

階層型グリッドミドルウェア DNAS3 の設計と実装

折戸 俊彦

Toshihiko ORITO

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る、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いう問題がある。このような問題を吸収するために、クライアントサーバ型の Ninf や NetSolve、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、DNAS2 など、様々な通信ライブラリやグリッドミドルウェアが開発されている。

グリッドミドルウェアにはそれぞれ用途に合わせた特徴があるが、一般的に管理ノードが存在するようなグリッドミドルウェアには特定のノードに負荷が集中するという問題点がある。また、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。そのため、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が、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。

DNAS2 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を流れるデータを中継する仕組みを有している。本研究では DNAS2 の問題点を明らかにし、DNAS2 を基に Distributed Network Application System3(以下、DNAS3) の設計と実装を行う。

2 DNAS2

DNAS はグリッド環境で動作し、状況に応じて動的に通信トポロジを変えることができる P2P 指向のミドルウェアである¹⁾。DNAS2 は DNAS をもとに設計されており、DNAS の特徴を踏襲しつつセキュリティに関する点や通信機構、拡張性といった DNAS において問題とされていた点が改善されている。DNAS2 の基本アーキ

テクチャを Fig. 1 に示す。Fig. 1 のように DNAS2 では各ノードにおいて DNAS デーモンを起動させる。DNAS デーモンは隣接する DNAS デーモンとのみ直接通信を行い、その他の DNAS デーモンとは各 DNAS デーモンがネットワーク上を流れるデータを中継することにより間接的に通信を行っている。一方、DNAS 上で動作するユーザアプリケーションはローカルで動作する DNAS デーモンとのみ通信を行う。ノード間の通信は DNAS デーモンによって自動的に行われるため、アプリケーション同士で直接通信する必要はない。

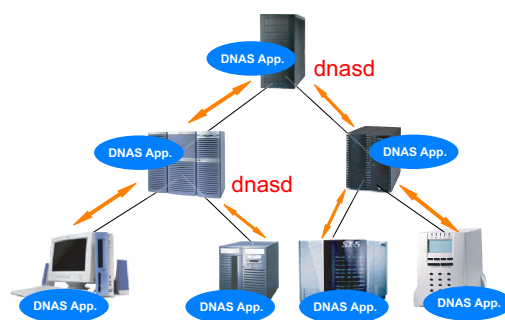


Fig. 1 DNAS2 のアーキテクチャ

DNAS2 は後述する 3 つの特徴を持ち、障害に強く、各ノードの負荷を分散させたシステムを構築することができる。

2.1 P2P 指向型システム

DNAS2 は P2P 指向型のシステムであり、任意のタイミングでシステムの一部として利用ノードを参加させることができる。また、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利用ノードの削減を行うことができる。

2.2 ツリートポロジ

DNAS デーモン間の通信のために論理的な通信路を設定し、システム全体としてツリートポロジを構成する。各ノードは隣接する一部のノードとのみ直接通信を行い、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情報を共有することができる。

2.3 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。また、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。DNAS2 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機能を3種類有している。

- 上位ノードに障害が発生し通信できなくなった場合
- 下位ノード数が設定された上限値を超えた場合
- システムのロードアベレージ値が閾値を超えてしまった場合

DNAS2 におけるトポロジの動的な再構成の様子を Fig. 2 に示す。

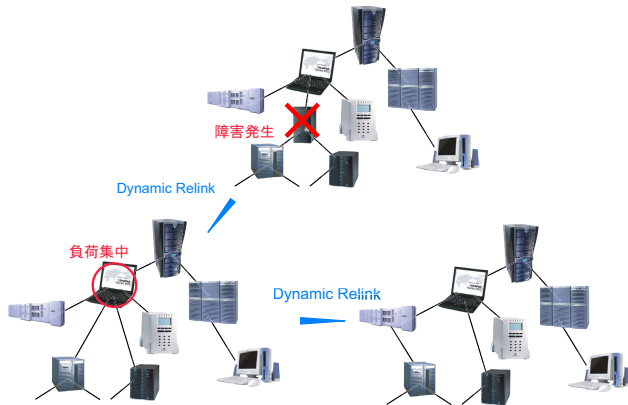


Fig. 2 通信トポロジの動的な再構成

2.4 DNAS2 の問題点

- アプリケーションデータへのアクセス

DNAS2 ではユーザデータにタグ（データ識別子）を付与し、それらを各ホスト毎に管理している。全てのデータはホスト毎にタグとデータのペアで表され、ホスト識別子、データ識別子を指定することによりユーザは任意のデータを取得できる。ユーザは複数のアプリケーションを同時に起動することが可能であり、それぞれのアプリケーションがローカルで動作する DNAS デーモンにデータを送信する。そのため、複数のアプリケーション間で同一のデータ識別子が用いられると、データが上書きされてしまう。また、それぞれのアプリケーションデータに対してアクセス制限をかけることができない。

