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

2003－HPC－95 (16)
2003／8／5

グリッドコンピューティングにおけるモニタリングシステムの自律的構成

白勢 健一郎[†] 小川 宏高[†††]
中田 秀基[†††,†] 松岡 聡[†,††]

グリッド環境における計算機やソフトウエアのモニタリングシステムは、役割ごとに分割されたコンポーネントから構成されている。コンポーネント間の依存関係をふまえた設定の複雑さやモニタリングシステムを常時稼働させる必要性を考えると、人間の手で管理のすべてを行うには限界がある。このため自律的管理の仕組みが必要である。本研究では、モニタリングシステムを自律的に管理するシステムを提案し、その機能の一部として、既存のモニタリングシステムの自動設定と障害復旧を行うシステムを実装した。実装したシステムをキャンパス内のグリッド環境のテストベッドで動かした結果、10 個の計算機に対して約 2 分程度の時間で設定を完了する事ができた。

Automatic Management System for Monitoring System on the Grid

Ken'ichiro Shirose,[†] Hirotaka Ogawa,[†††] Hidemoto Nakada[†††,†] and Satoshi Matsuoka[†,††]

Monitoring CPU, memory and disk usage and network performance is needed in Grid Computing environment. Generally, monitoring systems on Grid Computing consist of multiple components based on own functions. It is difficult to configure them all manually, because there are many dependencies between them and monitoring systems must run continuously. We propose a automatic management system for monitoring system on Grid Computing, implement a part of functions and evaluate its usefulness.

1. はじめに

物理的に分散した計算機を協調させて利用するグリッド環境においては、CPU、メモリ、ディスク、ネットワークの利用状況、ミドルウェアやアプリケーションの動作状況を含むさまざまなモニタリング情報の収集、管理および提供を行うシステムが必要である。こうしたモニタリング情報は、ジョブスケジューラの判断材料になるだけでなく、グリッド環境のユーザや管理者・サポートスタッフにとってはデバッグや障害検知に有益な情報源になるからである。とりわけ後者は、モニタリングシステムを基盤としたグリッド環境の耐故障性・自動管理の実現を目指す、IBM の Autonomic Computing プロジェクト[1] に代表されるように、近年注目を集めつつある。

NWS[2] を始めとするグリッド環境向けのモニタリングシステムは、単一の機能を有する複数のコンポーネントから構成されており、かつこれらのコンポーネントは相互に依存していることから、一般に人手による煩雑な設定を要求する。また、製品水準のグリッド環境では持続的な運用、動的かつ段階的な拡張が求められ、こうした人手による設定の負担が膨大になるであると考えられる。

我々は、このような負担を軽減するために、グリッド環境でのモニタリングシステムの自律的管理の実現を目指している。本稿では、自律的管理の仕組みを提案するとともに、その予備的な実装として NWS の自動設定と自動障害回復を行うシステムを実現した。また、東工大の大岡山・すずかけ台の両キャンパスをまたがるグリッド環境 TitechGrid をテストベッドとして本システムの予備的な評価も行ったので、その結果も併せて報告する。

2. グリッド環境のモニタリングシステム

ここではグリッド環境（およびその計算資源として一般的な PC クラスタの）モニタリングシステムに要請される機能、そこから導き出される一般的な構成を議論する。

† 東京工業大学
Tokyo Institute of Technology
†† 国立情報学研究所
National Institute of Informatics
††† 産業技術総合研究所
National Institute of Advanced Industrial Science and Technology

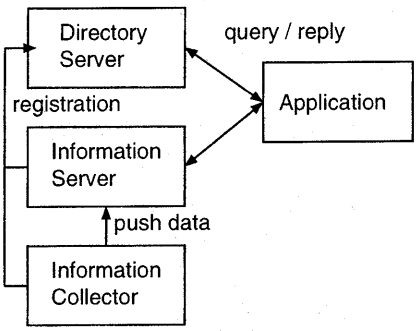

図 1　モニタリングシステムの機能要素の関係

2.1　モニタリングシステムの機能構成

グリッド環境におけるモニタリングシステムは、各計算ノード上で、CPU、メモリ、ディスク、ネットワークの利用状況、ミドルウェアやアプリケーションの動作状況を含むさまざまなモニタリング情報の収集、管理および提供を行う。ここでは、これらの機能に加えて、モニタリングシステムが提供する情報を利用するアプリケーションを含めて考える。

