階層型グリッドミドルウェアDNASの設計と実装

折戸俊彦[†] 廣安知之[††] 三木光範[††]

[†] 同志社大学大学院 [††] 同志社大学工学部

提案システム Distributed Network Application System(DNAS) は，グリッド上の計算資源でデータ交換を行うための P2P 指向のミドルウェアである．DNAS は，ツリー構造の通信トポロジを形成し，プロセスの起動や制御，リソース情報の提供などの機能を有している．また，状況に応じて動的にトポロジ構成を変化させる機能を持つ．本稿では DNAS について設計および実装を行った．稼働実験では，PC クラスタ環境において，トポロジの再構成，アプリケーションの起動，実行制御が確認された．分散環境において 175 ノードで動的にツリートポロジが構成されることが確認された．

Design and Implementation of the Grid Middleware with Hierarchical Network Structure: DNAS

Toshihiko Orito,[†] Tomoyuki Hiroyasu[††] and Mitsunori Miki[††]

[†] Graduate School of Engineering, Doshisha University

[††] Knowledge Engineering Dept., Doshisha University

Distributed Network Application System (DNAS) is a P2P oriented grid middleware to exchange data of applications on the grid. DNAS whose logical topology is tree has functions to control process, run applications, provide resource to applications. DNAS is able to reconstructure dynamically logical topology. In this paper, we designed concept of DNAS, and had examined and discussed the functions in PC cluster and distributed environment. Functions to restructure logical topology, control applications were determined in PC cluster. DNAS booted on 175 nodes in distributed environment.

1. はじめに

近年，計算機の製造技術の進歩や開発コストの削減により，PC クラスタやスーパーコンピュータなど様々な計算資源が豊富に存在している．また，それらの計算資源をネットワークで接続し，仮想的な一つの環境として利用可能なグリッドが注目されている．グリッド技術を利用することで，各地に分散した計算資源を利用した高速演算処理や，散在するサービスのシームレスな連携を低コストで実現することが期待されている．しかし，グリッド環境を利用するためにはセキュリティや通信機構などを考慮する必要があり，それらを全てアプリケーションレベルで実装すると開発コストが高くなってしまう問題がある．このような問題を解決するために，クライアントサーバ型の Ninf-G[1] や NetSolve[2]，プールに投げられたジョブをスケジュールする Condor，GridMP，ツリー型の通信トポロジを形成する Jojo[3),4)]，Phoenix[5] など，様々なグリッドミドルウェアや分散実行環境を整えるための通信ライブラリが開発されている．

グリッド環境はその形態の一つとして PC クラスタ環境を複数繋ぎ合わせた環境が想定されており，グローバルアドレスと複数のプライベートアドレスからなる階層構造のネットワークトポロジを形成する．また，ノード数の多い環境下では管理ノードへの接続要求が多くなり，接続が集中するノードに対する通信のオーバーヘッドが大きくなる．そのため，Jojo や Phoenix のようにグリッドミドルウェアは通信機構にあらかじめ階層構造を持つか，Ninf のようにクライアントサーバモデルを多段的に適用して階層構造に適応させる手法が用いられている[6]．これらの階層構造の通信トポロジを用いる手法はスケーラビリティの向上に非常に有効である．

一方，グリッド環境を利用する際には一般的にユーザは利用できるリソースの情報を把握し，それらのリソース上で実行環境を整える必要がある．しかし，グリッド環境は資源が各地に分散し，管理組織も多岐に渡るため，これらの作業は煩雑であることが多く，ユーザの負担となっている．さらに，グリッド環境のリソースは管理者によって追加されたり，メンテナンスや故障のために削除されたりする．そのため，リソースの増減にユーザアプリケーション側で対応する必要があるが，これらの作業はユーザにとっては予測不可能なため困難である．動的なリソースの変化に対

してはシステム側で対応し，必要なときにユーザアプリケーションがシステム情報を取得可能な仕組みが望ましい．

我々は，動的に変化するグリッド環境上で稼働するアプリケーション間のデータ交換を行うためのミドルウェアである Distributed Network Application System（以下，DNAS）の設計および実装を行う．グリッド環境が潜在的に階層構造のネットワークトポロジを形成することに注目し，通信トポロジは階層構造とする．従来のアプリケーションはシステムから切り離されて実装されてきたため，システムを考慮したアプリケーションを実装することは困難であった．DNAS ではロードアベレージなどのシステム情報をデーモンから取得できるようにし，システムを考慮したアプリケーションの開発を可能にする．また，MPI ライブラリに見られる通信方式の実装は考慮に入れるが，DNAS は通信ライブラリではなく，デーモンプログラムであり，異なる複数のアプリケーション間のデータ交換も視野に入れる．

