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

2006－HPC－107 (20)
2006／8／1

大規模P2Pグリッドでの大容量コンピューティングのためのスケーラブルなプログラミング環境の検討

中島佳宏[†,††] 佐藤三久[†] 建部修見[†]

大規模 P2P グリッドでの大容量コンピューティングのためのプログラミング環境について検討を行う．P2P グリッドとは，Peer-to-Peer 環境上の volatile な計算機をも含めて動的な計算環境を，グリッドと同じように計算資源の co-allocation をして活用し，安定した計算を行う環境をさす．また，大容量コンピューティングとは，多数のジョブ，大量のデータを扱うような計算処理である．この数万台規模の広域に分散された動的に変化する計算環境上で，計算インテンシブな処理やペタスケールの分散するストレージデータを活用するデータインテンシブな処理が必要な多数のジョブを実行可能なプログラミングモデルについて提案を行う．これまでのプログラミングモデルを調査し，大容量コンピューティングに合うプログラミングモデルについて検討する．そして，グリッド向けの並列分散ファイルシステム上に大規模 P2P グリッド大容量コンピューティングを支援するためのプログラミングモデルを構築することを提案する．このフレームワークでは，処理する対象のデータに対して大域的かつ統一的な操作を行う高階関数である map, reduce, filter, scan を用いてそれぞれのデータへの処理およびフローを記述する．

Design of a scalable programming environment for large capacity computing in a large P2P grid environment

YOSHIHIRO NAKAJIMA,[†,††] MITSUHISA SATO[†] and OSAMU TATEBE [†]

In this paper, we propose programming model and environment for large capacity computing in a large P2P grid environment. "P2P grid" is a grid environment which makes use of not only volatile computing nodes in peer-to-peer environment as well as conventional clusters, and provides a reliable calculations mechanism like a grid. Also "large capacity computing" is a kind of computing that can deal with a large number of jobs and a large amount of storage data. We propose a model that allocates computing resources in P2P grid environment and supports users to execute large number of jobs that includes both computation-intensive processing and data-intensive processing, which utilizes large amount data, on tens of thousands of nodes distributed geographically. In order to explore a suitable programming model for large capacity computing, we examine existing works on programming model for that. As one solution, we design a framework and programming model to support large capacity computing on global distributed file system. In this framework, the user can describe control flows and processing using higher order operation, such as map, reduce, filter and scan, in order to process general and global operations to the data.

1. はじめに

インターネットをはじめとする広域ネットワークや分散コンピューティング技術の進歩により，ネットワーク上の計算資源や大規模データ処理，並列コンピューティングの支援を可能とするグリッド技術が注目されている．また近年，企業などで，製薬のための分子構造設計や電子回路設計のシミュレーションなど，莫大な量の計算を迅速に処理するニーズが非常に高くなっている．

これらの計算は，大量の計算インテンシブな処理を扱う，大規模または分散しているデータストレージを扱うようなデータインテンシブな処理が必要である．さらに，計算資源は地理的に分散して存在している．

現在これらの計算を行うためには，グリッド向けに拡張された MPI や Grid RPC を用いて並列アプリケーションを作成する方法，多数のデータセットと並列に計算を行える処理をバッチとしてスケジューラに登録し大規模実行させる方法がある．

今後計算環境として，データセンタなどで良く管理されている計算資源だけはなく，家庭やオフィスにある PC なども計算資源として利用していくことが考えられる．これらの計算環境を考慮すると，既存の

† 筑波大学大学院システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba
†† 日本学術振興会特別研究員
Research Fellow of the Japan Society for the Promotion of Science

MPI や RPC といったプログラミングモデルでアプリケーション作成を行うのは難しく，広域に大規模かつ Volatile な計算機環境かつスケーラブルなアプリケーション作成のためのプログラミングモデルが必要である．また，統一的なプログラミングモデルから計算インテンシブな計算とデータインテンシブな計算の両方の処理をおこなえるほうが好ましい．

本稿では，大規模 P2P グリッドでの大容量コンピューティングのためのスケーラブルなプログラミングの検討を行う．

2 章で，大規模 P2P グリッドでの大容量コンピューティングの定義を行い，求められるプログラミングモデルの要件をまとめる．3 章でこれまで大規模なグリッドコンピューティングを行なうために提案されてきたプログラミングモデルを調査し，それぞれのプログラミングモデルの比較・検討を行なう．4 章で大規模 P2P グリッドでの大容量コンピューティングのためのスケーラブルプログラミングモデルのの提案を行なう．終章でまとめと今後の課題を述べる．