モニタリング情報の収集は一般にすべての計算ノード上で行う必要があるが、その情報の管理・提供はすべてのノードで行う必要はない。また、モニタリング情報を利用するアプリケーションは、基本的に情報収集を行うノード、管理・提供を行うノードとは独立に任意のノード上で実行され得る。また、グリッド環境では、一つのノードが中央集権的にモニタリング情報を管理・提供することはほとんど不可能であることから、どのノードがどのような情報を管理・提供するかというメタ情報を管理・提供するノードが必要である。

以上からモニタリングシステムは少なくとも以下の4つの機能要素を必要とする。この分類は Xuehai ら[7] によるものである。

Information Collector　各計算ノード上で、CPU、メモリ、ディスクの使用率やネットワークの性能等のモニタリング情報を測定・収集する機能

Information Server　収集された情報を管理・提供する機能

Directory Server　Information Server が、どの Information Collector のどのような情報を管理・提供するかを管理・提供する機能

Application　Information Server の情報を利用して適切な動作をするアプリケーション機能

これらの機能要素の関係を 1 に示す。

2.2　既存のモニタリングシステムとの対応

既存のグリッド環境向けモニタリングシステムの実装としては、以下のものがある。

NWS[2]　UCSD で開発されたグリッド環境向けモニタリングシステム。モニタリング情報を利用した "User Interface" と呼ばれる、モニタリング情報とそれに基づく将来の資源の利用予測を表示するアプリケーションを特徴とする。

NWS では、end-to-end のネットワーク性能計測によるトラフィック増大の緩和のためにクリークという機構を持つ。まず、ネットワーク構成に従って計算ノードをクリークに分割しておく。クリーク内の計算ノードでは全対全の性能計測を行う一方、クリーク間では代表ノード間の計測のみを行う。任意のノード間の性能は、これらの情報を合成することで近似的に得ることができる。

Monitoring and Discovery Service (MDS)[3]　Globus Toolkit[4] で用いられるグリッド資源情報を管理・提供するフレームワーク。

R-GMA[6]　Global Grid Forum で定義された Grid Monitoring Architecture (GMA) の EU DataGrid プロジェクトによる実装。

Hawkeye[5]　Wisconsin 大学で開発されている Condor をベースとしたモニタリングシステム。

表 1　モニタリングシステムの比較

	Information Collector	Information Server
NWS	Sensor	Memory Host
MDS	Information Providor	GRIS
R-GMA	Producer	Producer Servlet
Hawkeye	Module	Monitoring Agent
	Directory Server	**Application**
NWS	Name Server	User Interface
MDS	GIIS	N/A
R-GMA	Registry	N/A
Hawkeye	Manager	N/A

1 に 2.1 に示した機能構成とこれらのモニタリングシステムとの対応を示す。この表に示される通り、既存のシステムの多くが同様の機能構成を持つことが分かる。

3.　モニタリングシステムの自律的管理

3.1　概　　要

本研究では既存のモニタリングシステムを活用し、スケーラブルで、動的に拡張可能で、耐故障性を実現し、自動的にコンポーネントの設定や障害修復を行う、モニタリングシステム管理のフレームワークの実現を目指す。

3.2　自律的管理に必要な機能

モニタリングシステムの自律的管理に必要な機能は、モニタリングシステムのコンポーネントのコントロール、モニタリングシステムと管理システム自体の設定

の動的な変更、管理システム間の相互通信の 3 つである。

コンポーネントのコントロールは自動設定や障害復旧のために必要である。コンポーネントの設定変更にともなって、新しい設定を反映させるために、コンポーネントの再起動が必要になることもある。

設定の動的な変更を可能にする事は、モニタリングシステムを取り巻く環境の変化に対応するために必要である。また、管理システムそのものの振舞も必要に応じて変化する必要がある。