- アプリケーション起動時の問題

一般的に不特定多数ユーザが計算資源を共有して利用する場合、ユーザはリソースの状況を把握して利用するリソースを選択しなければならない。しかし、グリッド環境のリソースは地理的に分散し、資源の追加および削減が頻繁に行われるため、ユーザはリソースを全て把握することは非常に困難である上に、ユーザはリソースの情報を取得することすらできない可能性がある。またグリッド環境の大規模化に伴い、ユーザがアプリケーションの起動までに行う作業量も増加するということがわかっている²⁾。

グリッド環境のような大規模な実行環境ではユーザはリソースを把握できない上に、アプリケーション起動時に煩雑な作業が必要であるという問題がある。

- スレッド大量生成の問題

DNAS2 は P2P 型のシステムであり動的なノードの増減を許容しているため、1 ノードあたりの接続ノード数が時間とともに変化する。しかし、1 ノードに接続要求が集中するとスレッドが大量生成されてシステムがダウンしてしまう問題が確認されている²⁾。

3 DNAS3 の設計と実装

DNAS3 は DNAS2 の問題点を改善するように一から設計する。アプリケーションデータへのアクセス制御およびアプリケーションの起動制御と実行制御を実現し、1 ノードあたりの接続ノード数の増加に伴うスケーラビリティの向上を目的としたスレッド管理機構を強化する。

3.1 アプリケーションデータの管理

DNAS2 では全てのアプリケーションデータはホスト識別子とデータ識別子によってアクセスされ、全てのアプリケーションデータは同一の領域に保存されている。他アプリケーションのデータを誤って書き換えないようにするためにはユーザはそれぞれのアプリケーションがどの識別子を使用しているかを把握しておかなければならない。しかし、他ユーザのアプリケーションが用いている識別子は通常知ることができず、DNAS 側でデータの衝突が起こらない設計にする必要がある。

DNAS3 では新たにアプリケーション識別子を付与し、それぞれのアプリケーションが使用するデータ領域を完全に分離し、データの衝突が起こらないような設計になっている。

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他

アプリケーションに開示したくない場合には他アプリケーションからの参照を排除することが可能である。

3.2 アプリケーションの起動

DNAS2ではアプリケーションの起動はユーザにまかされており、ユーザはDNASデーモンが動作しているノードでアプリケーションを起動する。そのためには、ユーザ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る。

DNAS3では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。アプリケーションの起動の仕組みをFig. 3に示す。

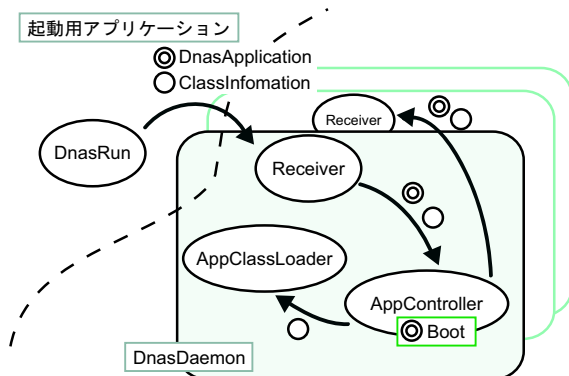


Fig. 3 アプリケーションの起動

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。

3.3 アプリケーションの制御

DNASでは1つのDNASデーモン上で複数のアプリケーションを実行することを許容している。また、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が非常に困難である。

3.2節で述べたようにDNAS3ではアプリケーションの起動はDNASデーモンによって行われ、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の動作を制御する仕組みである。

3.3.1 アプリケーションの同時起動数の制御

DNAS3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Fig. 4に示す。ここでは、模式的にDNASデーモンによって起動されるアプリケーションを球で表す。

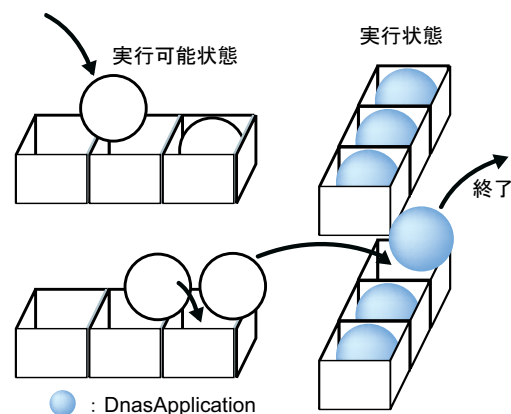


Fig. 4 アプリケーションの起動制御

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Fig. 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。

3.3.2 負荷に応じたアプリケーションの動作制御

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

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。DNAS3は実行中のアプリケーションの動作を負荷に応じて制御する仕組みを有している。

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中断させる。アプリケーションは後のものから順に選択され、負荷が閾値を下回るまで中断される。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。以上の仕組みにより資源に高負荷がかかっている状態が続くことを回避することができる。

3.4 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、スレッド処理を用いた方法に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^{2, 3)}。

