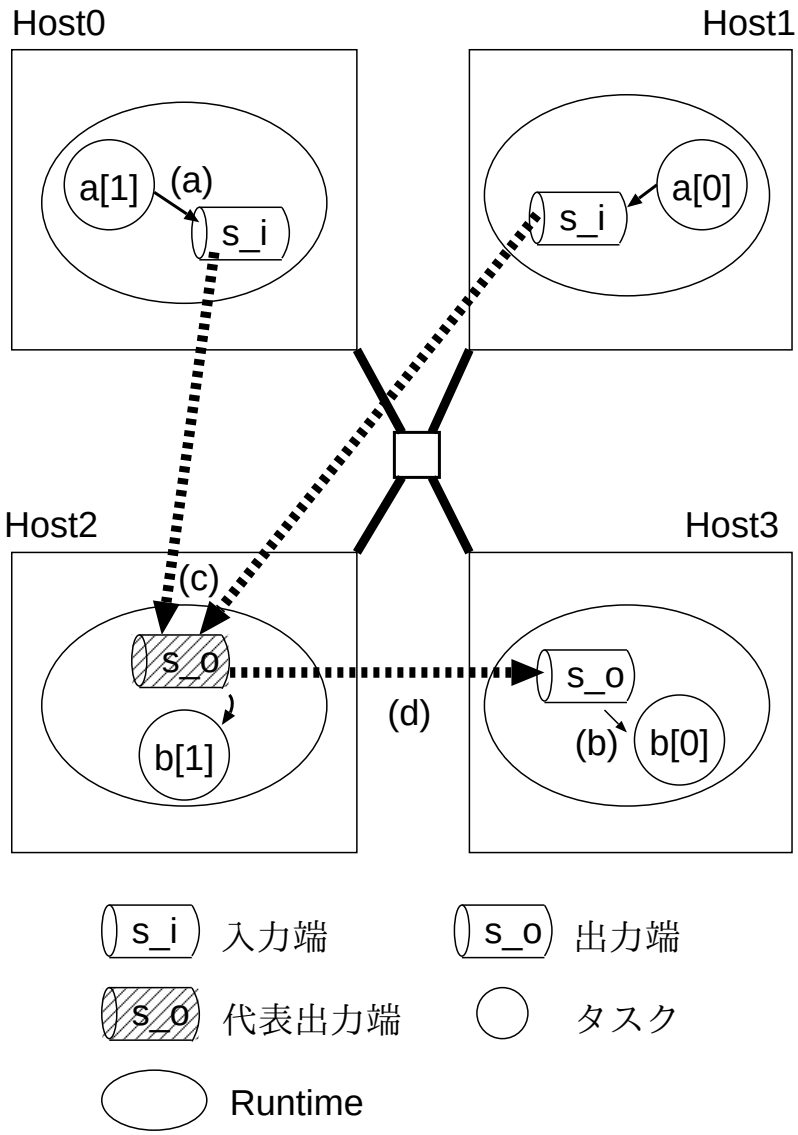

図 3.3: MegaScript 処理系の実装の概要

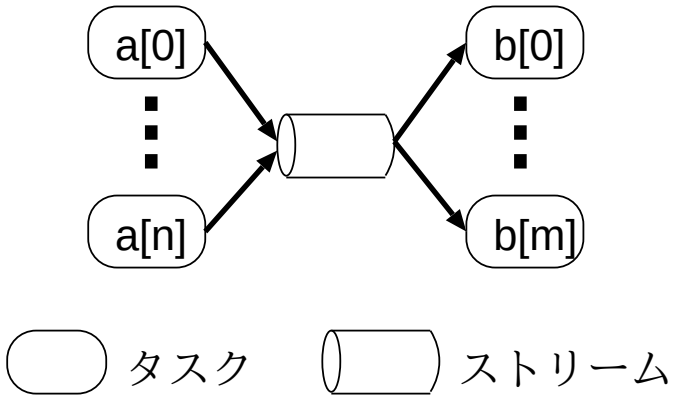

図 3.4: ワークフロー例

3.2 動作モデル

従来の MegaScript 処理系は，計算ホストの管理・制御の動作モデルとして集中管理型のマスター・スレイブモデルを採用していた．しかし実行環境が大規模化するに伴い，マスターホスト 1 台に計算ホストの管理・制御やスケジューリング等の負荷が集中しスケーラビリティに欠ける問題があった．

そこで我々はマスターホストの負荷を分散させ，より高いスケーラビリティを実現するためにホストの管理・制御やスケジューリングを行うサブマスターホストを新たに導入し，階層型動作モデルによる処理系の提案を行い実装を進めている [14]．

動作モデルを構築するホストにはマスターホスト，サブマスターホスト，スレイブホストがある．マスターホストは物理的なネットワーク距離が近いホスト群をスレイブホストとして起動・接続する．物理的なネットワーク距離が遠いホスト群についてはそのホスト内で代表ホストを一つ決定しそれをサブマスターホストとする．サブマスターホストはそのホスト内のホストをスレイブホストとして起動・接続する．また，サブマスターホストは物理的なネットワーク距離が近い他のサブマスターホストと接続を行う．ここで，サブマスターホストの管理するホスト群をグループ（以後ホストグループと呼ぶ）として扱い，ホストグループを階層的に管理・接続することで階層型動作モデルを実現している (図 3.5)．また，各ホストは基本的に直接接続されたホストのみと通信を行う．これは始め各ホストは他ホストの IP 等のホスト情報を保有していないこと