モニタリングシステムの設定でとりわけ問題になるのはネットワークの計測に関係するものである。ネットワークトラフィックの軽減のため、計算機を複数のグループに分割して、グループ内の計測は end-to-end で行い、グループ間の計測は代表同士の計測結果を用いることにする。このグループ分けと代表の決定も自動的に行う必要がある。

管理システム間の相互通信は、個々の管理システムが自前でコンポーネントや自分自身の設定を変更する際の情報となる。また管理システム同士を協調させる事で、すでに運用されている複数のモニタリングシステムの統合が可能になる。この結果、全体としてスケーラビリティーを持つモニタリングシステムを実現する事が可能である。

3.3 自律的管理機構の設計

モニタリングシステムのコンポーネントの自律的管理は次のようなステップを定期的に行う事で実現できる。(図 2) なお、管理システム間の通信に関わる事柄については現在検討中である。

- 計算機のネットワークトポロジーの推定および、計算機や計算機の上で稼働しているモニタリングシステムのプログラムが期待通りに動いているかどうかのチェック
- グループ分け (新規設定時) またはグループエントリーの編集
- コンポーネントの配置決定
- プログラムの実行と情報サービスへの情報変更

プログラムの動作チェックはモニタリングシステムに付属するプログラムや独自のチェックプログラムを用いて行う。計算機が動作しているかどうかのチェックはモニタリングシステム全体に影響を及ぼす障害の復旧のために必要である。

初期設定の場合は、グループ分けの結果とモニタリングシステムのコンポーネント間の依存関係に基づいてモニタリングシステムの構成を決定し、必要なプログラムを実行する。

モニタリングシステムが運用を開始した後は、障害の発生した計算機をグループから取り除いたり、復旧した計算機や新規に追加する計算機をグループに分けたりする作業を行う。そして変更されたグループをもとにして欠けたコンポーネントの復旧をコストのかからない方法で行う。

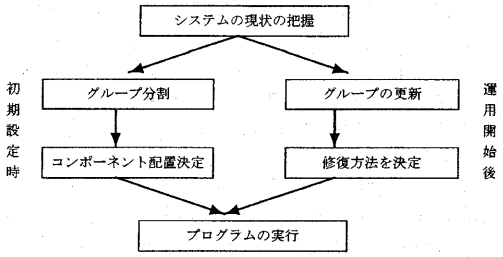

図 2 提案するモニタリングの自律的構成システムの流れ図

4. モニタリングシステムの自動設定システムの実装

モニタリングシステムを自律的に管理するシステムの一部の機能として自動的に NWS の設定と障害復旧を行うシステムを実装した。

実装したシステムは一つの計算機上で動かす。また次のような方針で NWS の設定を行う事にした。

- センサーホストは指定された計算機すべてで稼働させる。
- ネームサーバーは指定された計算機の中の 1 つの計算機で稼働させる。
- RTT の平均を尺度にした距離から計算機をグループに分割し、各グループにつき 1 つのメモリホストを稼働させることにする。
- ネットワークの計測のために計算機のグループ分割を行う。

以下で設計の所で説明した各段階をどのようにして行うかを説明する。

4.1 データ収集

計算機のグループ分けでは計算機の近さをもとにする。実装したシステムでは RTT の大きさから各計算機間の近さとする。片方の計算機でプログラムを動かせば RTT を求められる`ping` コマンドを使いる。

モニタリングシステムを設置する計算機のリストを入力として、これを元にしてデータを収集するのに用いるシェルスクリプトと、各計算機で ping を実行し必要なデータを取り出すのに用いるシェルスクリプトを生成する。シェルスクリプトを作成した計算機上で前者が実行され、各計算機に対して

- ping を実行するシェルスクリプトとデータ抽出用 awk スクリプトの転送
- 転送したシェルスクリプトへの実行権限の付与
- シェルスクリプトの実行 (具体的には次の事を行う)
  - ping コマンドの実行 (逐次処理で結果はファイルに保存する)
  - awk スクリプトを使ったデータ抽出

• 抽出されたデータを納めたファイルの取得

が行われる。最終的には各計算機ごとに求めた RTT のデータを含むファイルを集める。このデータを基にして後のステップを実行する。各計算機ごとに RTT の平均値が一番小さい計算機を求める。これを「一番近い」計算機であるとする。