2. 大規模 P2P グリッドでの大容量コンピューティングとは

P2P グリッドとは，これまでのグリッド環境における遠隔のサーバやクラスタなどの計算資源だけではなく，Peer-to-Peer 環境における PC などの計算資源も含めて計算資源を co-allocation して活用する計算である．

大規模 P2P グリッド環境での大容量コンピューティングのコンセプトは，Peer-to-Peer グリッドから提供された計算機を集結し，数万台規模の広域に分散された動的に変化する計算環境上で，計算インテンシブな処理や分散するペタスケールのストレージデータを活用するデータインテンシブな処理が必要な多数のジョブを実行させる計算である．

この大容量コンピューティングでは，大規模で複雑な並列プログラムではなく，全体的にフローが簡単なプログラムを用いて，多数のジョブまたは大量のデータに対して処理するアプリケーションに着目する．これは，グリッド環境上で実行されているアプリケーションは，比較的簡単な処理を大量のパラメータに対して処理するアプリケーションであり，複雑な通信を行うようなアプリケーションは少ないからである．

このようなプログラムでは，それぞれプロセスの個々の処理を明示的に記述するのではなく，全処理や全データに対して大域的で統一的な操作を記述するようなワークフローの機構が必要である．さらに，処理するデータセットが大量に存在，または処理するデータ量が莫大であるため効率よく処理するためには，数万台でも有効に活用可能なスケーラブルな機構が求められる．

P2P グリッド環境では計算機環境が動的に変化かつ大規模であるが考えられ，これらの不安定な計算資源を用いて，安定した計算を実現することが求められる．しかし，現在のグリッドコンピューティングでは，地理的に分散かつよく管理されている計算機環境を対象としており，ノードの故障は考慮されているが，動的に変化する計算環境は考慮されていない．

この大容量コンピューティングを支援するためのスケーラブルなプログラミングモデルでは以下のような要件が必要である．

- データインテンシブな処理や計算インテンシブな処理に対して，グローバルで統一に操作を行う機構 (データに対する統一的な処理の記述が可能な機構)
- 簡便にスケーラブルなプログラムを記述可能
- 広域に分散しかつ動的に変化する計算資源を有効に活用可能
- 耐故障性や負荷分散の処理を意識せずにプログラミング可能

3. 大容量コンピューティングのためのプログラミングモデルの関連研究

この章では，広域グリッド環境において並列に大容量コンピューティングをサポートするという観点からのプログラミングモデルの関連研究について述べる．

3.1 並列プログラミングを支援するためのプログラミングモデル

3.1.1 グリッド環境向けのメッセージパッシングに基づくプログラミングモデル

メッセージパッシングモデルでは各プロセスが send/recv を用いて通信を行ない計算を行なう．ユーザは各プロセスの通信やプロセスの担当する計算を明示的に記述する．ここではプログラミングモデル例として，MPI と Phoenix をあげる．

MPI は分散メモリ型並列処理のメッセージパッシングのライブラリの規格であり，コレクティブ通信を効率よく実行可能な機構をもち，クラスタ計算機で広く用いられている．この MPI をグリッド環境向けに，広域通信やセキュリティーを考慮した実装として，MPICH-G2[6] や GridMPI[14] があげられる．大規模なクラスタ環境で利用していたプログラムをそのままグリッド環境上で大規模実行できるという利点がある．基本的に動的な資源増減や故障に対応することは難しい．さらに，MPI から並列ファイルシステムの I/O を効率よく扱うためのインターフェースとして MPI-IO が提案されている．

Phoenix[11] は広域ネットワーク上の動的に計算機資源構成が変化する環境でメッセージパッシングモデルをもとに，動的にノードが計算に参加/離脱する環境に対応可能なプログラミングモデルを提供するフレームワークである．Phoenix では，アプリの状態の透過的なマイグレーションや広域 WAN 環境での透過的な通信が行なえるようになっている．また，動的な資源環

境を考慮して設計されているため，耐故障性のあるアプリケーションを作成することができる．ただし，ユーザが各計算状態の移送を行ないながら動作するコード記述するのは難しい．