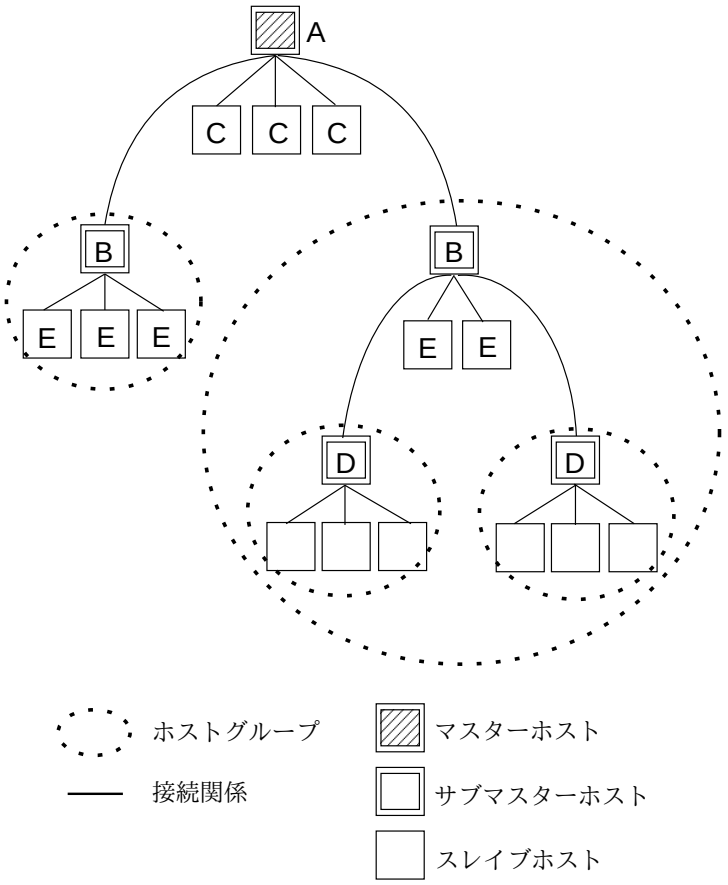

図 3.5: 階層型 MegaScript 処理系の動作モデル

やファイアウォール等が原因で直接通信できないことがあり，任意のホスト間の直接通信が可能とは限らないためである．通信ホスト間に直接通信を阻害する要因がないとき，新たな接続を構築する．

階層型処理系では，マスターホスト及びサブマスターホスト上でスケジューリングが行われ，スケジューリング結果に基づき各タスクやストリームをそれぞれ直下のホストに割り当てる．以降階層的にスケジューリングを繰り返し行い，最終的にスレイブホストにタスクやストリームを割り当てる．

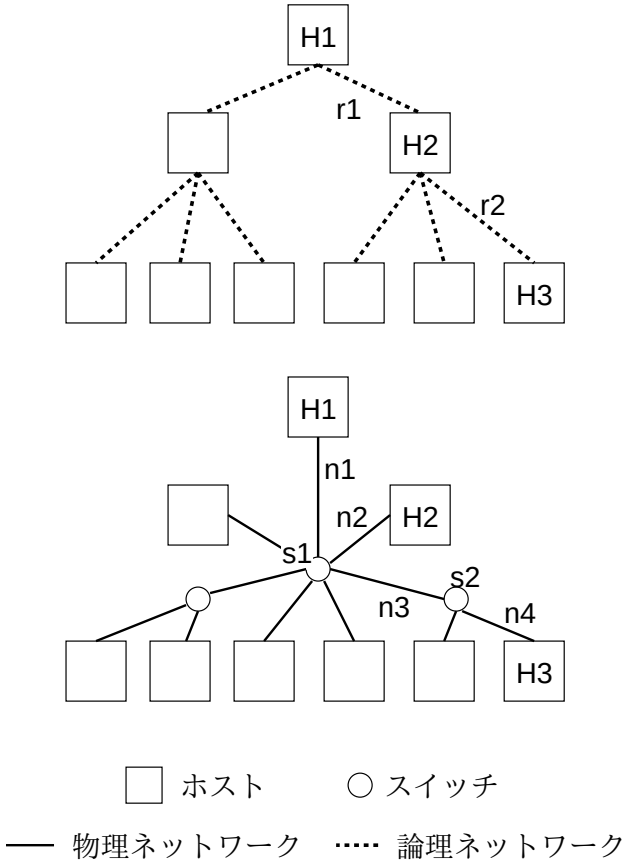

図 3.6: 論理ネットワーク（上）と物理ネットワーク（下）

3.3 物理ネットワークと論理ネットワーク

MegaScript 処理系における論理ネットワークと物理ネットワークについて説明する．処理系は上記で述べたように動作モデル構築を行っているため，論理ネットワークは図 3.6(上) のようになる．処理系でホスト H1 から H3 へ通信が発生した場合，通信は仮想的な通信路 r1，r2 及びホスト H2 を介して行われる．しかし，物理レベルの通信では H2 を介して行われるため，ネットワーク n1，スイッチ s1，n2，H2，n2，s1，n3，s2，n4 の経路を辿る (図 3.6(下))．このため，効率的なホスト間通信を行うには論理レベルでのルーティングが必要となる．