次に NWS のネームサーバをどの計算機で実行するかを決定するための指標の計算をする。これは ping で応答のあった計算機のからの RTT の平均値である。

4.2 計算機のグループ化

上記の方法で求めた RTT の平均値をもとにして、計算機のグループ分けを行う。

各計算機に対して上述の方法で求めた「一番近い」計算機に注目する。「一番近い」計算機が既に作られているグループに所属する場合は、その既に作られているグループと現在注目しているグループを連結させる。「一番近い」計算機が現在注目しているグループに属している場合は、ここでこのグループを既存のグループとして、新しくまだグループ分けされていない計算機の一つからなる、新しいグループを作成する。また「一番近い」計算機がまだグループ分けされていない場合は、グループ分けされていない計算機を新たに選択しこれらでグループを作る。

グループの相互関係については次の節で説明するような方法で NWS ネームサーバを動かす計算機を決定し、その計算機を含むグループを中心にハブを構成するようにする。

4.3 NWS の運用構成の決定

NWS ネームサーバは ping コマンドで RTT を一番多く求められた計算機の中で他の計算機からの RTT の平均が一番小さい計算機とする。

各グループのメモリホストを動かす計算機は、グループ内の計算機が「一番近い」と最も指定している計算機とする。

ネットワークの計測のためのクリークはグループに対応させる。グループ間についてはメモリホストを稼働させる計算機の間で Point-to-Point の計測が出来るようにクリークを設定する。

4.4 設定ファイルの生成

NWS は設定ファイルとコマンドのいずれかで設定をする事ができる。動的な設定に対応ために、コマンドのオプションによって、個別の設定をするようにした。各計算機で実行するプログラムのための起動シェルスクリプトの中で、必要な設定を引数にして渡す方法を取る。

NWS の実装では、メモリホストはネームサーバがどこで実行され、どのポートを用いているかという情報を必要とし、センサーホストはこのネームサーバの情報に加えて、各メモリホストがどこで実行されどのポートを用いているかという情報を必要としている。

実装したシステムでは各ホストに対してそれぞれの設定に必要なものを属性として持つオブジェクトを Java のクラスとして実現している。これをフォーマットと呼ぶことにする。依存する側のフォーマットの属性の一つとして依存対象のフォーマットを持つことで依存関係を表現する。ネームサーバ、メモリホスト、センサー、アクティビティ(センサーに計測する対象を追加するのに用いるコマンド) の順番にフォーマットを管理システムが埋めて行く。

4.5 運用開始

作ったフォーマットから SSH で必要なプログラムを逐次実行するシェルスクリプトを生成する。具体的にはコンポーネントの依存関係に基づいてネームサーバ、メモリホスト、センサー (CPU の利用率の計測を開始する)、アクティビティー (グループ間のネットワークの性能計測のみ) の順にプログラムを実行するようになっている。集中管理する計算機から SSH を使って各計算機に必要なジョブを投入する。前述の順にプログラムが実行される。

4.6 障害修復

想定する障害として 1. 以前動かしたはずの NWS のプログラムが動いていない場合、2. ハードウエアや OS のの不調により計算機そのものが使えない場合、の二つを考える。

1. のケースの場合は以前の設定情報に基づいてプログラムの再実行を行えば良い。2. のケースで不調になった計算機上で NWS のネームサーバやメモリホストのプログラムが動いていた場合は、他の計算機でこれらのプログラムを実行することにして、他の依存関係のあるコンポーネントのプログラムも設定を変更して再実行する必要がある。

具体的には、NWS のネームサーバが動いていた計算機が不調になった場合、メモリホストがどの計算機で動いているか、また各計算機でどのリソースのモニタをしているかといった一切の情報が得られなくなる。したがって代替の計算機を選んでプログラムを実行する必要がある。メモリホストやセンサーのプログラムは、実行する時にネームサーバがどの計算機で動いているかを引数として指定する事になっているため、今まで動かしていたすべてのコンポーネントを新しい設定情報に基づいて再実行する必要がある。