3.1.2 グリッド環境向け RPC

RPC は，関数の引数と実行する手続きを実行するためにクライアントが遠隔のサーバに対してメッセージを転送するプログラミングモデルである．基本的に RPC は並列処理のために設計されていないが，非同期 RPC を用いることで並列プログラムを実行させている．マスターワーカ型のプログラミングに適しており，グリッド向けのアプリとして良く見られるようなパラメータスイープ型のプログラムを記述するために RPC が用いられることが多い．遠隔地のクラスタに対し効率よく並列に RPC を行なうシステムの実装として Ninf-G2[9], OmniRPC[8] が挙げられる．

RPC のモデルでは 1 対 1 の通信を行なうため必然的にクライアントへの通信集中が起こり高いスケーラビリティーを保証することは難しい．また多数のプロセスの協調といったことも同じく難しい．Ninf-G のグループでは，遠隔データ通信を伴う場合には GridRPC，RPC 呼び出し先では MPI をそれぞれ用いてアプリケーションを作成し，スケーラビリティの向上と動的な計算環境へ対応した[15]．しかし，このように異なるプログラミングモデルを用いてプログラミングを行うのはプログラマの負担が大きい．

3.2 共有メモリに基づくプログラミングモデル

共有メモリに基づくプログラミングモデルとしてタプルスペースと JuxMem がある．プログラミングは共有メモリ空間に対して read/write するコードを記述すれば良く，さらに通信やプロセスを意識せず簡便に行うことができる．

3.2.1 タプルスペース

タプルスペース (TS) は共有メモリに類似した空間に，それぞれのプロセスがタプルを出し入れすることで通信を行なうプログラミングモデルである．TS を実装したシステムとして，Linda[3],JavaSpace[12] がある．TS ではプロセスを意識せずプログラミングできるため，動的に資源が変化する環境に適したプログラミングモデルである．また，TS は単純な共有メモリをソフトウエアで実現したものであるため，共有メモリの性質を継承し，タプル空間を管理するサーバがボトルネックになってしまう．タプルサーバがボトルネックにならない実装は難しく，さらに通信はタプル空間を介しておこなうため効率が悪い．

3.2.2 JuxMem

JuxMem[2] は，JXTA[5] ミドルウエア上に分散共有メモリに似たデータ共有サービスを提供するフレームワークである．JuxMem は永続的で排他的な共有データに対して透過的なアクセスを提供する．動的に資源が変化する環境を考慮してシステムが作成されており，ノードの参加/離脱についての処理やデータのレプリケーションを行い耐故障性を保証する．また，ノードのネットワークトポロジを考慮した階層的なデータ配置管理を行うことにより，データ空間への効率の良いアクセスを狙う．

3.3 パラメータサーチ向けの処理を支援するプログラミングモデル

パラメータスイープ型のアプリケーション作成を支援するフレームワークとして Condor-MW[7] や Nimrod[1] がある．これらのシステムでは，全体のフローを記述するプログラム，リモートで実行するプログラムを記述する．システムはタスクの分配や計算機の障害への処理をおこなうため，ユーザはこれらの処理を意識せずプログラムの作成を行なえる．

3.4 データインテンシブなアプリケーションに対するプログラミングモデル

3.4.1 MapReduce

MapReduce[4] は，Google File System 上で動作する関数型プログラミングからヒントを得た大規模データ解析のためのプログラミングモデルである．ユーザは *key*/*value* の組のデータを処理する map 関数と，そこから生成される同じ *key* 値の *value* を処理する reduce 関数を記述する．プログラムは，実行時にシステムはデータの構成から並列に動作するジョブを生成し，大規模クラスタ計算機でファイルが存在するノードにプロセスを割り当て実行する．システムは入力データの分割，ジョブのスケジューリング，ファイル I/O や耐故障性についての処理を自動的に行なうため，ユーザはシステムの動的な資源変化，通信やプロセスを意識せずにプログラムを記述できる．しかし，各プロセスが通信し合うような計算は考慮されておらず，また広域ネットワーク環境を意識した設計はされていない．

3.4.2 Gfarm ファイルシステム

Gfarm ファイルシステム[10] は広域ネットワーク環境を意識して作成された並列分散ファイルシステムで，ペタバイトを越える大容量，大規模データ処理可能なスケーラブルなアクセス性能を実現している．基本的に Gfarm はファイルが格納されているノードでプログラムを実行し，ファイル断片が分散している場合には並列実行もサポートしている．. レガシーアプリケーションをそのまま実行出来る利点もある．