4　提案手法

MegaScript におけるタスク間通信は一対一，一対多，多対一，多対多の 4 パターンに大きく分類できる．タスク間通信では，3.1 節で述べたように入力側タスクの出力するメッセージは出力端の代表である代表出力端に一度集められマージされる．その後，そのホストから代表出力端以外の出力端を持つホスト群にメッセージを送信している．このことからタスク間通信をホスト単位の視点で考えると，多対多のタスク間通信は多対一と一対多のホスト間通信の組み合わせと置き換えることができる．したがって，タスク間通信は一対一，一対多，多対一の 3 パターンのホスト間通信で構成されているとみなすことができる．

そこで，ワークフローから得られたタスクとストリーム間の接続関係やホスト間の直接通信の可否等の情報を用い，MegaScript 処理系の通信パターンに応じて以下に述べる既存の通信効率化手法を適宜組み合わせて適用する．

4.1　マルチキャストの適用

代表出力端を持つホストから出力端を持つ各々のホストに一つずつメッセージを送信する場合，出力端を持つホスト数が多いと送信ホストに負荷が集中してしまう．そこで，一つのホストグループ内に出力端を持つホストが複数台あるときはそのホストグループの代表ホストであるサブマスターホストにメッセージを送信する．そしてメッセージを受け取ったサブマスターホストは，そのホストグループ内の出力端を持つすべてのホストにメッセージの送信を行う (図 4.7)．ホストグループ内に配置された出力端が多いほど異なるホストグループ間の通信回数やメッセージ量が削減でき，通信の効率化が見込める．

しかしマルチキャストは対象ホスト間の直接通信と比べて，本来不要なホストを中継するため通信回数が増大している．そのため，出力端の配置や通信性能によっては直接通信するほうが高効率となることがある．効果的な性能向上を目指すには，状況に応じてマルチキャストを使用するかどうかを決定する必要がある．

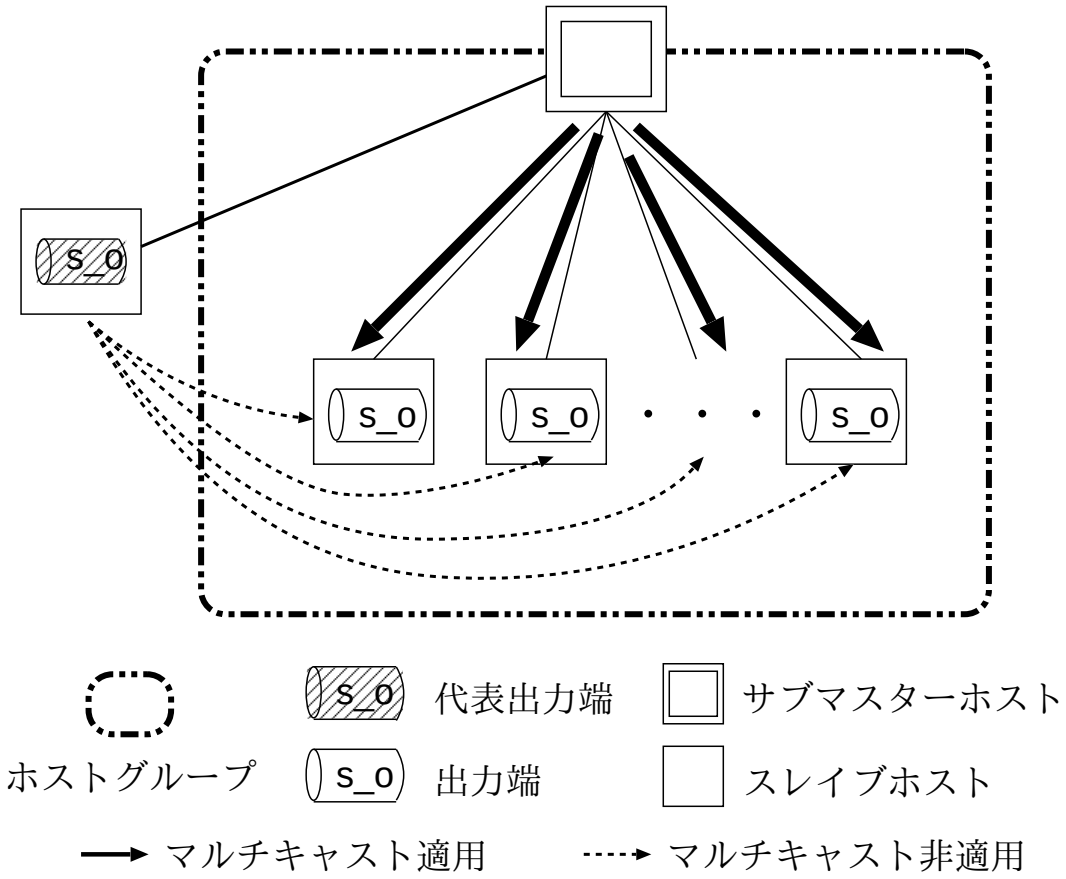

図 4.7: マルチキャスト手法

4.2 メッセージのバッファリングの適用

送信するメッセージのサイズが小さいとき，そのまま送信するのではなくメッセージをバッファに貯めておく．そして，ある一定のメッセージサイズになったら一度に送信する．このように通信のバッファリングを行うことによって，ホスト間で発生する通信回数を削減することが可能である．