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 N_c 、 N_a とすると、単位時間に発生するコネクション数 TN_c は $TN_c = N(N_c + N_a)$ で表される。

ここで、 TN_c の削減について考える。 N_c 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、 N_a 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。DNAS3 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いる。

3.4.1 送信に関する処理

DNAS デーモンは送信要求を受け取ると、送信データを送信キューに格納する。各 DNAS デーモンにおいて送信処理において同時に生成されるスレッド数をあらかじめ設定することができ、スレッドに空きがあれば送信データは送信され、空きがなければ保留される。保留されたデータはスレッドに空きができ次第順次送信される。このような仕組みにより送信に用いられるスレッド数は一定を保ち、スレッドが大量生成されることはない。また送信に利用されるスレッドは再利用されるため、スレッド生成に必要なリソースを節約することができる副次的な効果もある。

3.4.2 受信に関する処理

DNAS デーモンは接続要求や受信データの有無を監視しており、要求やデータが存在するとそれに対応した処理を行う。その際にはマルチスレッド処理による処理の多重化ではなく、非同期 I/O を用いて1つのスレッドでマルチプレクサ処理を行う。1つのスレッドで非同期で全ての受信データを処理するため、スレッドの大量生成を防ぐことができる。

4 稼働実験

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について実験を行う。実行環境を Table 1 に示す。

Table 1 実行環境

name	gaia1 - gaia10
CPU	PentiumIV 2.8GHz * 2
memory	512M * 2

4.1 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 Fig. 5 のようになった。

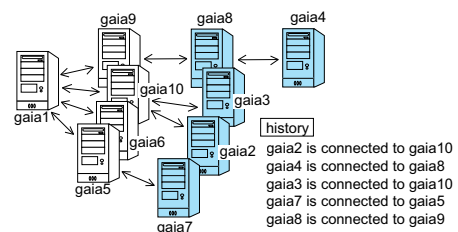


Fig. 5 1 ノードに対する接続ノード数の制限

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、再構成が行われた。Fig. 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。

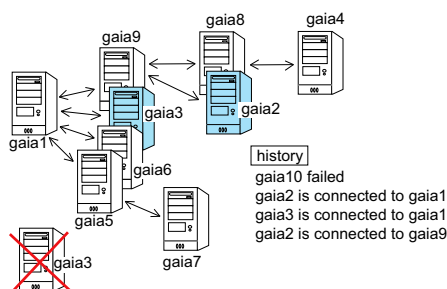


Fig. 6 障害に対する動的再構成機能

4.2 アプリケーションの実行制御

DNAS デーモンが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した。Fig. 7 は閾値が 0.5 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、処理内容はそれぞれ無限ループプログラムである。Fig. 7 はアプリケーションがロードアベレージが閾値である 0.5 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 0.1 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、下回ると再開されていた。

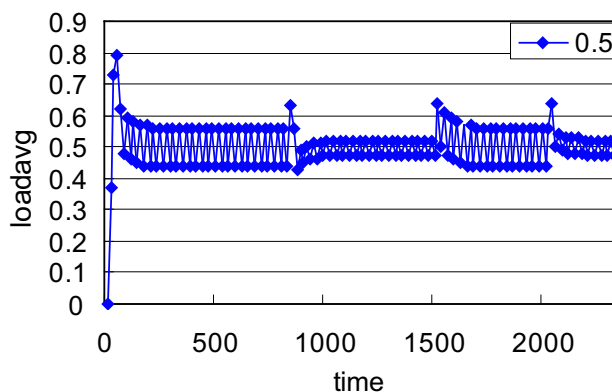


Fig. 7 アプリケーションの実行制御

5 今後の課題

今後の課題としてはユーザ認証、受信データサイズが大きくなったときの処理が挙げられる。

- ユーザ認証

DNAS3 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。

- 受信処理

DNAS3 の受信処理では受信データサイズが大きくなり受信処理に時間がかかるような処理が頻繁に発生すると、その処理がボトルネックになり全ての処理のオーバーヘッドが高くなることが考えられる。解決策としてはデータサイズが大きい場合は別スレッドで実行する、メタデータを扱って大きなデータサイズの情報がネットワークを頻繁に流れないようにするなどが挙げられる。

6 おわりに

本稿では階層型グリッドミドルウェアである DNAS3 の設計と実装を行った。DNAS3 は DNAS2 を基に開発され、動的再構成機能により通信トポロジを動的に変化させることができることが確認された。接続ノード数の増加に伴うスケーラビリティの向上を目的としたスレッド管理機構、アプリケーションに関する機構としてアプリケーションデータへのアクセス制御、アプリケーションの起動制御および実行制御が強化され、スレッドの大量生成を防ぐ効果やグリッド資源に高負荷がかかっている状態が続かないようにする効果が確認された。

今後の課題としてはユーザ認証、受信データサイズが大きくなったときの処理が挙げられる。

参考文献

- 1) 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.
- 2) 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, 2005 No.5, pp. 260-261.
- 3) 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.