3.5 大規模 P2P グリッドでの大容量コンピューティングのためのスケーラブルなプログラミングモデルの検討

プログラミングモデルの比較を，計算またはデータインテンシブな処理に関してそれぞれ**表 1** と**表 2** に示す．

大容量コンピューティングでの大量のデータの処理や多数のジョブを統一的に扱うような処理では，データの通信とプロセスの管理またプロセスとデータの割り当てはシステムが行い，ユーザは大域的かつ統一的

表 1 計算インテンシブな処理を支援するプログラミングモデルの比較

Items	MPI	Phoenix	RPC	TupleSpace	JuxMem
並列度	データ並列	データ並列	タスク並列	タスク並列	タスク並列
スケーラビリティ	-10^6	-10^3	-10^3	-10^2	-10^3
プロセスの割り当て	コードに明示的に記述	コードに明示的に記述	コードに明示的に記述	システムが管理	システムが管理
データ通信の記述	明示的に send/recive	明示的に send/recive	関数の引数	TS 空間への読み書き	共有空間への読み書き
データのプロセスへの割当て	コードに明示的に記述	コードに明示的に記述	コードに明示的に記述	システムが管理	システムが処理
コレクティブ通信機構	Yes	Yes	No	No	No
並列ファイルアクセス	MPI-IO を使用	–	–	–	–
負荷分散	明示的にコードに記述	明示的にコードに記述	明示的にコードに記述	システムが管理	システムが管理
計算資源	static	dynamic	can be dynamic	dynamic	dynamic
Fault tolerance	poor	good	better	better	good
		(コードに明示的に記述)	(クライアント故障に対処不可)	(サーバ故障に対処不可)	(システムが対処)
プログラミングの容易さ	難しい	難しい	比較的容易	比較的簡単	比較的簡単

表 2 データインテンシブな処理を支援するプログラミングモデルの比較

Items	MPI-IO	MapReduce	Gfarm
並列度	データ並列	データ並列	タスク並列
スケーラビリティ	-10^2	-10^3	-10^3
処理の対象	MPI のユーザ定義データ型の配列	key/value ペアの集合	断片ファイル
プロセスの割り当て	コードに明示的に記述	システムが管理	システムが管理
データ通信の記述	明示的に send/recve	ファイルに key/value ペア値を書き込む	システムが処理
ファイルのデータへのアクセス方法	部分ファイル	key/value ペアで指定	断片ファイル
データのプロセスへの割当て	コードで明示的に記述	システムが処理	システムが処理
計算環境の Volatility	考慮されていない	考慮されている	考慮されている
データのコレクティブな機構	Yes	map/reduce	プログラムの並列実行のみ
並列ファイルアクセス	Yes	Yes	Yes
負荷分散	明示的にコードに記述	システムが管理	システムが管理
ノードの故障	考慮されていない	システムが対応する	システムが対応する
プログラミングの容易さ	難しい	比較的簡単	比較的簡単

な操作を行う高階関数を用いて個々のデータへの処理を記述する方がより簡便にプログラム可能である．これはプログラミングモデルにおいて，プロセスとデータの割り当てや通信，処理の並列性を明示的に記述しないことで容易にプログラムを記述することが出来る．さらに，データの扱いも，低レベルなデータを操作するのではなく，抽象化されたデータ集合への操作を記述可能なほうが，より抽象度が高いプログラミングが可能となり，大容量コンピューティングに適している．

大規模な計算機環境を有効に使用するためには，プログラミングモデルに高いスケーラビリティが必要である．つまり，データインテンシブな処理を支援するプログラミングもでると同様に，データの局所性有効に活用し，データをローカルから読み込みローカルに書き出すようにし，高いスケーラビリティを実現する．さらに，プロセス間のデータ通信はシステムが最適化し，通信の集中をおさえるような機構が必要である．

大規模な P2P グリッド環境を考慮すると，動的な計算環境への対応と耐故障性は必要である．また，ユーザが動的な計算環境へ対応及び耐故障性のためのコードを記述するのは一般的に難しく，システムが対処しユーザには透過的に扱わせることが望ましい．また，特定のプロセスの故障によって，処理が中止しないような機構が必要である．

