Transactions of JSCES, Paper No.20080027

# DNAS：システム情報を取得する仕組みを有したグリッド用ミドルウェア*

# DNAS : A Grid Middleware with support for System Information Acquisition

中尾 昌広[1]，折戸 俊彦[1]，山崎 弘貴[1], 廣安 知之[2]，三木 光範[3],

Masahiro Nakao, Toshihiko ORITO, Hirotaka YAMAZAKI

Tomoyuki HIROYASU and Mitsunori MIKI

[1] 同志社大学大学院工学研究科 (〒610-0321 京田辺市多々羅谷 1-3)

[2] 同志社大学生命医科学部 (〒610-0321 京田辺市多々羅谷 1-3)

[3] 同志社大学理工学部 (〒610-0321 京田辺市多々羅谷 1-3)

Distributed Network Application System (DNAS) is a P2P-oriented middleware which supports to obtain system information and to exchange data among several applications on Grid. DNAS constructs a dynamic communication topology for fault tolerance using tree topology and provides API for acquisition of system information and application data. Users can develop an application that adapts to a dynamic change of the calculation resource by using DNAS. In this paper, we proposed the design and implementation of DNAS and developed a max number search application in the DNAS environment. Moreover, we evaluated the performance of DNAS functions. As a result, we confirmed that the time required for the dynamic re-composition of the communication topology was short.

***Key Words***: Grid Computing, Communication Topology, Resource Management, Software, Peer-to-Peer

## 1. はじめに

複数の管理組織に属するPCクラスタなどの計算資源をネットワークで接続し，それらを仮想的に1つの計算資源として利用するグリッドコンピューティング（以下グリッド）が普及しつつある．

グリッド環境上でアプリケーションを実行させる場合，ユーザは利用できる計算資源のシステム情報（ノードの性能や負荷率，生死など）をアプリケーション実行前に把握しておく必要がある．しかし，グリッドの計算資源は各地に分散しており，かつ計算資源の管理組織はそれぞれ異なるため，これらの作業は煩雑であることが多く，ユーザの負担になっている．さらに，グリッド環境の計算資源は頻繁に追加，もしくはメンテナンスや故障による削除が行われる．グリッド環境を有効に利用するためには，アプリケーションがシステム情報を必要なときに取得し，その情報を基にアプリケーションの動作を柔軟に変化させることが可能な仕組みが望まれる．例えば，グリッド環境でよく用いられる最適化手法の遺伝的アルゴリズムを複数台のノードで実行したい場合，各ノードのシステムの変化にともない動的に探索点の数などのパラメータを変化させることで，効率的に計算を行うことが可能になる．

また，今後のグリッドに求められていることに，異なる管理組織が持つ異なるアプリケーション同士を連携させることにより，新しいサービスを開発，提供することが挙げられる．そのため，ユーザが計算資源の管理組織を意識せずにアプリケーション間でデータ交換を行える仕組みがあると望ましい考える．

本稿では，アプリケーションからグリッド環境の計算資源のシステム情報を取得でき，かつユーザが管理組織の違いを意識することなくアプリケーション間のデータ交換を行うことが可能なミドルウェアDistributed Network Application System（以下DNAS）の提案を行う．DNASはグリッド上のシステム情報を取得し，その情報をユーザが作成するアプリケーションが利用できる形で提供する．ユーザはDNASを用いることで，計算資源の動的変化を考慮したアプリケーションの開発

* 原稿受付 2008 年 06 月 06 日, 改訂年月日 2008 年 08 月 22 日, 発行年月日 2008 年 10 月 10 日, 

が可能になる．また，ユーザは異なるアプリケーション間の連携を容易に行うことが可能になる．

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，システムを構成するノードの生死や負荷率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．例えば，リソースの変化やノードの生死といった情報に基づいて，動作しているアプリケーションの挙動を変化させるといったことは通常行わない．しかし，ユーザはDNASを用いることでシステム情報を考慮したアプリケーションの開発が可能になる．