例えば極めて小さいサイズのメッセージ通信が大量に発生するタスクが存在するとき，この手法を適用することで通信回数を大幅に削減することができる．異なるホストグループ間で通信を行う場合，WAN 等の低速なネットワークを経由する可能性が高いため，通信回数を削減することにより通信の効率化が見込める．またメッセージの送信に必要なヘッダを一つにまとめることが可能となり，メッセージの実データの割合が高まり処理系から見たバンド幅が向上する．しかし，レイテンシが低下し後続タスクの実行が遅れるという問題がある．従ってバンド幅とレイテンシを両立可能なバッファリングサイズを決定する必要がある．

4.3 ルーティング最適化の適用

処理系の階層化によって大規模なホスト群を管理することが可能となる．しかしホスト数が増加したことによって，ホスト間通信の際に階層モデルで経路上にあるホストすべてを中継すると通信効率が大きく低下する．従って，通信相手のホストへ効率良くメッセージを送信するためには中継ホストの選択が重要となる．ルーティング最適化ではホスト間通信が発生したとき，処理系全体に対して可能な限り通信回数や通信量を削減する．

ここで送信ホスト H_s から受信ホスト H_r への通信する際のルーティング最適化について説明する．中継ホストの集合を R=$\{H_{r1},...,H_{rn}\}$，ホスト H_i に対し先祖関係にあるホストの集合を A(H_i) とする．このとき，ルーティングには大きく分類して以下の 4 パターンが存在する (図 4.9).

(i) $R = \emptyset$

(ii) $A(H_s) \cap R \neq \emptyset$, $A(H_r) \cap R = \emptyset$

(iii) $A(H_s) \cap R = \emptyset$, $A(H_r) \cap R \neq \emptyset$

(iv) $A(H_s) \cap R \neq \emptyset$, $A(H_r) \cap R \neq \emptyset$

送信ホストと受信ホスト間に通信を阻害するファイアウォール等が存在しないとき，中継ホストを省略することが可能である．図 4.8 において，ホスト A から H に通信が発生した場合を例に説明する．図のようにファイアウォールが存在しホスト D の下位ホストへ外部から直接通信できない場合，ホスト A から H への直接通信は不可能である．このとき通信効率を考慮しない単純なルートは A，B，C，D，F，H となる．しかし，ここでホスト A から D への直接通信は可能である．ホスト D から H も同様に直接通信が可能なため，通信効率を考慮したルートは A，D，H となり，ホスト B，C，F との通信を省略することができる．これにより中継ホスト数を削減し通信の効率化を図ることができる．

ルーティング最適化にあたり，タスク間通信のパターンやホスト間の直接通信の可否等の静的情報をパラメータとして用いる．また，一定期間内に発生した通信回数と通信サイズ等の動的情報もパラメータとして扱うことが可能である．

4.4 一対一のホスト間通信

対象ホスト間の直接通信が可能な場合，ルーティングはパターン i となる．これは一対一のホスト間通信において直接通信が最も効率的な通信となるためである．

対象ホストとの直接通信が不可能な場合，ルーティングはパターン i の次に中継ホストの数が少ないパターン ii・iii となる．しかし，送信ホスト H_s とホスト $A(H_s)$ 間及びホスト $A(H_r)$ と受信ホスト H_r 間にファイアウォール等が存在するときパターン ii・iii は通信ができない可能性がある．その場合ルーティングは確実に通信が可能なパターン iv となる．

4.5 一対多のホスト間通信

マルチキャストを効率的に利用することを考慮し，出力端を持つホストの所属するホストグループの代表であるサブマスターホストにメッセージを送信する．メッセージを受け取ったサブマスターホストは子孫に出力端を持つすべての直下のホストにメッセージをマルチキャストする．

図 4.8 を例に説明する．このとき，代表出力端を持つホスト A は入力側タスクのメッセージを受信しているとする．ホスト A は出力端を持つ受信ホスト G，H，I，J の所属するホストグループ群の代表であるサブ