4. 大容量コンピューティングのためのスケーラブルなプログラミングを支援するフレームワーク

大容量コンピューティングのために，広域環境での大規模な計算環境において，多数のジョブ，大量のストレージを大域的にかつ統一的な操作を行うようなフレームワークが必要である．大容量のストレージを扱うため，我々は提案するプログラミングフレームワークを並列分散ファイルシステム上に実現する．これにより，データのレプリケーションやステージングなどをシステム側で処理することが出来，ユーザにはデータへのアクセスが透過的に出来るようになる．

さらに，大規模な P2P グリッド大容量コンピューティングを支援するためのプログラミングモデルとして，処理する対象のデータに対して大域的かつ統一的な操作を行う高階関数である map, reduce, filter, scan を用いて，それぞれのデータへの処理を記述し，フローを記述する．プロセスの割り当て，データ通信，動的な計算環境や耐故障性などの記述が難しい処理はフレームワークが行うようにする．これによって，ユーザは個々のデータへの処理のみを記述するだけで大容量コンピューティングが可能になる．

4.1 プログラミングモデル

提案するプログラミングモデルでは，処理するデータの対象を *key/value* の組の集合として扱う．またそれらのデータへの操作については，大域的かつ統一的な操作を行う高階関数に基づいた処理を行う map, reduce, filter, scan を用いて行う．大容量コンピューティングの対象とするアプリケーションでは，計算する内容はデータ並列に処理できることを想定しており，そのデータ並列の個々のデータを処理するデータ集合として map,reduce, filter, scan 操作を行い，それらの処理の出力を．入力データを *key/value* の組として処理する関数に渡し，計算を行い，結果のデータを *key/value* の組としてする．ただし，データの集合を *key/value* の組へ変換または，データの分割をする処理はユーザが記述する．

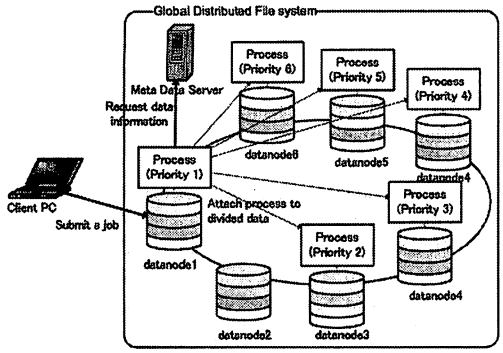

図 1 提案システムの概要

この方法を取ることによって，プロセス間のデータ通信やプロセスの割り当ての記述を必要としないプログラミングモデルとなり，プログラムの記述が簡便になる．また，スケーラビリティに関しては，処理する問題の並列性にもよるが，基本的にデータを大量の *key/value* の組ごとに分割が可能な問題を対象としているため，ほぼ独立に処理することができ高いスケーラビリティーが実現できる．

提案するプログラミングモデルでは，以下の操作を使用する．但し，f は単項演算子，op は二項演算子である．また，x_i は key/value の組を示す．

- map f $[x_1, x_2, \ldots, x_n] \rightarrow [fx_1, fx_2, \ldots, fx_n]$
- reduce op $[x_1, x_2, \ldots, x_n] \rightarrow x_1\ op\ x_2\ op\ \ldots op\ x_n$
- filter f $[x_1, x_2, \ldots, x_n] \rightarrow [x_1, x_3, \ldots, x_{2n+1}]$
  (ここで f は $x_1, x_3, \ldots, x_{2n+1}$ の時真を返す)
- scan op $[x_1, x_2, \ldots, x_n]$
  $\rightarrow [x_1, x_1\ op\ x_2, \ldots, x_1\ op\ x_2\ op \ldots op\ x_n]$

データレポジトリ空間として分散ファイルシステムを利用することにより，統一的な名前空間を利用可能にすることができる．また，分散ファイルシステムで複数のファイルをひとつの名前として定義できる機能を利用し，それぞれの *key/value* の値を保持するデータを個別のファイルとして，登録し，集合を作るようにする．