## 2. 要件

グリッド環境の計算資源のシステム情報を取得し，かつ異なるアプリケーション間でデータ交換を行うための要件としては以下が挙げられる．

**要件1** 負荷分散と可用性の向上
1台の管理サーバに全計算資源のシステム情報を保存する場合，大規模環境下では管理サーバに負荷が集中するため，性能低下を引き起こすことが考えられる．また，可用性の問題点もある．そのため，あるノードのシステム情報を複数台のノードに分散して複製保存することで，負荷分散と可用性の向上を図る

**要件2** 2層構造の通信トポロジへの対応
一般にグリッド環境はLAN等を用いた内部ネットワークとインターネット等を用いた外部ネットワークという2層構造の通信トポロジを形成する（Fig. 1）．そのため，上記で述べたノード間におけるシステム情報の複製保存を行うための通信路は，インターネットとLANにまたがった構成を作成可能にする

**要件3** 任意のタイミングでの計算資源の追加/削除
グリッド環境の計算資源は頻繁に追加，削除が行われる．そのため，DNASが動作する計算資源の追加，削除を任意のタイミングで行えるように対応する

**要件4** アプリケーションを動作させるノードの自動選択
グリッド環境の計算資源は動的に変化するため，ジョブを実行する度にユーザが使用する計算ノードを指定する方法はユーザの負荷が大きい．そのため，ユーザはどのノードを利用するかを意識せずに，負荷の低いノードを自動的にDNASが選択し，アプリケーションを実行できる仕組みを開発する

要件1は3.1節で，要件2と3は3.3節で，要件4は3.6節で，それぞれの要件を満たすDNASの提案を行う．

## 3. DNASの設計と実装

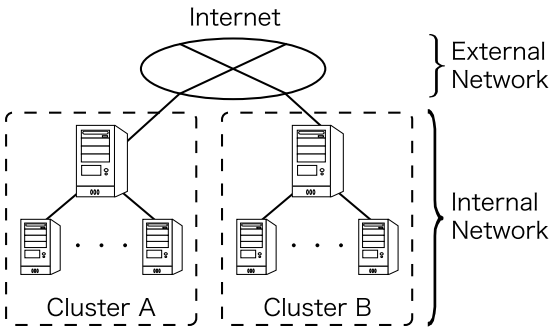


Fig.1 Communication Topology of Grid

**3.1 システム構成** DNASはデータを格納するデータプール，データプールに保存されたデータを操作するプログラミングAPI，システム情報の取得とその情報を共有するための通信を制御するDNAS daemon（DNASデーモン）の3つから構成される．

DNASが動作している各ノードでは，システム情報とアプリケーションが出力するデータを格納するためのデータプールを持っている．そして，DNASはデータプールにアクセスするためのプログラミングAPIを提供している．ユーザが作成するアプリケーションはDNASが提供するAPIを呼び出すことで，任意のタイミングでシステム情報の取得とアプリケーションデータの取得と保存を行うことができる．

グリッド上の各ノードでは，DNASデーモンと呼ばれるデーモンプログラムが動作している．DNASデーモンは定期的に自ノードのシステム情報をデータプールに保存している．また，DNASデーモンは計算資源の動的変化に対応した論理的な通信トポロジを各ノード間で構築し，その通信路を用いて各ノードがデータプールの情報を送受信している．データプールの情報を何ノード先まで送信するかは，DNASの起動時に指定する．また，通信路のトポロジはツリー構造を採用しているため，1台のノードに負荷を集中させない特徴を持っている．これらの仕組みにより，2章で挙げた負荷分散と可用性の高いシステムを実現している．なお，通信は一定時間間隔で行われ，通信路はSSLを用いて暗号化される．

**3.2 データの識別方法** システム情報とアプリケーションが出力する全てのデータにはタグと呼ばれる識別子が付与される．タグには，ホストに対するホスト識別子，アプリケーションに対するアプリケーション識別子，ユーザが任意で付けるデータ識別子の3種類があり，ユーザはDNASが提供するAPIを用いてそれらの識別子を指定することで，任意のデータの送受信を行うことができる．

DNASが提供している主要なAPIを下記に示す．

- Object recv(String APPNAME, String TAG)
  システム情報とアプリケーションデータの受信処理を行うための関数である．任意のアプリケーシ