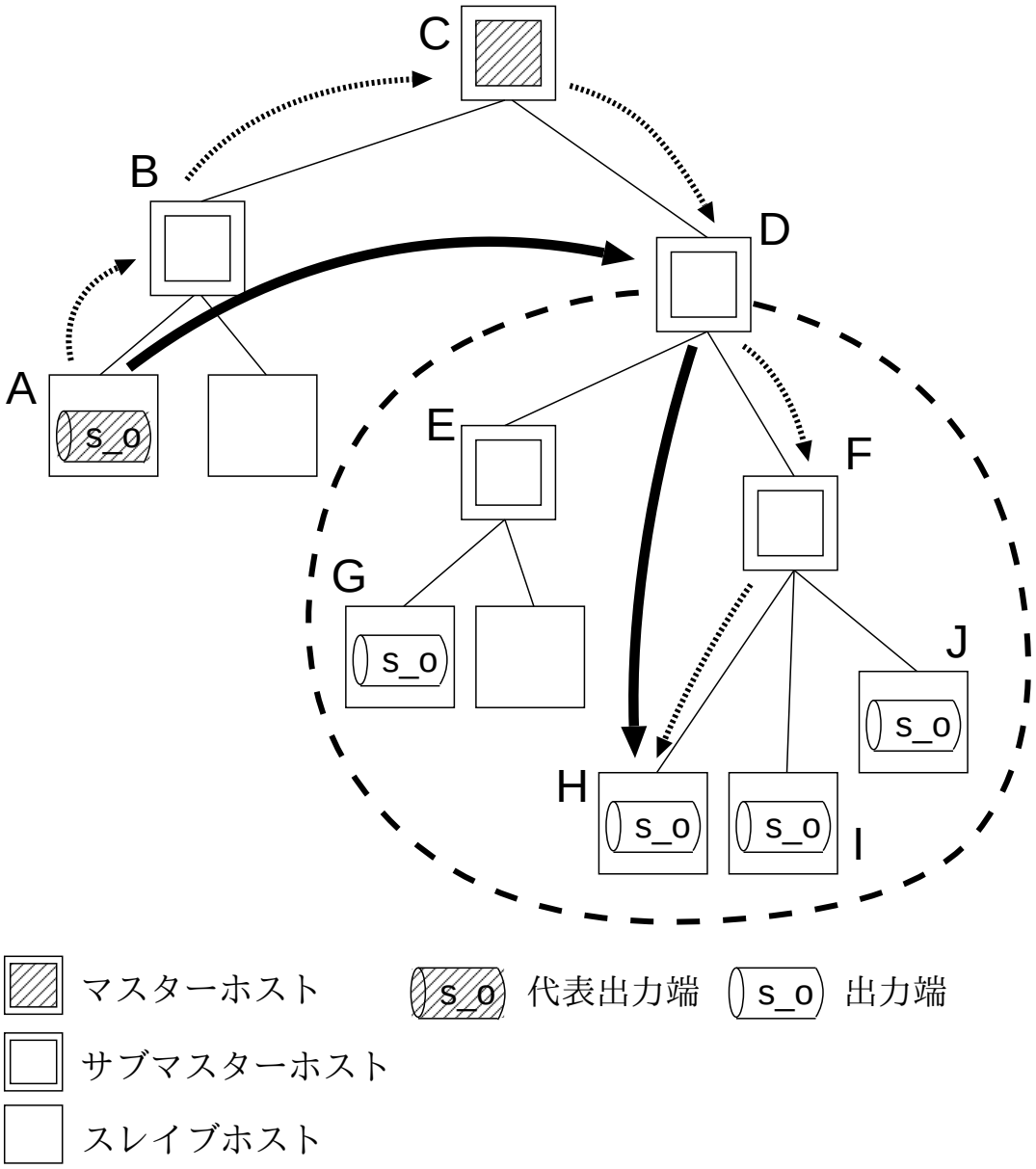

図 4.8: ルーティング最適化における中継ホストの省略

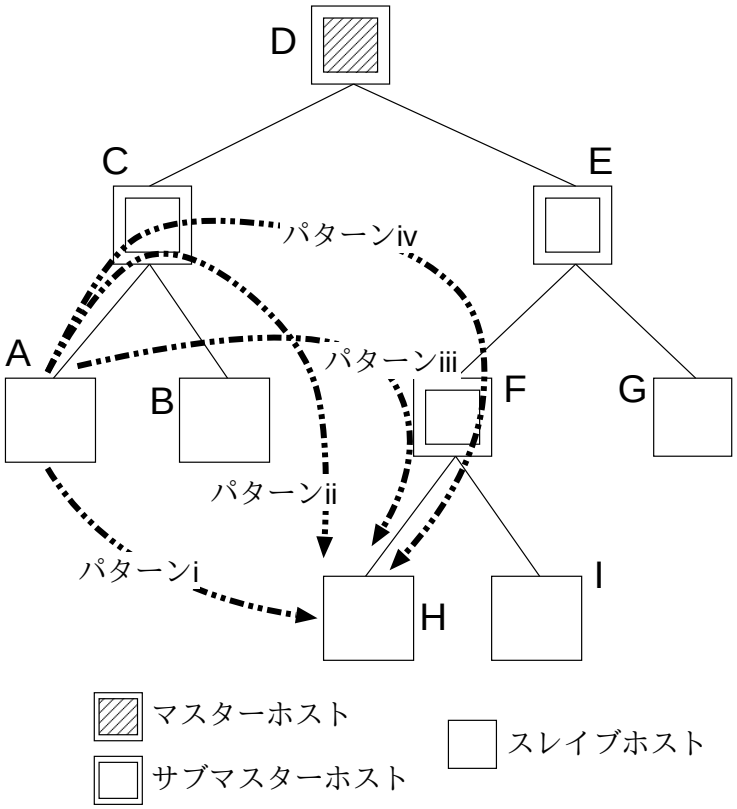

図 4.9: ルーティングのパターン

マスターホスト D にメッセージをマルチキャストする．メッセージを受け取ったホスト D はホスト A と同様に次にマルチキャストを行うホストを決定するが，ここでホスト E を代表するホストグループには受信ホストが G だけであるため，ホスト D はホスト G，F にメッセージをマルチキャストする．そしてメッセージを受け取ったホスト F は出力端を持つすべての下位ホスト H，I，J にメッセージを送信する．以上のようにして一対多のホスト間通信を実現する．

4.5.1 多対一のホスト間通信

多対一のホスト間通信では直接通信の可否はルーティングに大きな影響を与えず，また，マルチキャストを利用することが出来ない．一方，通信バッファリングを利用することは可能である．まず最初に，入力側タスクを持つホストは所属するホストグループの代表であるサブマスター

ホストにメッセージの送信を行う．メッセージを受け取ったサブマスターホストはバッファリングを行う．そして，代表出力端を持つホストに直接通信が可能であればそのままメッセージを送信する．直接通信が不可能であれば 4.3 節で述べたルーティングを行う．

5　実装

5.1　ホスト構成情報の取得

提案手法の実装には，他のホスト群とそれらの関係を表すホストの構成情報が必要となる．ホスト構成情報はユーザがあらかじめ用意したホスト名とホスト間の親子関係を記述したファイルを読み込むことで取得する．情報の保持には階層構造を持つ双方向リンクを使用する．各ノードは各ホストと対応しており親ノード及び子ノードへのポインタを持つ．ファイアウォールによる直接通信の可否情報も同様にユーザが記述したファイルを読み込み取得する．直接通信の可否情報は接続行列を使用する．

実行環境が大規模になると，各ホストが自分以外の全ホストの構成情報をあらかじめ保持することが難しくなる．そこで今後はホスト構成情報をホストグループ内のホストの集合情報として扱い，ホスト管理をホストグループ単位で行う必要がある．また，直接通信の可否情報はあらかじめ送受信テストを行うなどして自動的に取得することが考えられる．

5.2　マルチキャスト

マルチキャストは一対多のホスト間通信，すなわち代表出力端を持つホストから出力端を持つホスト群へメッセージを送信するときに適用する．まず，送信ホストは出力側タスクがどのホストに配置されたかをスケジューリング履歴を利用して探索する．出力端を持つすべてのホストを特定したら関数 getMulticastHost(hostTree_t* tr, int thr) を呼び出す．

