動的なノード群構成機構を備えた階層型グリッド環境

青木仁志[†] 中田秀基[††,†] 松岡聡[†,†††]

1. はじめに

複数の管理主体に属する計算資源を集合的に活用して大規模な計算を行うグリッドと呼ばれるシステムが普及しつつある。今後のグリッドにおける計算機資源としてはクラスタが有望であり、特にクラスタを複数個結合する形が、将来のグリッドとして一般的になると考えられる。このようなクラスタの各ノードは、ローカルの IP アドレスを持ち、ルータノードで提供される NAT を用いて外部と通信する場合が多くなると考えられる。また複数のクラスタを使用する環境ではノードの総数が数百数千台規模に達することも考えられ、これらのノードをフラットに管理することは、ルートとなるクライアントノードにかかる負荷を考えると現実的ではない。

これらを考慮すると今後のグリッド環境は階層構造をとらざるを得ないと考えられる。すなわち、クラスタの外部に構成されるネットワークとクラスタ内部のネットワークの 2 階層である。

このようなグリッドの階層構造に合致したプログラミング環境として Jojo[1] が提案されている。Jojo は柔軟な多階層実行機構を提供するが、そのトポロジーは実行前に静的に決定される。しかしグリッド環境では大規模・広域になればなるほど、その構成要素がダウンすることが頻繁に生じるため大規模・広域グリッド環境では実行時のノードの増減に柔軟に対応できる必要がある。また静的な情報による管理ではメンテナンスコストが非常に高く、ユーザへの負担が大きい。[2] そこで大規模グリッド環境下では動的なノード群構成が必要不可欠となる。

本稿では、動的なノード群構成機構を備えたプログラミング環境 Jojo 2 の設計・実装について述べる。Jojo2 はグリッドの階層構造に適合した動的なノード群の構成を行うことで、自律的なノードの発見、動的なノードの追加・削除を可能にする。また動的構成を前提としたプログラミング API を提供することで、ノードの参加・脱退に柔軟に対応したプログラミングが容易に可能となる。

† 東京工業大学
†† 産業技術総合研究所
††† 国立情報学研究所

2. 設計

2.1 システム構造

Jojo2 はグリッド環境の階層構造を反映したシステム構造をとる。Jojo2 ではクラスタ外のネットワークとクラスタ内のネットワークで異なるアプローチを用いて、そのトポロジーを構築する。まずクラスタ外のネットワーク、すなわち各サイトのログインノードなどに対しては静的なコンフィグレーションを用いたサブマスタプロセスの起動を行う。これはインターネット上での通信となるため、Globus や ssh を用いたセキュアな起動と通信を行う。

一方で、クラスタ内においては計算ノードは数百台数千台規模となることが考えられるため、静的な管理では非常にコストが高い。そこで UDP ブロードキャストを利用した動的なノード群構成を行うことで、自律的なノードの発見、動的なノードの参加・脱退を実現する。

2.2 プログラミングモデル

Jojo2 はレイヤによらず同じプログラミングモデルで実装可能なクラスフレームワークを提供する。各ノードのそれぞれで実行されるプログラムを記述することができるが、典型的には一つのレイヤで同一のクラスオブジェクトを実行する。

各ノードで実行されるクラスオブジェクトは、上位レベルのノード、下位レベルのノード群と通信可能なメッセージパッシング API が提供される。このとき下位レベルのノード群は動的に変化するため、特定子ノードとの一対一通信をサポートする機構は適さないと考えられる。そのため下位レベルとの通信はブロードキャストのみをサポートする。一方で上位レベルへの通信は一対一通信をサポートする。

ノードの動的な参加・脱退をサポートをするためには、それらに対する処理が容易に記述できる枠組みが必要である。そこで Jojo2 ではノードの参加・脱退に対するハンドラメソッドを提供する。ユーザはこのメソッドを実装することで、ノードの参加・脱退に対する処理を行うことができる。このハンドラメソッドは計算処理と別スレッドで実行するため、ノードの参加・脱退に対する処理を計算処理と独立して行うことが可能である。

3. 実　　装

3.1 動的なノード群構成

Jojo2 では UDP ブロードキャストを用いた自律的なノードの発見、動的なノード群の構成を行う。Jojo2 のネットワークトポロジーの構築について 2 層構造を例にして述べる。 (図 1)