これまで我々はグリッドミドルウェアとして，本研究の前身である階層型の DNAS[7] や，マスタワーカ型の GMWS[8] の研究を行ってきた．前身である DNAS には唯一のマスターノードが存在しており，マスターノードではアプリケーションを実行できない点，モニタリングシステムであり，通信が下位ノードから上位ノードの一方向に限定されていた点が問題であった．GMWS には起動時にシステムの構成するノード数を決定する必要があり，動的なリソースの変化に対応できない点，全てのノードがフラットに接続され，負荷が単一のノードに集中してしまう点が問題であった．本提案システムは C 言語で開発されていた前身の DNAS を基に，Java 言語を用いてネットワークセキュリティや拡張性を考慮し，一から設計する．

2. DNAS の設計

本章では提案システムである階層型グリッド通信ミドルウェア DNAS の設計および実装について説明する．DNAS はグリッド環境のようなグローバルアドレス，プライベートアドレスを含んだ複数のネットワークが混在する環境で動作可能なミドルウェアである．ユーザにリソースを意識させずにグリッドを利用する環境を提供する．

DNAS で利用される各ノードでは DNAS デーモンと呼ばれるアプリケーションの通信を請け負うデーモンプログラムが起動し，DNAS デーモンはローカルホスト情報とユーザアプリケーションのデータを隣接ノードに送信する機能を有している．通信は一定時間間隔で双方向に行われ，通信路は SSL を用いて暗号化される．ユーザアプリケーションは DNAS によって提供される DNAS API を利用して DNAS デーモンと通信を行う．そのため，ユーザアプリケーション間の通信は DNAS デーモンを介して行われる．

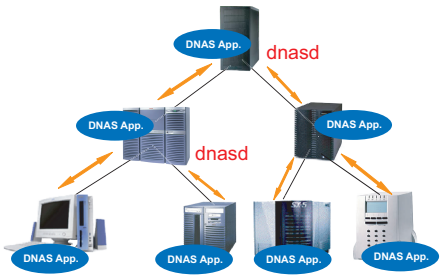

図 1 DNAS の通信トポロジ
Fig. 1 Communication Topology of DNAS

DNAS はメッセージパッシングライブラリではなく，デーモンプログラムであるため，アプリケーション起動前にデーモンを立ち上げておく必要がある．また，1 つの DNAS デーモンは独立した複数のアプリケーションと同時に通信可能であり，異なる複数のユーザアプリケーション間の連携を支援できる．

2.1 データ構造

ホスト名や IP などのリソースの管理情報やユーザアプリケーションなどの全てのデータにはタグと呼ばれるデータ識別子が付与される．また各ホスト，各アプリケーションにはそれぞれホスト識別子，アプリケーション識別子が割り当てられており，ユーザはアプリケーション識別子，ホスト識別子，データ識別子の 3 つの識別子を指定することにより，任意のデータを取り出すことができる．また，各アプリケーションは情報を任意のアプリケーションに公開する，あるいは非公開にすることを選択できる．

2.2 P2P 指向型システム

各 DNAS デーモンは下位ノードのリストを持ち，接続要求があるとそのリストに追加する．その後はリストを参照しながら一定時間間隔で双方向にノードが保有する情報を交換し，タイムアウトにより接続できなかったノードがあった場合はリストから削除する．

上記の仕組みにより，システム全体を停止させることなく，任意のタイミングでシステムに利用ノードを追加，削減を行うことができる．ただし，ユーザアプリケーションのフォールトトレランス機構は現時点では想定していない．

2.3 ツリートポロジ

計算ノードを投入する際にはすでに稼働中の DNAS デーモンのホスト名を上位ノードに与えて DNAS デーモンを起動する．新しいノードで接続要求が受理され，ACK が返ってくるそのノードを上位ノードとして登録する．これらの操作を計算ノードを追加することを繰り返すことによって，DNAS デーモン間の通信のための論理的な通信路が設定され，**図 1** のように，システム全体としてツリートポロジを構成する．