4.2 実行モデル

図 1 に提案システムの概要を示す．基本的に，クライアントから実行プログラムを提案システムに向けて実行を行うと，各プロセスが協調し，処理を進めていくこととなる．

ユーザはクライアントより分散ファイルシステム上にプロセスを起動させ，処理するデータセットを生成または指定を行う．map 関数を実行する場合には，生成したデータセットを保有するノードに，プログラムを起動させ，map 処理する関数に *key/value* の組のデータを提供し，計算，中間データをストレージに書き出す．

reduce 関数が実行される場合には，map 関数と同様にデータを保持するのノードを分散ファイルシステムに問い合わせ，システムが reduce 処理する関数にデータの 2 組の *key/value* のデータセットを提供し，計算，結果を出力する．filter,scan 関数も同様に *key/value* の組で処理を行う．

並列実行に関しては，map, reduce, filter, scan 関数を行う入力データセットをシステムが解析を行い，自動的に遠隔ノードに複数のプロセスを生成させ実行させる．基本的に提案システムでは，データの存在するノードでプログラムを実行するが，計算資源の計算能力，ストレージ容量やノードごとの負荷分散を考慮してプロセスのスケジューリング，データのレプリケーション，ステージング操作を行なう．

耐故障性については，システムがユーザにとって透過的に対処することとする．これを実現するために，各プロセスで処理の進行状況の共有とプロセスチェックポイントを利用する．プロセスに優先度をつけ優先度が高いプロセスがスケジューリングを行う方法をとり，そのスケジューリングを管理するプロセスが異常終了しても代替のプロセスがスケジューリングを引き継ぐようにする．また動的に変化する計算機環境についても，システムが管理を行い，ユーザはプログラムコードにおいてノードの増減を意識せずに処理を記述できるようにする．

4.3 プログラミング例

DNA やアミノ酸配列について最尤法に基づく系統解析を行う PAML[13] のコドン置換モデルとアミノ酸置換モデルに基づく分析を行なう **codeml** を提案システムを用いて並列実行させる．**codeml** の処理では系統樹の解析が大部分を占め，また系統樹ごとに並列に処理し，並列に処理した結果をひとつのファイルに書き出すことを行う．提案システムを用いると，系統樹の解析は map 関数，結果の集計は reduce 関数で記述することが出来る．**図 2** に本システムを利用したときのプログラムの記述，**図 3** に実行した際のフローを記述する．

処理する系統樹データを *key* に系統樹番号，*value* 系統樹データの組を作成し，グローバルなデータレポジトリに登録する．その登録したデータ集合に対して map 操作を行い，reduce 操作で結果をまとめ，ひとつのファイルに出力する．map 関数で呼ばれる *codeml_submit* 関数は *key/value* の組のデータが変数 *d* として与えられ，*key* は系統樹番号，*value* は系統樹のデータが設定されており，そのデータで解析を処理する．出力として *key* に系統樹番号，*value* に統計データの文字列を設定し，*imo* に書き出す．reduce 関数で呼ばれる *codeml_collect* 関数では，map 関数のので処理されたデータの *value* で設定された 2 つ文字列の結合の処理を行う．

5. 今後の課題

今後の課題としては，提案するプログラミングモデ

```
/* map 関数で呼ばれる関数，d: key/value を保持するデータ，
imo:key/value を出力する空間 */
int codml_submit(lc_data_t *d, imo_t *imo){
  char *no = (char*)getKey(d); /* 系統樹番号 */
  char *treedata = (char *)getValue(d); /* 系統樹データ */
  lc_data_t result; /* key の値は系統樹番号，value の値は統計デー
タの文字列 */
  codeml_init(); /* 初期化 */
  Forestry_remote(treedata, &result) /* 系統樹の解析 */
  commit(imo, result); /* 結果の出力 */
}

/* reduce 関数で呼ばれる関数，d1,d2: key/value を保持するデータ，
imo:key/value を出力する空間 */ */
int codeml_collect(lc_data_t *d1, lc_data_t *d2, imo_t *imo){
  lc_data_t result;
  setKey(result, getKey(d1));
  /* cat() は2つの文字列を結合する関数*/
  setValue(result, cat(getValue(d1),getValue(d2)));
  /* 出力は key: 番号，value: 2つの統計データを結合した文字列 */
  commit(imo, result)
}

int main(int argc, char **argv){
  lc_init(argc, argv);
  // TREE を file に分割し gfarm に登録（集合として
  // gfarm:/work/treedata とする）（tree の数:noTrees)
  // gfarm:/work/treedata/{0,1,2,...,199}
  create_files("tree_data", 200 "gfarm:/work/treedata");
  lc_map(codeml_submit, "gfarm:/work/treedata",
      "gfarm:/work/treeoutput", mopt);
  lc_reduce(codeml_collect, "gfarm:/work/treeoutput",
       "gfarm:/work/codemlout",ropt);
  lc_finalize();
}
```