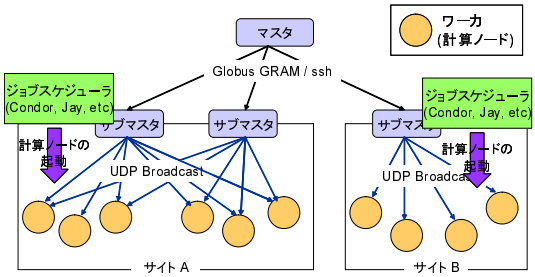

図 1　動的なノード群構成

(1) ユーザはマスタプロセスをクライアントノードで起動する。

(2) 次にユーザは各サイトのジョブスケジューラなどを用いて、各サイトの計算ノード上にワーカプロセスを起動する。

(3) マスタプロセスは設定ファイルに記述されたログインノードなどに対して、サブマスタプロセスを Globus GRAM あるいは ssh を用いて起動し接続する。

(4) サブマスタは自身に接続されたワーカの数を定期的に UDP ブロードキャストする。

(5) ワーカは起動時は UDP パケット待ちしており、サブマスタから UDP パケットを受信するとサブマスタに接続する。

ワーカプロセスはサブマスタから UDP パケットを受信してすぐに接続するのではなく、各ワーカプロセスでランダムな時間だけ待ち、複数のサブマスタからパケットを受信するようにする。そして負荷分散のために、その中で最も接続数の少ないサブマスタに接続する。

サブマスタは定期的に UDP ブロードキャストを行っているので、ユーザが計算ノード上にワーカプロセスを起動するだけで、自律的にそのノードを発見することができる。そのため任意のタイミングで動的にノードの追加が可能である。

3.2 クラスフレームワーク

Jojo2 上でのプログラミングは、Jojo2 の提供する抽象クラス Code を継承したクラスを実装することで行う。 Code では ParentNode, DescendantNodes, Message などのサポートクラスを用いてプログラミングを行う。

Code クラスの定義を図 2 に示す。parent, descendants はそれぞれ上位レベル、下位レベルのノードを指し、これらのオブジェクトに対してメソッドを発行することで通信を行う。 init メソッドは初期化を行う。引数となる Map には Jojo2 起動時に引数として渡す Properties 形式ファイルの内容が渡される。 init メソッド終了以前に start メソッドや handle メソッドが呼び出されることはない。 start メソッドは、実際の処理を行うメソッドである。

```
public abstract class Code {
    ParentNode parent; /* 親ノード */
    DescendantNodes descendants; /* 子ノード */

    /* 初期化 */
    public void init(Map arg);
    /* 本体の処理 */
    public void start();

    /* 送信されてきたオブジェクトの処理 */
    public void handleReceiveParent(Message msg);
    public Object handleReceiveDescendants(Message msg);

    /* ノードの参加・脱退に対する処理 */
    public void handleAddDescendant(int id);
    public void handleDeleteDescendant(int id);
}
```

図 2　Code クラス

上位レベルあるいは下位レベルからメソッドを受信すると、handleReceiveParent、hadleReceiveDescendant メソッドがそれぞれ起動される。また下位レベルのノードが参加・脱退したときは、handleAddDescendant、handleDeleteDescendant がそれぞれ起動される。これらのハンドラメソッドを実行するスレッドは、それぞれのイベントが生じたときに新たに起動される。

4. まとめと今後の課題

本稿では、動的なノード群構成機構を備え、階層型グリッド環境に適したプログラミング環境 Jojo2 の設計と実装について述べた。現在、Jojo2 はプロトタイプ実装されており、今後遺伝的プログラミングなどの組み合わせ最適化問題を通じて、システムの性能評価とスケーラビリティの検証を行う予定である。

参　考　文　献

1) Hidemoto Nakada, Satoshi Matsuoka, and Satoshi Sekiguchi: A java-based programming environment for hierarchical grid: Jojo, In *CCGrid 2004*, April 2004

2) 青木仁志、中田秀基、松岡聡: 大規模グリッド環境下での Jojo の評価, 先進的計算基盤システムシンポジウム SACSIS2005, Vol, 2005 No.5, pp. 266.