getMulticastHost は第一引数にノードへのポインタ tr，第二引数に閾値 thr を取り，メッセージの送信先ホストの ID リストを返す．getMulticastHost は葉ノードであるスレイブホストから上位の親ホストに向けて再帰的に次のメッセージの送信先ホスト群を決定する．各ノードの管理している受信ホストの数が thr を上回る場合，そのノードを次のメッセージの送信先ホストとしてリストに加える．送信ホストはリストを基にメッセージを送信し，メッセージを受け取ったホスト群はこの処理を繰り返す．

5.3　ルーティング最適化

ルーティング最適化ではホスト構成情報及び直接通信の可否情報を参照し，送信ホストが受信ホストと通信するため次にどのホストにメッセー

ジを送信すればよいか決定する関数 searchNextHost(int ids, int idr) を呼び出し決定する．

関数 searchNextHost は第一引数に送信ホストの ID，第二引数に受信ホストの ID を取る．searchNextHost は再帰的に送信ホストから受信ホストまで辿り，受信ホストを含む全ての中継候補となるホストの ID のリストを返す．送信ホストはリストから逆順に ID を取り出し，最初の直接通信が可能なホストを次の送信先ホストとする．

5.4 バッファリング

通信データのバッファリングの実装について説明する．まず，ホストに配置されたストリーム毎に最初に実行されるタスクの出力量を取得する．この出力量が閾値を下回る場合，バッファリングを適用する．

現在の閾値は予備実験で求めた値を静的に使用しているが，今後は実環境における LAN・WAN 通信といった通信対象ホスト間のネットワーク情報，同一ストリームに属するタスクの性質から得られる実行タスクの出力頻度や後続タスク情報を利用して動的に決定する必要がある．

6　評価

本章では，実装した提案手法を評価するために行った実験について示す．評価には，CPU に AMD Athlon(tm) II X2 B24 Processor（800MHz），2GByte のメモリを搭載した計算機を 28 台使用した．

実験に使用するホストは図 6.10 に示すように構成した．太枠はファイアウォールを表しており，外部のホストから太枠内のホストへの直接通信を許可しない．破線内のホスト間には LAN 通信，破線外のホスト間には WAN 通信が使用される．本実験では UNIX の tc コマンドを使用し，帯域制御を行うことによって擬似的に WAN 通信の環境を構築した．