図 2 提案システムを用いた codeml の疑似コード

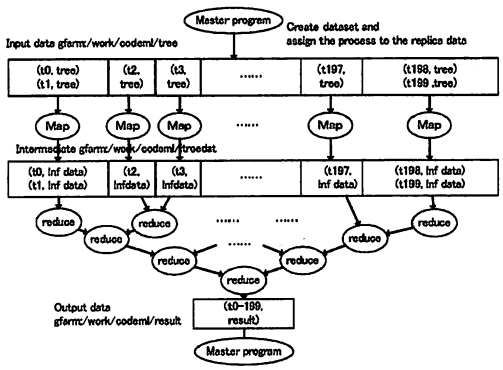

図 3 提案システムにおけるプログラムのフロー

ルをサポートするフレームワークを Gfarm 上に実装させ，大規模計算機環境において提案システムの基本的な性能評価を行ない，有効性を確認する予定である．また，現在の Gfarm は P2P 環境への対応していないが，対応することが検討されており，その際には提案システムの P2P 環境上の性能評価も行いたいと考えている．

謝辞 本研究の一部は，文部科学省科学研究費補助金 基盤研究 A 課題番号 172002, 特別研究員奨励費 課題番号 17・7324, および，日仏共同研究プログラム (SAKURA) による．

参 考 文 献

1) D.Abramson, R.Sosic, J.Giddy, and B.Hall. Nimrod: a tool for performing parametrised simulations usingdistributed workstations. pp. 112–121, 1995.
2) G. Antoniu, L. Bougé, and M. Jan. JuxMem: An adaptive supportive platform for data sharing on the grid. pp. 49–59, 2003.
3) N. Carriero and D. Gelernter. Linda in context. *Communications of the ACM*, 32(4):444–458, 1989.
4) J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Proc. OSDI 2004*, 2004.
5) L. Gong. Industry report: Jxta: A network programming environment. *IEEE Internet Computing*, 5(3):88–, 2001.
6) N. T. Karonis, B. R. Toonen, and I. T. Foster. MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface. *Journal of Parallel and Distributed Computing*, 63(5):551–563, May 2003.
7) J.Linderoth, S.Kulkarni, J.-P. Goux, and M.Yoder. An enabling framework for master-worker applications on the computational grid. In *Proceedings of HPDC9*, pp. 43–50, August 2000.
8) 佐藤, 朴, 高橋. OmniRPC:グリッド環境での並列プログラミングのための Grid RPC システム. 情報処理学会論文誌コンピューティングシステム, Vol. 44(SIG11 (ACS 3)):34–45, 2003.
9) Y. Tanaka, H. Nakada, S. Sekiguchi, T. Suzumura, and S.Matsuoka. Ninf-G: A Reference Implementation of RPC-based Programming Middleware for Grid Computing. *J. Grid Comput.*, 1(1):41–51, 2003.
10) O. Tatebe, Y. Morita, S. Matsuoka, N. Soda, and S.Sekiguchi. Grid Datafarm architecture for petascale data intensive computing. In *Processing of CCGRID2002*, pp. 92–100, 2002.
11) K. Taura, K. Kaneda, T. Endo, and A. Yonezawa. Phoenix: a parallel programming model for accommodating dynamically joining/leaving resources. In *Proceedings of PPoPP '03*, pp. 216–229, 2003.
12) J.Waldo, etal. Javaspace specification-1.0. Technical report, Technical report, Sun Microsystems, March 1998, 1998.
13) Z.Yang. Paml: a program package for phylogenetic analysis by maximum likelihood. *Computer Applications in BioScience*, 13:555–556, 1997.
14) 松田, 石川, 鐘尾, 枝元, 岡崎, 鯉江, 高野, 工藤, 児玉. GridMPI Version 1.0 の概要. Technical report, SWoPP2005, 8 月 2005.
15) 武宮, 田中, 中田, 関口. 大規模長時間実行 grid アプリケーションの実装と評価. In *SACSIS 2006*, pp. 351–358, 2006.