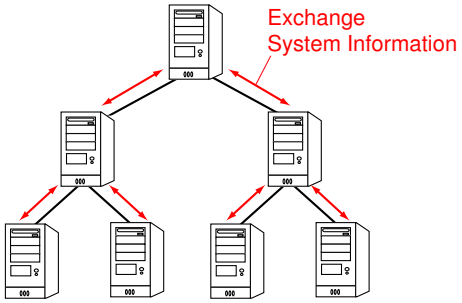


Fig.2　Communication Topology of DNAS

ョン識別子（APPNAME）とデータ識別子（TAG）に一致するデータをデータプールから取得する．また，アプリケーション識別子に”sys-info”，データ識別子にノード名を指定すると，そのノードのシステム情報を取得することができる．データ識別子に”hostname”を指定した場合は，実行できるホストのリストが返される．

- boolean send(String TAG, Serializable DATA)
  共有したいアプリケーションのデータ（DATA）にデータ識別子（TAG）を付与して送信する関数である．
- boolean first_send(String HOSTNAME, String TAG, Serializable DATA)
  第一引数（HOSTNAME）で指定したノードに対して，共有したいアプリケーションのデータ（DATA）にデータ識別子（TAG）を付与して直接送信する関数である．

なお，ホスト識別子はそのアプリケーションが実行されるホスト名，アプリケーション識別子はプログラムファイル名の拡張子を除いた部分が自動的に設定される．

**3.3　通信トポロジ**　DNAS では自ノードで取得したシステム情報を各ノード間で複製保存するための通信路として，Fig. 2 のようなツリートポロジを採用している．

計算ノードを既存の環境に追加したい場合，ユーザは追加したいノード上において，既に DNAS が起動しているホスト名を指定してから DNAS を起動する．指定したホスト名と接続許可が得られた場合，追加したいノードは指定したノードを自身の上位ノードとしてリストに追加し，接続する．また，指定されたノードは追加したノードを下位ノードとしてリストに追加する．以上の操作を計算資源の追加の度に行うことによって，すべてのノード間でツリートポロジの通信路が構成される．この仕組みにより，2 章で挙げた任意のタイミングでの計算資源の追加/削除を実現している．

なお，異なる管理組織上に存在する各ゲートウェイノード上で DNAS を起動させ，それらのゲートウェイノード間で通信路を構成することにより，異なる管理組織内に存在するノード間においてシステム情報の送信が可能になる．この特徴により，Fig. 1 に示した 2 層構造の通信トポロジに対応することが可能になる．また，ユーザは管理組織を意識することなく，計算資源のシステム情報を取得することが可能になる．

DNAS デーモンでは上位ノードのリストと下位ノードのリストを参照しながら一定時間間隔でデータプール内の情報を交換する．全てのノードは自ノードと論理的に接続された以外のノードとは他のノードを経由した間接的な通信を行うため，一部のノードに負荷を集中させることなく情報を共有することができる．なお，タイムアウトにより接続できなかったノードがある場合，該当ノードをリストから自動的に削除する．そして，後述する通信トポロジの動的な再構成機構により，接続不可になったノードよりも下位にあるノードは，別ノードに接続する．

データプールの情報を交換する頻度は，DNAS 起動時にユーザが決定する．現在想定している DNAS の通信の頻度は 5 秒 から 20 秒であり，その場合，3Hop 離れた上位ノードに対してシステム情報を伝達するには，15 秒から 60 秒を要する．

システム情報のデータ通信は Fig. 2 のようなツリートポロジで通信を行うが，アプリケーションのデータ通信については高速に通信を行うことが求められるため，Fig. 2 に示したツリートポロジとは関係のない，通常のネットワークを用いた送信を行うことも可能にする．

**3.4　通信トポロジの動的な再構成機構**　ツリートポロジを用いた通信機構は中間層のノードの障害に脆弱であるという問題がある．また計算資源の追加により，あるノードに接続される下位ノードの数が想定以上に多くなる可能性がある．以上の問題に対応するため，DNAS では下記のような通信トポロジの動的再構成機能を実現している．

- 上位ノードと接続が行えない場合
  上位ノードに障害が起きた際に別ノードに接続するため，DNAS デーモンは常に代替ノードの情報を保持している．代替ノードは上位ノードの上位に位置するノードが選択される．DNAS デーモンは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．代替ノードの情報を取得した下位ノードは，代替ノードに対して接続テストを試みる．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．上位ノードが故障し代替ノードに接続した場合，DNAS デーモンは上位ノードと代替ノードの情報を更新する．
- 自ノードの下位ノード数が設定した上限値に達している場合
  接続を試みた下位ノードを，さらに 1 つ下の階層に移動させる．DNAS デーモンは移動させる下位