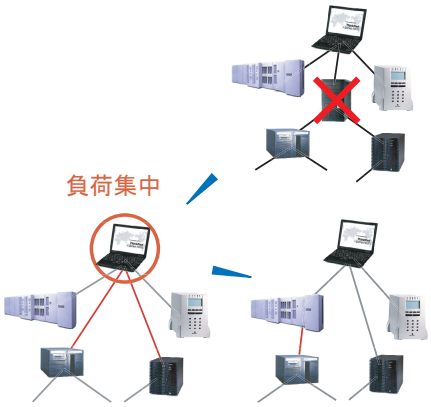

図 2 通信トポロジの動的な再構成
Fig. 2 Dynamic restructuring of communication topology

各ノードは隣接する一部のノードとのみ一定時間間隔で直接通信を行い，それ以外のノードとの通信は他のノードを経由して間接的に通信を行う．ツリートポロジを用いた通信により，一部のノードに負荷を集中させることなくユーザが指定したパス長の範囲で情報を共有することができる．

2.4 通 信 機 構

DNAS は各資源を動的な階層型ネットワークで接続し，各ノードがネットワークに流れるデータを中継する仕組みを有する．そのため，DNAS デーモンは構成されたツリートポロジに則ってデーモン上のデータを中継する．ただし，ユーザアプリケーションの通信はこの限りではなく，DNAS デーモンから取得できるシステム情報を基にユーザが自由に送信先を選択できる．ただし，通信は送信先で稼働している DNAS デーモンと行われ，データ交換は DNAS デーモンを介することで実現する．メッセージパッシング方式に見られるブロードキャストなどのような通信機構は現在実装していないが，実装することを視野に入れ，ツリートポロジに依存しない通信を可能としている．

2.5 通信トポロジの動的な再構成機構

グリッド環境にはノードの追加および削減，ネットワーク障害がいつどこで発生するか予測できないという特徴がある．さらに，ツリートポロジを用いた通信機構にはその特性上，中間層ノードの障害に非常に脆弱であるという問題がある．また，任意のタイミングでのノードの追加や，動的再構成機能による通信トポロジの変化により，特定のノードを上位ノードとする下位ノードの数が想定外に多くなってしまう可能性がある．

DNAS ではこれらの問題に対応するために通信トポロジの動的再構成アルゴリズムを 2 種類有している．DNAS におけるトポロジの動的な再構成の様子を**図 2** に示す．

- 上位ノードに障害が発生し通信できなくなってしまった場合に下位層のノードが停止したノードを迂回するように再接続する．
- 下位ノード数が設定された上限値を超えた場合に下位層のノード数を調整するように再構成する．

2.5.1 上位ノードの再構成アルゴリズム

DNAS デーモンは上位ノードが接続できなくなった場合に再接続するために代替ノードの情報を保持している．ルートノード（第一層）は上位ノードが存在しないために代替ノード情報は保持していない．第一層，第二層を除いて，代替ノードは上位ノードの上位ノードが選択される．第二層についても上位ノードの上位ノードが存在しないために代替ノードの情報は保持していない．

各ノードは下位ノードのリストを参照し，一定時間間隔で上位ノードの情報を代替ノードの情報として下位ノードに送信する．すなわち，代替ノードの情報の取得については下位ノードは上位ノードの下位ノードのリストに載っていれば良い．代替ノードの情報を取得したノードは，代替ノードに対して接続を試みて，応答を待つ．応答があれば，接続可能と判断し，代替ノードとして登録する．基本的に代替ノードは上書きされる．グリッド環境では複数のネットワークが混在するために，代替ノードとして送られてきた情報が正しくても接続できない場合がある．代替ノードに接続できない場合，代替ノードが存在しない場合（第一層，第二層）は代替ノードを登録しない．

各ノードは一定時間間隔で上位ノード，代替ノードに対して接続を試みる．上位ノードの応答がない場合は再接続，代替ノードの応答がない場合は代替ノードの登録削除が行われる．再接続の操作は代替ノードに対して登録申請することで完了する．代替ノードの情報についてはその時点で削除し，次の代替ノードについては上位ノードから送られてきたものを登録する．

第一層の障害，上位ノードとその上位ノードの同時障害への対応は現時点では実装していないが，すべてのノードは全く同じプログラムで構成されているため，代替ノードを決定するルールを設定するだけで容易に再構成を行うことが可能である．現時点では再構成ができなかった場合はシステムは停止せず，ツリートポロジが分断される仕様となっている．