6.1　評価方法

評価には一対多と多対一のタスク間通信が発生する 2 つのワークフローを用いた．入力端側タスクとして，起動直後に合計 30MByte の文字列をプリントする計算量 0 のタスクを使用し，出力端側タスクには入力を受け取るだけの計算量 0 のタスクを使用した．また，ホスト構成については図 6.10 のように二種類のパターンで実行し，スレイブホストの数 N を 2，4，8 と変えて実験を行った．この条件において，提案手法の適用時と非適用時の実行時間を計測した．

6.2　評価結果

表 6.1 は一対多のタスク間通信の評価結果である．表 6.1 の評価結果から，ルーティング最適化を適用させた場合，パターン B の実行環境におけるワークフローの実行時間はパターン A の実行環境を用いた場合と比較して大幅に減少した．これはパターン A の実行環境にはファイアウォールが存在するので，ルーティング最適化によって中継ホスト数を大幅に減らすことができなかったためである．一方，パターン B の実行環境ではファイアウォールが存在しない．従って，ルーティング最適化によって送受信ホストが直接通信を行ったため，ワークフローの実行時間を大幅に削減することができた．

表 6.2 は多対一のタスク間通信の評価結果である．表 6.2 の結果から，マルチキャストとルーティング最適化を適用させた場合，パターン A の実行環境におけるワークフローの実行時間はパターン B の実行環境を用

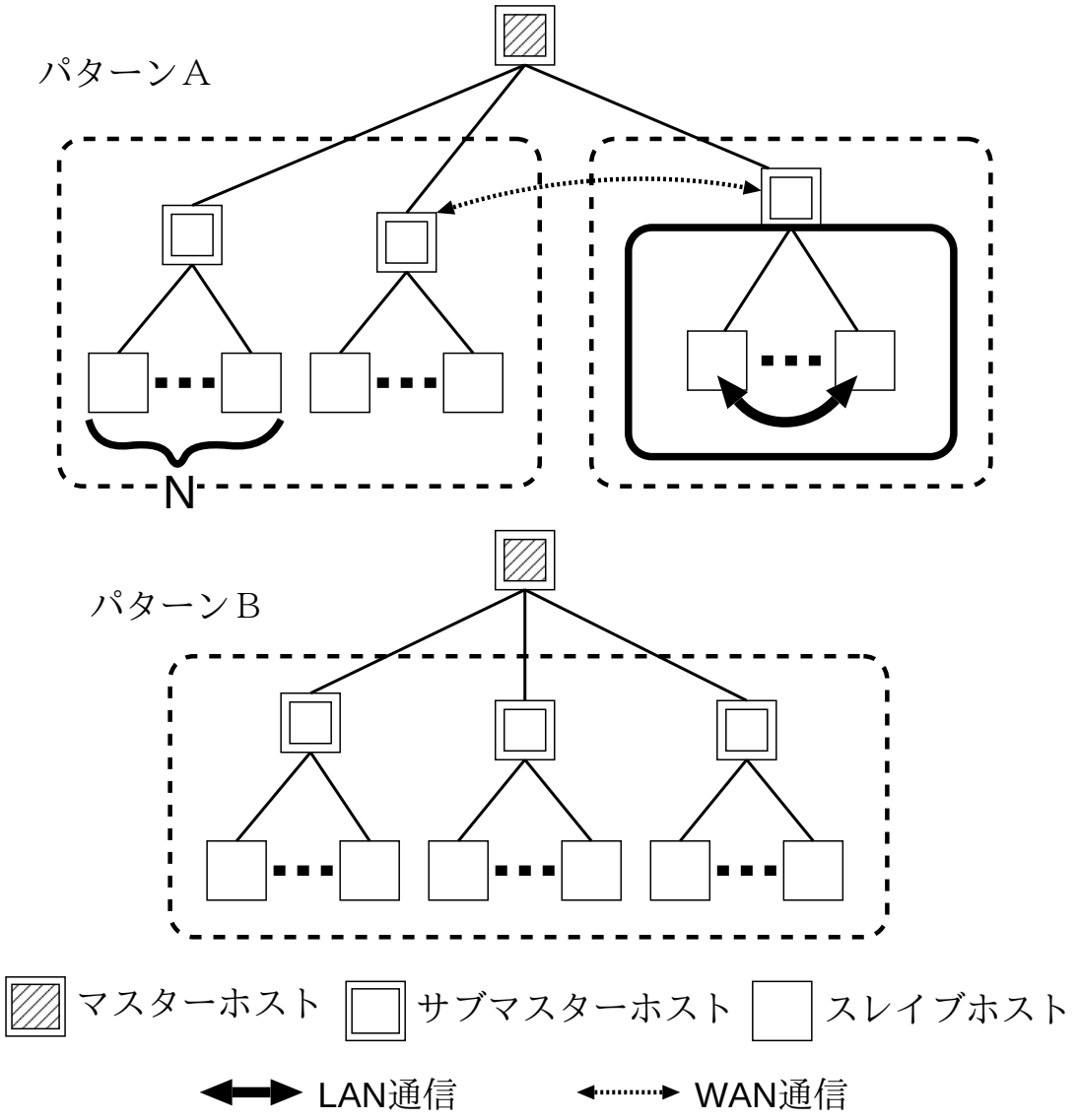

図 6.10: 評価環境

表 6.1: 多対一のワークフローの実行時間 (s)