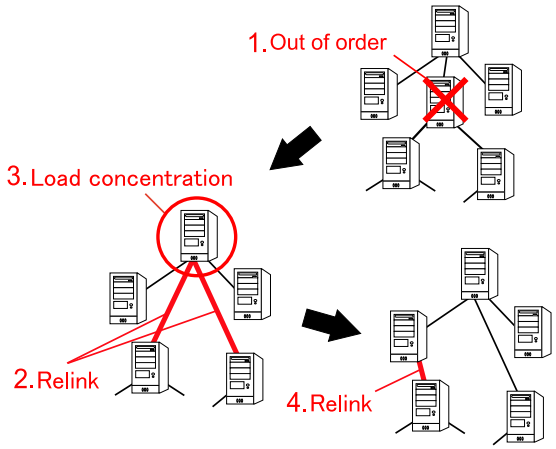


Fig. 3　Dynamic restructuring of communication topology

ノードに対して，下位ノードのリストから無作為に選択した 1 台を移動先のノードの情報として送信する．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として登録し，受信した情報のノードに対して接続を試みる．接続が許可された場合は受信した情報のノードを上位ノードとして登録する．受信した情報のノードとの接続が許可されない場合は代替ノードに接続する．

通信トポロジの動的再構成のシナリオを下記に，そのシナリオを図示したものを Fig. 3 に示す．

1. あるノードが故障し，故障したノードと接続していたノードとの接続が途切れる
2. 接続が途切れたノードは代替ノードに接続を試みる
3. 代替ノードは接続を試みたノードの 1 台は接続を許可するが，2 台目は下位ノードの上限値が越えたと判断する
4. 代替ノードは自ノードの下位ノードの内の 1 台に，接続を試みた 2 台目のノードを振り分ける

**3.5　ノード数の増加に伴う負荷の軽減**　グリッド環境はネットワークで接続さ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，メモリ不足によってジョブがエラー終了するなどの問題が報告されている [4, 5]．スレッドの大量生成に関する問題を解決するために，DNAS では通信トポロジを階層型にすることで 1 つのノードに接続されるノード数を抑えている．しかしながら，階層型トポロジによる通信制御は有用な手法であるが，スレッドの大量生成に対する根本的な解決策ではない．

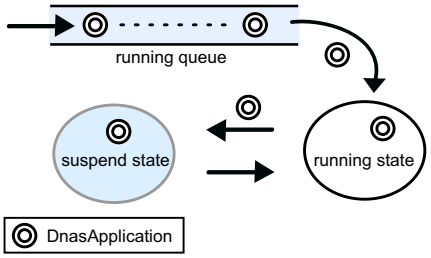


Fig.4　Control the running application

ここで，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．あるノードと通信を行っているノード数を $N$，システム制御，アプリケーション処理において発生するノードあたりのコネクション数をそれぞれ $Nc$, $Na$ とすると，ノードあたりのコネクション総数は $N(Nc+Na)$ で表される．$Nc$ 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あるため，その値は変化しない．また $Na$ はアプリケーションに依存している．そのため，同時に通信を行うノード数 $N$ を制限する仕組みを DNAS に導入している．具体的には，DNAS において送受信に関する通信機構を設計し，非同期 I/O を用いた多重化を行っている．

**3.6　アプリケーション制御機構**

ユーザが DNAS 上でアプリケーションを実行する際，DNAS が提供しているアプリケーション起動用プログラム (DnasRun) を用いる．DnasRun は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情報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．通知を受けた別ノードで動作する 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，どのノードでアプリケーションを実行するかを意識することなく，グリッド環境の計算資源を利用できる．この特徴は，2 章で挙げたアプリケーションを動作させるノードの自動選択に対応している．