2.5.2 下位ノードの再構成アルゴリズム

下位ノードの数が設定値を越えた場合に無作為に選択した下位ノードを下の階層に移動させる．移動させる下位ノード，移動先のノードの選択する他のアルゴリズムについては現時点では実装していない．移動させる下位ノードは選択された時点で上位ノードの下位ノードリストから削除される．

上位ノードは選択した移動させる下位ノードに対して，移動先のノードの情報を送信する．情報を受信した下位ノードは上位ノードの情報を代替ノードに登録し，受信したノードに対して接続を試みる．応答が

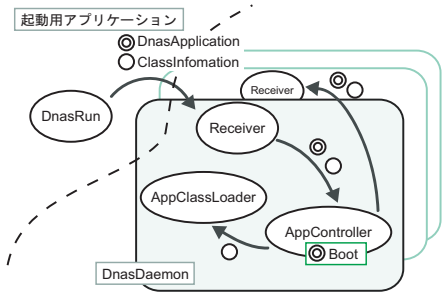

図 3 アプリケーションの起動の仕組み
Fig. 3 Function of application boot

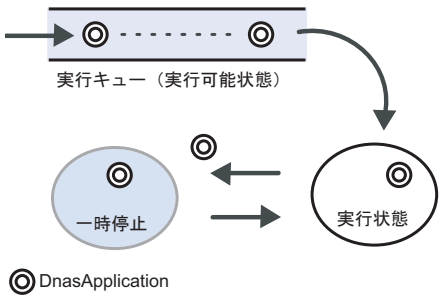

図 4 アプリケーションの同時起動数の制御
Fig. 4 Number control of boot applications

ある場合は受信したノードを上位ノードとして登録する．応答がない場合は受信したノードと接続できないと判断し，上位ノードの再構成アルゴリズムに従い，代替ノードに接続をする．再構成前の上位ノードを代替ノードとして登録するため，下位ノードの最構成中に障害が発生する場合についても対処できる．

下位ノードの再構成については上位ノードが完全に管理するため，再構成の過程でツリートポロジが分断されることはあっても，ループ構造になることはない．

2.6 アプリケーション制御機構

グリッド上で提供される資源を利用する際には資源にかかる負荷を軽減することが重要である．資源を酷使することはシステムの故障やパフォーマンスの低下に繋がるため，資源に対する管理コスト増大の原因となる．しかし，グリッド上には複数のユーザが同時に存在し，グリッドは複数のネットワークを繋いだ環境であるために，資源の情報をユーザが把握することは非常に困難である．

DNAS はアプリケーションの起動をユーザ側から請け負う仕組みを有している．アプリケーションの起動を DNAS デーモンが行うため，DNAS デーモンが DNAS デーモン上で動作する全てのアプリケーションを把握している．そのため，DNAS デーモンはアプリケーションを制御するための次の 2 つの仕組みを有することが可能となる．すなわち，アプリケーションの同時起動数を制御する仕組みと負荷に応じてアプリケーションを中断，再開する仕組みである．

2.6.1 アプリケーションの起動

アプリケーションの起動は DNAS デーモンに委譲することにより行われる．ユーザがアプリケーションを起動する場合は，ホスト名および利用状況などの各リソース情報や各ホストで DNAS デーモンが動作していることを把握しておく必要があり，ユーザに負担がかかる．アプリケーションの起動の仕組みを**図 3** に示す．

ユーザはアプリケーション起動用のプログラム（DnasRun）を用いて任意のホストで動作する DNAS デーモンにアプリケーションとクラス情報を送信し，アプリケーションの起動を委譲する．DNAS デーモンでは Receiver で受信したアプリケーション，クラス情報をそれぞれ AppController，AppClassLoader に受け渡し，アプリケーションは AppController によって起動される．AppClassLoader は必要に応じてクラス情報をロードする．AppController はさらに隣接する DNAS デーモンにアプリケーション，クラス情報を通知する．通知を受けた DNAS デーモンも同様の処理を行い，システム内で次々とアプリケーションが伝播する．これらの仕組みにより，ユーザは単一のノードにアクセスするだけで複数のリソースを利用することができる．そのためユーザはアプリケーションの起動の際に各ノードの起動の有無を知る必要がなく，各ノードの存在すら意識する必要はない．

2.6.2 アプリケーション同時起動数の制限

DNAS はアプリケーションを同時に起動する数を制限することにより資源に必要以上にジョブが投入されることを防ぐ仕組みを有している．アプリケーションの起動制御の様子を**図 4** に示す．