メモリホストが動いている計算機が不調になった場合は、グループ内の計算機から代替の計算機を選択する。センサーはメモリホストの稼働している計算機を引数として受け取るので、同じグループの計算機上で実行されているセンサーのプログラムを新しい設定に基づいて実行しなおす。

実際に修復を行う手順は次の通りである。まず、ps コマンドを各計算機で実行して NWS のコンポーネントの稼働状況を調べる。過去の設定情報は NWS 自動管理システムを動かす計算機上のファイルとして保存されていて、この情報と実際のプロセス稼働状況を比

較する。SSH が実行できなくてプロセスの稼働状況が把握できない計算機は、ハードウエアか OS が不調であるとみなされ、計算機上でネームサーバやメモリホストが動いていた場合は代替の計算機を選択する。プログラムの再実行が必要なものに関してのみシェルスクリプトが生成される。最後に設定の時と同じように生成されたシェルスクリプトを実行する。

5. システムの評価

実装したシステムの有効性を確認するため、実環境での評価を行った。評価環境は東京工業大学の Titech Grid である。これは東京工業大学の二つのキャンパス上に分散したクラスタ 15 台から構成されている。キャンパスのさまざまな所に配置されている。これらは 100Mbps のネットワークインターフェイスを持ち、ギガビットのバックボーンにつながっている。今回はクラスタから代表の計算機を選んでそれらの計算機上に NWS のプログラムを設定して実行した。

5.1 初期設定にかかる時間

台数を変えて実装したシステムを実行した結果を表 2 に示す。結果は、台数に比例して増加している事が分かった。実行時間の内訳は ping によるデータ収集と NWS のコンポーネントの実行で半分ずつを占めている。

またグループ分けの結果、二か所のキャンパスに分かれた 10 台の計算機を、キャンパスごとにグループ分けすることができた。(図 3)

5.2 障害復旧

実装した障害復旧機能の内、ネームサーバやメモリホストの稼働している計算機が不調になった場合に、コンポーネントの再設定をする実験を行った。

まず、メモリホストの稼働している計算機が不調になった場合を扱う。図 4 の上部は障害が発生する前の様子を表している。計算機が二つのグループに分かれており、tgn015001 と tgn005001 という計算機上でメモリホストのプロセスが動いている。各計算機で稼働しているセンサーのプログラムは所属しているグループのメモリホストにデータを送信している。ここで tgn015001 という計算機が利用不能になったとする。すると同じグループの計算機で動いているセンサーはデータの送り先を失ってしまう。図 4 の下部は実装したシステムを適用した結果を表す。メモリホストを動かす代替の計算機として tgn013001 が選ばれる。グループ内のセンサーは tgn013001 で稼働するメモリホストにデータを送信するように設定され、プログラムが再実行される。

次に、ネームサーバの稼働している計算機が不調になった場合を扱う。図 5 の上部ではネームサーバが tgn015001 で稼働している様子を表す。他の各コンポーネントは起動する際にこのネームサーバに、自分自身の情報を登録する。tgn015001 が不調になると NWS のデータ取得インターフェースはデータを取得する際に必要とするネームサーバへの問い合わせが出来なくなる。図 5 の下部は tgn015001 が不調になった後、実装したシステムを適用して設定の変更が行われた結果を表している。ネームサーバの稼働場所は tgn005001 に移ったので、今まで稼働していた他のすべてのコンポーネントは設定を変更した上で再起動される。

上述の状況でネームサーバの動いていた計算機が不調になったケースで、RTT やプロセスに関するデータを収集した後の修復作業にかかった時間は 38 秒であった。設定は 1 秒程度で終了し、生成されたシェルスクリプトの実行に 37 秒かかっている。データ収集のかかる時間は SSH クライアントのコネクションが確立できず、タイムアウトするまでの時間が大きな影響を与える。これは OS やクライアントの実装に依存するので、今後対処策を考える必要がある。

現状では、本システムは集中管理型のシステムであるため、システムを動かす計算機が不調になると管理が出来なくなるという問題がある。

表 2 Titech Grid で実装したシステムを動かした場合の所要時間

台数	所要時間	ping の実行	NWS の実行
3 台	40 秒	21 秒	19 秒
6 台	69 秒	39 秒	30 秒
10 台	128 秒	76 秒	52 秒