また，DNAS は同時に起動するアプリケーションの数を制限することにより，計算資源に必要以上にジョブが投入されることを防ぐ機能を有する．実行可能状態になったアプリケーションはまず実行キューに保存する．次に，DNAS 上で動作しているアプリケーションの数が設定値に満たない場合は実行状態に推移し，設定値に達している場合は実行キューに保留する．実行中のアプリケーションが終了すると，DNAS デーモンは保留されているアプリケーションの有無を調査し，アプリケーションがある場合は実行状態に推移させる．

さらに DNAS はノードの負荷に応じて，実行中のアプリケーションの中断，再開する機能を有する．DnasRun

```
1  public void allot_task(int number_of_spaces){
2     int[] search_space = split_search_space(number_of_spaces);
3     Map <Long, Task> map = create_task(search_space);
4     Map <String, Serializable> hostlist = recv("sys-info", "hostname");
5     Iterator iterator = hostlist.entrySet();
6
7  while(iterator.hasNext()){
8     String hostname = iterator.next();
9     first_send(hostname, hostname, map);
10    }
11 }
12
13 public Map <Long, Task> recv_task(){
14    return recv("DoTaskApp", "AppResult");
15 }
```

Fig.5　A part of ”TaskAllot.java”

```
1  public void do_task(String my_hostname, String send_hostname){
2     Map <Long, Task> map = recv("TaskAllot", my_hostname);
3     Task task = map.getTask();
4
5     map = search_max_value(task);
6     first_send(send_hostname, "AppResult", map);
7  }
```

Fig.6　A part of ”DoTask.java”

によって起動された全てのアプリケーションは，DNAS 上でスレッドを用いて実行している．DNAS はすべて java 言語を用いて作成しているため，java の持つスレッドの機能を用いることで，アプリケーションの中断・再開を行うための機構を作成した．アプリケーションの起動制御の様子を Fig. 4 に示す.

**3.7　システム情報のデータ量**　DNAS が受け取る単位時間あたりのデータ量は，通信の頻度，最大下位ノード数，何ノード先の上位ノードまで自ノードのシステム情報を伝えるかの値によって決定する．これらの値は，DNAS の起動時にユーザが設定する．なお，1 ノードのシステム情報のデータ量は 150Byte から 200Byte 程度である．

例えば，通信の頻度を 15 秒，最大下位ノード数を 2，何ノード先の上位ノードまで自身のシステム情報を伝えるかの値を 5 とした場合，一番負荷のかかるサーバでは，最大 62（$2^1 + 2^2 + 2^3 + 2^4 + 2^5$）ノードの情報が一定時間に受信することになる．その情報量は約 10kByte/15 秒である．最大下位ノード数と各ノードにおけるシステムの負荷の関係に関しては 5.3 節で検証を行う.

## 4. DNAS アプリケーションの例

**4.1　アプリケーションの概要**　DNAS 上で動作するアプリケーションの例として，膨大な数値リストの中から最大値を探し出す全探索問題を実装する．実装するアプリケーションは，タスク割り当てアプリケーションとタスク処理アプリケーションの 2 つから構成される．

タスク割り当てアプリケーションは，下記の動作を行う.

1. 全探索領域を分割し，探索開始行と探索終了行の組を作成し，それを 1 つのタスクとする．
2. もし計算資源が起動していれば、計算資源へのタスクの割り当てを行う.
3. 処理が終了した計算資源から処理結果の情報を受け取る．

タスク処理アプリケーションは，指定された探索開始行から探索終了行までの探索を行う．探索が終了するとタスク割り当てアプリケーションに結果を送信する．

**4.2　プログラムの説明**　タスク割り当てアプリケーションのプログラムの一部を Fig. 5，タスク処理アプリケーションのプログラムの一部を Fig. 6 に示す.

まず，タスク割り当てアプリケーションから説明する．Fig. 5 の 2 行目から 3 行目で探索領域の分割と各ノードに振り分けられるタスクの生成を行っている．次に，4 行目から 5 行目でタスクを実行可能ノードのリストを作成し（3.2 節と 3.6 節を参照），7 行目から 10 行目でその実行可能ノードに対しタスクの割り振りを行う．今回は first_send 関数を用いて，タスク処理を行うノードにアプリケーションデータ（タスク）を直接送信するようにした．なお，アプリケーションデータに付与するデータ識別子は，各ノードのホスト名である.