実行を委譲されて実行可能状態になったアプリケーションは図 4 のように実行キューに保存される．DNAS 上で動作しているアプリケーションの数があらかじめ設定された数に満たない場合に実行状態に推移し，設定値に達している場合は保留される．実行中のアプリケーションが終了すると，実行可能状態のアプリケーションのうち先に到着したアプリケーションから順に DNAS デーモンによって起動され，実行状態に推移する．

2.6.3 アプリケーションの中断と再開

資源にかかる負荷はアプリケーションによって異なり，同一ジョブでない限りアプリケーションの数で判断することはできない．すなわち，資源に高負荷がかかっている状態を回避することはアプリケーションの同時起動数を制限するだけでは実現することはできない．DNAS は実行中のアプリケーションの動作を負荷に応じて中断，再開する仕組みを有している．アプリケーションの中断，再開のアルゴリズムについては現時点では考慮していない．アプリケーションの状態を保存したまま中断，再開できることが重要であり，スケジュールなどについては他のシステムを利用するこ

とも考慮する．現時点では，DNAS デーモンは資源にかかる負荷があらかじめ設定された閾値を超えた場合に，実行状態のアプリケーションを負荷が閾値を下回るまで中断し，中断されたアプリケーションはそれぞれ一定時間経過後に自動的に再開される．

本提案システムでは様々な異なるアプリケーション間のデータ交換を想定している．アプリケーションを中断することは並列アプリケーションなどにとっては相互に通信できなくなることを考える必要があるが，DNAS ではアプリケーション間で直接通信することは許可しておらず，アプリケーションは原則 DNAS デーモンと通信を行う．そのため，アプリケーションが中断しているノードに対しても，DNAS デーモンを介してデータの送受信することが可能である．

2.7 ノード数増加に伴う負荷の軽減

グリッド環境はネットワークで繋がれた複数のマシンで構成されているため，それらを連携させるための通信制御にはマルチクライアント処理を行う必要があり，送受信の各処理にはスレッド処理が利用されることが多い．しかし，スレッド処理による多重化は同時接続数が多くなるとスレッドの大量生成によるパフォーマンスの低下や，メモリ不足によるエラー終了が発生するなどの問題が報告されている[9),10)]．

このようなスレッドの大量生成に伴う問題を解決するための一つの回避策として通信トポロジを階層型にして，1 つのノードに接続されるノード数を抑える手法があり，提案システムでも採用している．階層型トポロジによる通信制御は有用な手法であり，スケーラビリティの向上に対して一定の効果を上げているが，スレッドの大量生成に対する解決策とは言えない．

スレッド生成数はコネクション数に依存するので，ある特定のノードに発生するコネクション数に注目する．あるノードに接続するノード数を N，システム制御，アプリケーション処理において発生する単位時間あたりのコネクション数をそれぞれ Nc，Na とすると，単位時間に発生するコネクション数 TNc は $TNc = N(Nc + Na)$ で表される．

ここで，TNc の削減について考える．Nc はシステムの維持に必要な制御に関するコネクション数で変化しないため，同時接続数 N を制限することにより制御可能である．一方，Na はユーザアプリケーションに依存しているため，DNAS デーモン起動時に知ることができない．そのため，ユーザがアプリケーションを登録するまで適切な同時接続ノード数はわからず，起動時に同時接続ノード数を設定することによって特定のノードへのコネクション数を制限することはできない．また，DNAS は任意のタイミングでノードを追加することができることや動的に通信トポロジを再構築し接続形態が変化することから接続ノード数が時間によって変化し，それに併せて必要なスレッドの数も増減する．DNAS ではスレッドの大量生成に関する問題を考慮して送受信に関する通信機構が設計されており，スレッド数を制限する仕組みや非同期 I/O を用いた多重化を行っている．

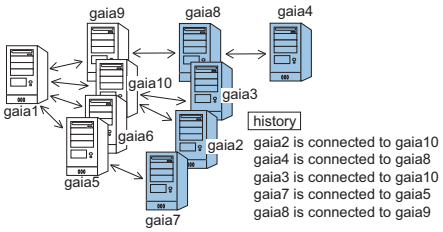

図 5 1 ノードに対する接続ノード数の制限
Fig. 5 Connected node number control

3. 稼 働 実 験

DNAS は階層型の通信トポロジを形成し，DNAS デーモン間の通信はそのトポロジを用いて行われる．また P2P 型のシステムであるため，ノードの追加および削減を任意のタイミングで行うことができる．大きな特徴としてツリーを動的に再構成する機能があり，動的な環境の変化に対応し，1 ノードあたりの最大接続ノード数を制限する機能を有している．