ホスト構成	N	非適用	ルーティング最適化	全て適用
パターン A	2	191.00	184.41	13.28
	4	366.0	341.66	25.45
	8	761.680	714.15	49.82
パターン B	2	195.3	52.73	4.01
	4	352.9	53.33	5.078
	8	732.13	56.58	8.54

表 6.2: 一対多のワークフローの実行時間 (s)

ホスト構成	N	非適用	マルチキャストと ルーティング最適化	全て適用
パターン A	2	389.68	252.94	9.08
	4	708.772	502.1	17.99
	8	1211.3	894.41	33.66
パターン B	2	349.6	304.9	12.33
	4	646.21	502.14	19.40
	8	1134.66	956.26	31.07

いた場合と比較して大幅に減少した．パターン A の実行環境には低速な WAN 通信と高速な LAN 通信が混在しているため，マルチキャストを行うことで低速な WAN を用いた通信の回数を減らせたためである．

また，表 6.1，表 6.2 の結果から，パターン A とパターン B の両方の環境においてバッファリングを含めた全ての手法を適用した場合，ワークフローの実行時間は大幅に減少した．従って，提案手法の実装によって処理系全体の通信効率は向上したといえる．

7　おわりに

本論文では，現在実装中の階層型 MegaScript 処理系におけるホスト間の効率的な通信方式を提案し，MegaScript 処理系の通信パターンに合わせて既存の通信効率化手法の適用について検討し，実装を行った．評価では，比較的小規模なタスクを計算機 28 台上で実行し，提案した手法の有用性を示した．

今後の課題としては，タスクの総通信回数や総通信量を記述したメタ情報やプログラム実行中に得られるタスクの通信傾向等の動的情報を利用した通信方式を開発する予定である．また，大規模な実環境下において実ワークフローを用いて性能評価を行う必要がある．

謝辞

本研究を行うにあたり，御指導，御助言頂きました大野和彦講師，並びに多くの助言を頂きました近藤利夫教授，佐々木敬泰助教に深く感謝致します．また，様々な局面にてお世話になりました研究室の皆様にも心より感謝いたします．

参考文献

[1] K. Taura, T. Matsuzaki, M. Miwa, Y. Kamoshida, D. Yokoyama, N. Dun, T. Shibata, C. S. Jun and J. Tsujii.: *Design and Implementation of GXP Make – A Workflow System Based on Make. eScience*, IEEE International Conference on, 214-221, (2010).

[2] E. Deelman, G. Singh, M. Sua, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob and D. S. Katz.: *Pegasus: A framework for mapping complex scientific workflows onto distributed systems*, Scientific Programming. 219-237, (2005).

[3] M. Wilde, M. Hategan, J. M. Wozniak, B. Clifford, D. S. Katz and I. Foster.: *Swift: A language for distributed parallel scripting*, Parallel Computing. (2011).

[4] S. Altschul, W. Gish, W. Miller, E. Myers and D. Lipman: *Basic local alignment search tool*, *Journal of Molecular Biology*, Vol. 215, pp. 403–410, (1990).

[5] D. J. Lipman and W. R. Pearson: *Rapid and sensitive protein similarity searches.*, *Science*, Vol. 227, pp. 1435–1441, (1985).

[6] J. D. Thompson, D. G. Higgins and T. J. Gibson: *CLUSTAL W:improving the sensitivity of progressive multiple sequence alignment through sequence weighting*, position-specific gap penalties and weight matrix choice, *Nucleic Acids Reseach*, Vol. 22, pp. 4673–4680, (1994).

[7] E. Deelman, D. Gannon, M. Shields, and I. Taylor.: *Workflows and e-science: An overview of workflow system features and capabilities*, Future Gener. Comput.Syst., 25:528-540, (2009).

[8] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau and J. Myers.: *Examining the challenges of scientific workflows*, Computer, 40:24-32, (2007).

[9] 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, 73-76, (May 2003).

[10] 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, In 情処研報 2004-HPC-99, pages 7-12, July, (2004).

[11] 湯山 紘史, 大塚 保紀, 西里 一史，大野 和彦, 中島 浩: タスク並列スクリプト言語 MegaScript によるタスクモデルの記述手法, 先進的計算基盤システムシンポジウム SACSIS2004, 135-136, (May 2004).

[12] 阪口 裕輔, 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処理系におけるユーザレベル機能拡張機構, 情報処理学会論文誌 コンピューティングシステム, Vol. 47 No. SIG 12(ACS 15), pp. 296-307, (September 2006).

[13] K. Ohno, A. Mita, M. Matsumoto, T. Sasaki, T. Kondo, H. Nakashima: *Efficient Implementation of Large-scale Workflows based on Array Contraction*, Proceedings of the 22th IASTED International Conference on Parallel and Distributed Computing and Systems –PDCS 2010–, pp. 153-162, (November 2010).

[14] 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, 251–252, (2005).