次に，タスク処理アプリケーションのプログラムについて説明する．Fig. 6 の 2 行目において，自ノードのホスト名のデータ識別子と”TaskAllot”というアプリケーション識別子が付与されたデータを取得する．3 行目で，得られたアプリケーションデータからタスクを取得する．そして，5 行目でそのタスクを処理し，最大値となる目的のデータを取得する．そして，6 行目において，”AppResult”というデータ識別子を付与して，第一引数で指定したホスト（今回の場合，タスク割り当てアプリケーションが動作しているホスト）に送信する．最後に各ノードの結果を Fig. 5 の 13 行目から 15 行目に定義した関数で受け取る.

Table 1　Experimental environment(PC Cluster)

| | |
|---|---|
| Name | node1 - node5 |
| CPU | AMD Opteron 1.8GHz |
| Memory | DDR-SDRAM 512MByte |
| Network | Fast Ethernet(100Mbps) |

## 5. DNAS の基本機能の性能評価

**5.1　通信トポロジの動的再構成機能の検討**　DNAS の基本機能である通信トポロジの動的再構成機能の動作確認と，通信トポロジの再構成に必要な時間の計測を行う．なお，計測時間は 20 試行の平均値を用いた.

まず，ノードの故障による通信トポロジの再構成に要する時間について計測する．初期設定として，Table 1 に示す 5 台のノードを LAN ネットワークで接続し，

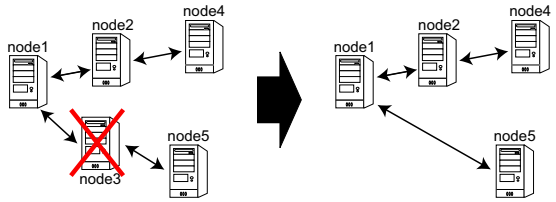


Fig.7　Dynamic restructuring function (when node is out of order)

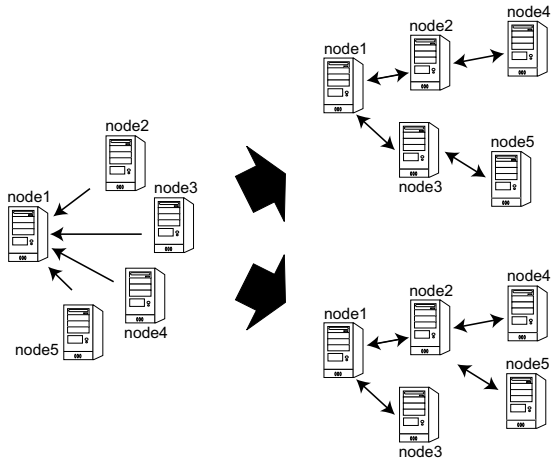


Fig.8　Dynamic restructuring function (when node is added)

Fig. 7 の左図の状態で node3 の DNAS デーモンを停止させた．

通信トポロジの再構成が行われた結果，node5 は node1 に接続先を変更し，Fig. 7 の右図に示す通信トポロジが構成された．node5 が node3 の故障を検知してから，通信トポロジの再構成が終了するまでの時間は 71ms であった．

次に，下位ノードの数が設定した上限値に達したことによる通信トポロジの再構成に要する時間について計測する．Table 1 に示す 5 台のノード上の下位ノードの最大接続数を 2 とし，node1 上で DNAS を起動し，node2 から node5 の上位ノードを node1 として DNAS を同時に起動した（Fig. 8 の左図）．

通信トポロジの再構成が行われた結果，Fig. 8 の右図の上下どちらかの通信トポロジになった（但し，どのノードが上位/下位ノードになるかは毎回異なる）．すべての下位ノードが node1 に接続を開始してから，通信トポロジの再構成が完了するまでの時間は 724ms であった．

### 5.2　大規模環境における動的再構成機能の検討

3 つの PC クラスタと 1 台のノード（計 230 ノード）を用いた大規模環境における DNAS の動作を確認する．実験環境を Table 2 に示す．実験の初期状態として，それぞれの PC クラスタの内部は LAN ネットワークで接続されており，各 PC クラスタと Node D とはインターネットで接続されている．また，下位ノードの最大接続数を 4 とし，すべての PC クラスタのゲートウェイノードの上位ノードを Node D とし，PC クラスタ内は全てのノードがゲートウェイノードを上位ノードとした設定で，すべてのノード上で DNAS を同時に起動した．