本稿では，ツリーの動的再構成機能とアプリケーションの実行制御，スレッド生成数，ノード数に関するスケーラビリティに関する実験を行う．実行環境を**表 1** に示す．

3.1 ツリーの動的再構成機能

1 ノードあたりの最大接続ノード数を 4 に設定し，gaia1 に gaia2 から gaia10 の 9 ノードを接続した．gaia2，gaia3，gaia4，gaia7，gaia8 がそれぞれ接続先を変更し論理ネットワークは**図 5** のようになった．

次に gaia10 の DNAS デーモンを停止し，接続できないようにした．gaia2，gaia3 がそれぞれ接続先を変更し，再構成が行われ，**図 6** のような構成となった．gaia2 は gaia1 に接続した後，接続ノード数の制限により gaia9 に再接続することが確認された．

図 5，図 6 より，DNAS システムが 10 ノードの PC クラスタ環境で稼働したこと，ツリーの動的再構成機能が再構成アルゴリズムに従い，正常に動作していることが確認された．

3.2 アプリケーションの実行制御

アプリケーションの実行制御を行うロードアベレージ値を 0.5，1.0 に設定し，アプリケーションを実行

表 1 実行環境（PC クラスタ）
Table 1 Experimental PC cluster environment

name	gaia1 - gaia10
CPU	PentiumIV 2.8GHz * 2
memory	512MByte * 2
network	100BASE-T

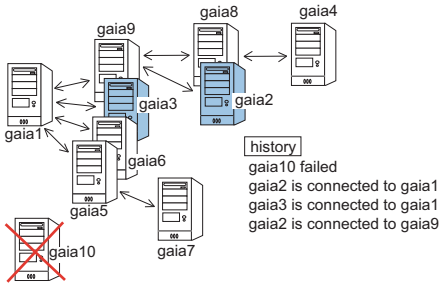

図 6 障害に対する動的再構成機能
Fig. 6 Dynamic restructuring function against system failure

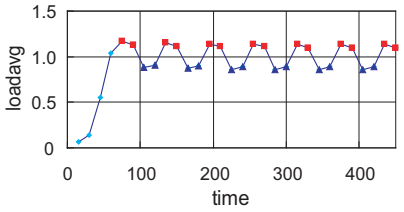

図 7 アプリケーションの実行制御
Fig. 7 Application load average control

した．**図 7** は閾値が 1.0 の場合のグラフであり，横軸が時間，縦軸がロードアベレージを示している．実験では 2 つのアプリケーションを実行した．処理内容はそれぞれ無限ループプログラムであり，制御しない場合はロードアベレージ値は 2.0 になる．図 7 はアプリケーションのロードアベレージが閾値である 1.0 を超えた場合に中断され，下回ると再開されることを示している．閾値が 0.5 の場合も同様の結果となり，ロードアベレージが閾値を超えるとアプリケーションが中断され，下回ると再開された．

ジョブをスケジュールするアルゴリズムやスケジューラについては考慮していない．今回の実験では DNAS がアプリケーションの実行を中断，再開することが可能なことを示した．

3.3 スレッド生成数に対する検討

スレッド生成数を制限する機能によって，特定のノードに接続されたノード数とスレッド生成数の関係を調べた．gaia1 に gaia2〜gaia10 の 9 ノードを接続した構成とし，送信処理に関するスレッド生成数を 10 に制限した場合とスレッド生成数を制限しない場合について gaia1 のスレッド生成数を調べた．スレッド生成数とは DNAS デーモンがノード間で送受信を行う間隔内に生成されるスレッド数を指す．ここではその間隔を 15 秒とした．**図 8** は gaia1 におけるスレッド数の推移のグラフであり，横軸に接続ノード数，縦軸にスレッド生成数を示している．

DNAS システムに必要なスレッド数は 6 であり，受

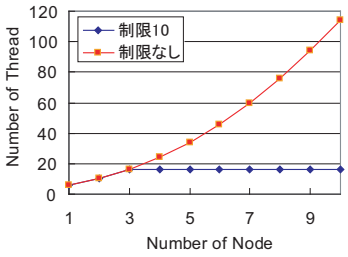

図 8 接続ノード数によるスレッド生成数の推移
Fig. 8 Transition number of generated threads with along to the connected number of nodes