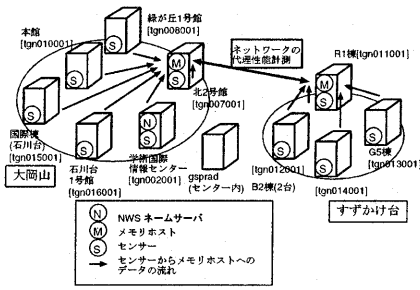

図 3 Titech Grid 上の 10 台の計算機について実行した結果

6. まとめと今後の課題

本研究ではモニタリングシステムの自律的構成システムの設計を提案し、提案した機能の一部として NWS の初期設定と運用開始以降の障害復旧を行うシステムを実装した。実際に東京工業大学にあるグリッド環境のテスト環境である Titech Grid においてこのシステムを動かし、自動設定と障害修復を行う事を確認した。

今後の課題としては時間の経過に伴って変化するネットワークトポロジーに対応した計算機のグループ

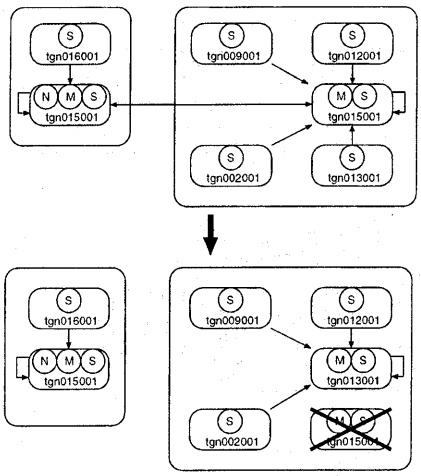

図 4 メモリホストが動いている計算機が不調になった場合の修復例
tgn*** は計算機の名前、N: ネームサーバ、M: メモリホスト、S: センサー、矢印はセンサーからメモリホストへのデータの流れを表す

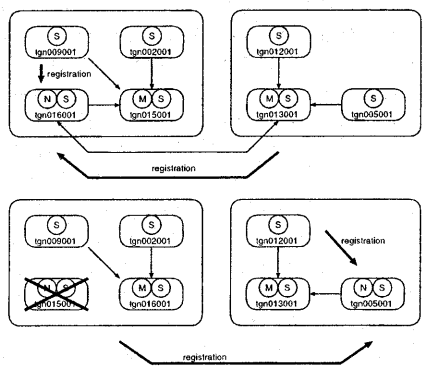

図 5 NWS のネームサーバが動いている計算機が不調になった場合の修復例
registration の矢印はコンポーネントが実行される時にネームサーバに情報が流れる様子を表す

分けのアルゴリズムの調査、モニタリングシステムの管理の分散化、負荷分散を考慮にいれたモニタリングシステムのコンポーネント配置決定、モニタリングシステム及び管理システムのスケーラビリティの実現が挙げられる。

謝　　辞

なお、本研究の一部は文部科学省「経済活性化のための重点技術開発プロジェクト」の一環として実施されている超高速コンピュータ網形成プロジェクト (NAREGI : National Research Grid Initiative) による。

参 考 文 献

1) Pratap Pattnaik, Kattamuri Ekanadham, and joefon jann Autonomic ccomputing and Grid, in Grid Computing (Edited by Fran Berman et al.), WILEY, 2003
2) Rich Wolski and Neil T. Spring and Jim Hayes, *The network weather service: a distributed resource performance for ecasting service for metacomputing*, Future Generation Computer Systems Vol. 15, 1999
3) K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman. *Proceedings of the Tenth IEEE International Symposium* High-Performance Distributed Computing (HPDC-10), IEEE Press, August 2001.
4) Globus project, http://www.globus.org/
5) Hawkeye, http://www.cs.wisc.edu/condor/hawkeye/
6) "DataGrid Information and Monitoring Services Architecture: Design, Requirements and Evaluation Criteria", Technical Report, DataGrid, 2002.
7) Xuehai Zhang, Jeffrey L. Freschl, and Jennifer M. Schopf *A Performance Study of Monitoring and Information Services for Distributed Systems* High-Performance Distributed Computing Vol. 12, 2003