Table 2　Experimental environment(Grid)

| Name | Number of node |
|---|---|
| Cluster A | 10 |
| Cluster B | 216 |
| Cluster C | 3 |
| Node D | 1 |

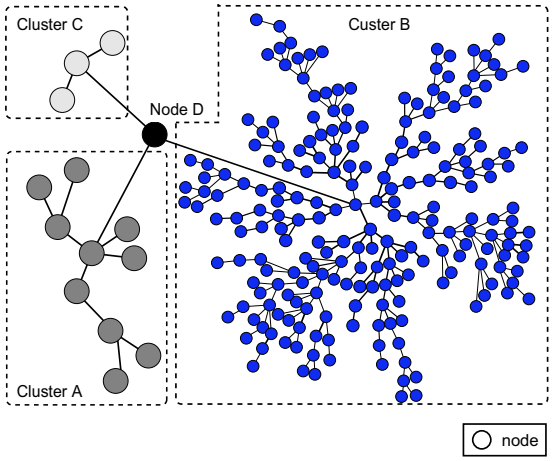


Fig.9　Communication topology on the Grid

上記の初期設定で DNAS を実行した結果を Fig. 9 に示す．すべてのノードにおいて，最大下位ノード数を 4 としたツリートポロジを構成することが確認できた．しかしながら，Cluster B のようにノード数が多い PC クラスタの場合，上位のノード（本実験の場合，ゲートウェイに近いノード）が少ない下位ノードしか持たないことがあるため，全体のツリートポロジの階層が深くなることがわかった．

### 5.3　最大下位ノード数とロードアベレージに関する検討

DNAS の特徴であるシステム情報の分散管理に対する優位性を検証するため，DNAS が動作するノード数を一定に保ったまま，最大下位ノード数を変化させた場合のロードアベレージの変化に関する実験を行った．

DNAS が動作するノード数を 216，各ノードの最大下位ノード数を 2, 4, 8, 16, 32, 64, 128，DNAS 間の通信の頻度を 15 秒とした．最大下位ノード数が少ない値ほど，より分散している環境といえる．また，自ノードのシステム情報を何ノード先の上位ノードまで送信するかの値を十分大きくとり，どの最大下位ノード数の場合も，すべてのノードのシステム情報が最上位ノードに集まるようにした．そして，最大下位ノード数を変

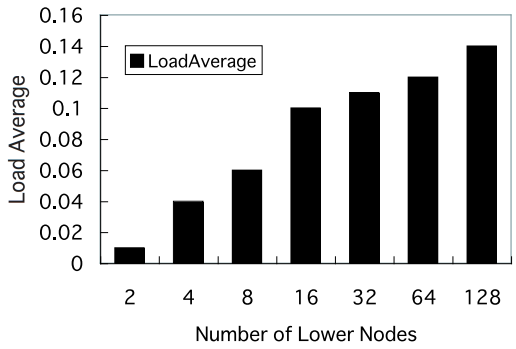


Fig.10　Relation between number of lower nodes and load average

Table 3　Experimental environment(PC Cluster)

| | |
|---|---|
| CPU | AMD Opteron 1.8GHz * 2 (Dual CPU) |
| Memory | DDR-SDRAM 2GByte |
| Network | Gigabit Ethernet(1000Mbps) |

化させた場合における最上位ノードのロードアベレージを計測した．結果を Fig. 10 に示す．なお，本実験に用いたノードのスペックを Table 3 に示す．

結果を Fig. 10 の結果より，最大下位ノード数が少ないほど，つまりより分散させた環境の方がノードにかかる負担が少ないことがわかった．

## 6. 関連研究

GMWS[3] はマスタワーカ型の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[9] を利用し，強固なセキュリティ機構を有している．しかし，システムの起動時に構成する計算資源数を決定する必要があるため，システム起動後に計算資源の追加および削除ができない．また全てのノードが 1 台のノードに接続されるため，そのノードに負荷が集中する．