信処理もその中に含まれている．それ以外のスレッドは全て送信処理に用いられている．送信処理の内訳は接続ノード数を N とする．送信スレッド生成数を制限しない場合に注目すると，ノードの制御情報に要するスレッド生成数は $3 * N$ であり，各ノードの情報を送信するスレッド数が $N * N$ となっている．スレッド生成数を制限しない場合は $o(N^2)$ であり，スレッド生成数は接続ノード数によって指数的に増加する．一方，送信スレッド生成数を制限した場合は送信スレッド数が設定された値を超えないことが確認された．

スレッド生成数を制限することでスケーラビリティが向上したが，設定値に対する接続ノード数の割合が高くなった場合に送信処理が追い付かない可能性が考えられる．そこでスレッド生成数を 1 とした場合について実験を行った結果，送信処理に対するスレッド生成数は 1 で，なおかつ送受信の間隔内に全てのタスクを処理していた．実験環境，送受信データサイズが小さいことが要因であると考えられるが，スレッド処理能力が不足する現象は確認できなかった．

3.4 ノード数に関するスケーラビリティの検討

スレッド生成数に関して管理者側でコントロールできることが確認された．前述の 3 つの実験では実験環境として PC クラスタ環境を用いていたが，本節では広域分散環境においてノード数を増やした場合について動作を確認する．用いた実験環境を**表 2** に示す．

表 2 の ibp は単一ノードであり，galley，supernova，xenia は PC クラスタである．単一ノードに 3 つの PC クラスタを接続した状態を初期状態とした．PC クラスタ内は master ノードに全てのノードをフラットに接続した状態から始めた．1 つのノードに接続できる下位ノード数を 4 とし，通信トポロジの変化を調べた結果，175 ノードの全てにおいて DNAS デーモンが動作し，最大下位ノード数を 4 としたツリートポロジを構成していることが確認された．

表 2　実行環境（分散環境）
Table 2　Experimental distributed environment

name	galley	supernova	xenia	ibp
number of node	11	160	3	1
CPU	PentiumIV 2.8GHz * 2	Opteron 1.8GHz *2	xeon 2.40GHz *2	PentiumIV 1.13GHz * 2
memory	512MByte * 2	2GByte	1GByte	1GByte
network	100BASE-T	1000BASE-T	myrinet	-

4. 今後の課題

4.1 システムの性能評価

本稿ではアプリケーションとして無限ループプログラムを利用し，システムの稼働を確認した．実際に利用されることを想定したアプリケーションを用いた際の挙動に関する検討，アプリケーション制御に関する検討，通信トポロジの動的再構成，故障からの修復，アプリケーション間，DNAS システムとアプリケーション間などのデータ通信のオーバヘッドに関する検討を行う．

4.2 ユーザ認証

DNAS では複数のアプリケーションを同時に実行することができ，複数のユーザのアプリケーション同士を連携させて一つの仕事をさせることも可能である．しかし，DNAS デーモンを立ち上げたアカウント以外のアプリケーションを請け負うことはセキュリティ上大きなリスクを伴うため，今後はユーザ認証にも焦点をあてて開発を行っていく必要がある．

4.3 フォールトトレランス

DNAS はリソースの状況に応じて通信トポロジを動的に再構成する機能を有しているため，障害によりノードが停止した場合においてもシステム全体を停止することなく処理を続けることができる．しかし，停止したノードで実行中のデータについては復元することができないため，今後新たな仕組みを考える必要がある．

5. 関 連 研 究

グリッド環境が階層構造のネットワークを構成することを考慮した関連研究としてマスタワーカ型の GMWS[8]，クライアントサーバモデルの Ninf[1] を利用したものや階層型の通信機構を有する Jojo[3),4)]，Phoenix[5] がある．

GMWS はマスタワーカ型のグリッドミドルウェアであり，システムの起動およびノード間の通信に Globus を利用し，強固なセキュリティ機構を有している．一方で，システムの起動時に構成するリソース数を決定する必要があるため，リソースの追加および削減ができない．また，全てのノードがフラットに接続されるため，負荷が単一のノードに集中してしまう問題がある．