Ninf-G[6]/Ninf-1[7] を利用したモデル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実現している．また，Ninf-G/Ninf-1 を利用したモデルは，インターネット上ではセキュアな Ninf-G を用い，ローカルネットワーク上には高速な Ninf-1 を用いることで，セキュリティと性能の両立を図っている．しかしながら，Ninf-G/Ninf-1 はユーザが明示的に階層構造を設計する必要があり，また，GMWS と同様にシステム起動後に計算資源の追加，削除ができない．

Jojo2[1] ではマスタワーカ型をもとにした階層構造を定義しており，計算資源の動的な変化を前提としたプログラミングモデルを提供している．Phoenix[2] は階層型のネットワークに適応可能な並列計算ライブラリであり，WAN において独自の API を用いた集合通信を可能とする．Jojo2，Phoenix ともに計算環境が動的に変化することを想定している点では DNAS と同じであるが，環境の変化に応じて動的に通信トポロジを変化させることはできず，また計算に用いるノードの負荷率などの取得をアプリケーションから行うことで，計算資源の状況によってアプリケーションの動作を柔軟に変化させることはできない．

## 7. まとめと今後の課題

本論文では，グリッドの計算資源のシステム情報とアプリケーションデータを簡易に取得できる機能を持つグリッドミドルウェア DNAS を提案した．ユーザは DNAS を用いることで，システムの動的変化に対応できるアプリケーションの開発と連携を行うことができる．また，DNAS は通信トポロジの動的再構成機能とスレッド生成数の制限などの機能を有し，それぞれが正常に動作することを確認した．

今後の課題としては以下が挙げられる．

- グリッドでよく用いられる実アプリケーションを DNAS で実装し，データ通信のオーバヘッドに関する検討を行う
- 異なる複数のユーザのアプリケーション同士を連携させることを考えた場合，ユーザ認証の仕組みを DNAS に追加する必要がある
- 現在の実装では，最上位ノードに障害が起きた場合については対応していない．その対策として，DNAS 起動時に最上位ノードのみを他のノードと区別しておき，もし最上位ノードに障害が発生した場合は，最上位ノードが持っていた下位ノードの内の 1 つが最上位ノードとなり，他の下位ノードは新しい最上位ノードを上位ノードとして自動的に登録する仕組みを作成することが挙げられる
- 3.6 節で述べたアプリケーションの制御機構については，condor[8] などの既存のジョブスケジューリングシステムを利用することで，グリッドの計算資源をより効率的に利用できるようにする．また，複数のノードを用いてアプリケーションを実行している場合，計算に用いているノードのアプリケーションが中断すると，アプリケーション全体の挙動に影響を与える可能性がある．そのため，負荷の高いノードから負荷低い別のノードにアプリケーションを自動的に委譲させる機構を作成する
- 5.2 節で述べた DNAS が構成する通信トポロジは，ノード数に対して階層が深くなる可能性がある．また，ノードの追加，削除を行った場合も同様の可能性があるため，任意のタイミングで通信トポロジを均等な深さの階層にするために再構成する機能が必要である

## 参考文献

(1) 青木仁志, 中田秀基, 田中康司, 松岡聡: 動的なノード群構成を備えた階層型グリッド環境: Jojo2, 先進的計算基盤システムシンポジウム SACSIS2006, pp.101-108 (2006).

(2) 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).

(3) 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).*

(4) 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.

(5) 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.

(6) Yoshio Tanaka, Hiroshi Takemiya, Hidemoto Nakada and Satoshi Sekiguchi. Design, implementation and performance evaluation of GridRPC programming middleware for a largescale computational Grid. Proceeding of 5th IEEE/ACM International Workshop on Grid Computing (2004).

(7) 中田秀基, 高木浩光, 松岡聡, 長嶋雲兵, 佐藤三久, 関口智嗣. Ninf による広域分散並列計算. 情報処理学会論文誌 vol.39, no.6, pp. 1818-1826 (1998).

(8) Michael Litzkow, Miron Livny, and Matt Mutka. Condor - A Hunter of Idle Workstations, Proceedings of the 8th International Conference of Distributed Computing Systems, pp.104-111 (1988).

(9) I. Foster and C. Kesselman: Globus: A Metacomputing Infrastructure Toolkit, International Journal of Supercomputer Applications, Vol.11, No.2, pp. 115-128 (1997).