Ninf を利用したモデルではクライアントサーバモデルを多段的に適用することにより，階層構造のネットワークを適応している．また，グローバルなネットワークには Ninf-G，プライベートなネットワークには Ninf を用いることにより，セキュリティと性能の両立を図っている．Jojo を利用したモデルでは全てのノードを利用するような階層構造を定義した上で通信を開始する．開始時にシステムとアプリケーションを各リソースに転送するため，階層構造を定義する作業はユーザに委ねられている．Phoenix は階層型のネットワークにも適応する並列計算ライブラリであり，Wide-Area Network において MPI ライクな独自の API を用いた集合通信を可能とする．また，動的な計算環境の変化にも耐えうる設計となっている．

まず，階層構造の通信トポロジを有し，多段的に適用するなどの工夫をせずにグリッドを環境へ適応することができる点で Ninf と異なる．また，Ninf，Jojo は計算環境は計算開始時から変化しないことを前提としている点が提案手法と異なる．Phoenix は動的に計算環境が変化することを想定している点では同じであるが，Jojo と MPI ライクな並列計算ライブラリであり，デーモンプログラムである DNAS とは根底にあるコンセプトが違う．また，DNAS デーモンによってアプリケーションの起動，中断，再開が行われるため，ユーザはリソースを意識する必要がなく，ジョブはリソースの空き時間を利用して処理を実行することが期待される．

6. お わ り に

グリッド環境は潜在的に階層構造のネットワークを形成し，ユーザが利用できるリソースの状況は動的に変化するという特徴を持っている．そのため，システムのスケーラビリティやユーザの負担の軽減を考えると，階層構造の通信トポロジを構成すること，システムがリソースの動的変化に対応することが有効である．

本稿では階層型グリッド通信ミドルウェアである DNAS の設計および実装を行った．PC クラスタ環境，グリッド環境ともに動的再構成機能により下位ノード数を設定した値以下になるように通信トポロジを変化させることができることが確認された．また，故意に中間層のノードに障害を発生させた場合の動的再構成も確認された．スケーラビリティに関しては，スレッド

管理機構によって生成されるスレッド数をコントロールできること，アプリケーションの起動，中断，再開ができること，分散した計算環境において少なくとも175 ノードで動作することが確認された．

参 考 文 献

1) 田中 良夫，中田 秀基，平野 基孝，佐藤 三久，関口 智嗣: Globus による Grid RPC システムの実装と評価，情報処理学会ハイパフォーマンスコンピューティングシステム研究会，no.77 (2001)
2) Casanova. H. and Dongarra. J.: NetSolve: A Network Server for Solving Computational Science Problems. *Proceedings of Super Computiong '96 (1996).*
3) 中田 秀基, 松岡 聡, 関口 智嗣: グリッド環境に適した Java 用階層型実行環境 Jojo の設計と実装, 情報処理学会研究報告 2002-HPC-92, pp. 31-36 (2002).
4) 中田 秀基, 松岡 聡, 関口 智嗣: Java による階層型グリッド環境 Jojo の設計と実装，情報処理学会論文誌コンピューティングシステム Vol.44 No. SIG 11 , pp. 46-56 (2003).
5) Hideo Saito, Kenjiro Taura, Takashi Chikayama: Collective Operations for Wide-area Message Passing Systems Using Adaptive Spanning Trees, 情報処理学会論文誌コンピューティングシステム Vol.46 No. SIG 12 , pp. 373-383 (2005).
6) 小野 功，水口 尚亮，中島 直敏，小野 典彦，中田 秀基，松岡 聡，関口 智嗣，楯 真一: Ninf-1/Ninf-G を用いた NMR 蛋白質立体構造決定のための遺伝アルゴリズムのグリッド化，先進的計算基盤システムシンポジウム SACSIS2005，Vol, No.5 (2005).
7) Junichi Uekawa, Tomoyuki Hiroyasu, Mitsunori Miki and Yusuke Tanimura: A Dynamic Hierarchical System for Large Scale Distributed Applications, *Proceedings of the 14th IASTED International Conference, Parallel and Distributed Computing and Systems, pp. 422-427 (2002).*
8) Yusuke Tanimura, Tomyuki Hiroyasu, Mitsunori Miki: Development of Master-Worker System for The Computational Grid, *IPSJ Transactions on Advanced Computing Systems, Vol.45, No.06 pp. 197-207 (2004).*
9) 折戸 俊彦，廣安 知之，三木 光範: グリッド環境における DNAS2 の実行テストおよびその検討, 先進的計算基盤システムシンポジウム SACSIS2005，Vol, No.5, pp. 260-261.
10) 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, No.5, pp. 266.