自律運用システム Phantom における高可用化方式の実装

鈴木和宏[†] 松原正純[†] 勝野昭[†]

我々は自律コンピューティングの実現に向け，クラスタシステムの効率的な運用を実現する自律運用システム “Phantom” を提案している．Phantom ではクラスタ内のノードを仮想化することによって，アプリケーションからノードの構成の変化や，故障などを隠蔽することができる．またアプリケーションに割り当てるノード数を自律的に制御することによって，柔軟かつ効率的なリソース管理を実現する．しかし，Phantom は 1 台の管理ノードでクラスタシステム全体を管理して，自律運用のための処理を行っているため，管理ノード自身に障害が発生した場合にはシステムの運用を継続することができなくなる．そこで本稿ではこの問題を解決するために，運用中のノードを管理ノードに切り替えることによって，余分な資源を用意することなく可用性を向上させる高可用化方式を提案する．本方式を Phantom に実装し，IA ブレードサーバ上で実験した結果，ノード故障が発生したときにも，システムを停止することなく処理を継続できることを確認した．

Implementation of High Availability Mechanism for Autonomous Management System Phantom

Kazuhiro Suzuki,[†] Masazumi Matsubara[†] and Akira Katsuno[†]

We are proposing autonomous management system “Phantom” that achieves efficient operation of the cluster system for the achievement of autonomous computing. The change of the node composition and the node failure can be concealed from the application by the virtualization of the node. Moreover, a flexible, efficient resource management is achieved by controlling the number of nodes allocated in the application autonomous. However, when the trouble occurs in the management node because Phantom manages the entire cluster system by one management node, and processes it for autonomous operation, the operation of the system cannot be continued. This paper proposes and describes the high availability mechanism for Phantom. According to this mechanism, we can reduce, so called, “single point of failure” problem. We have implemented this mechanism on IA blade servers, and it has been confirmed to be able to continue processing without stopping the system when the node failure occurred. Experimental results indicate that Phantom with this mechanism can perform correctly, and shows the usability of this mechanism.

1. はじめに

インターネットを利用した *e*-ビジネスでは，24 時間 365 日サービスを提供し続けることが成功のための重要なカギとなる．しかし現実問題として，1 台のマシンが故障や過負荷により停止しただけで，顧客へのサービスが全面的にストップしてしまうことがある．そうなると，莫大な損害を引き起こすことになる．

こうした事態に備えるためにシステムの MTBF（平均故障間隔）を改善し，稼働率を向上させる，高可用化の要求が高まりつつある．可用性を向上させ，システム停止時間を最小限にすることで，被る損害や危険を最小限に抑えることができる．

一般的にシステムの可用性を向上させるには，そのシステムを構成する部品を冗長化することが重要である．そのような高可用化システムとして，クラスタシステムが利用されるケースが多い．クラスタシステムにおいては，2 台またはそれ以上のノードを使用して冗長化し，1 台を現用系として動作させ，残りを待機系とすることで，何らかの原因で現用系が動作不能になった場合に待機系がその処理を引き継ぐことができる．このようなクラスタシステムはフェイルオーバ型クラスタと呼ばれている．フェイルオーバ型クラスタには PRIMECLUSTER [1)]，ClusterPerfect [2)]，CLUSTERPRO [3)] などの製品がある．

基幹業務のデータベースサーバや，アプリケーションサーバ，ファイルサーバなどでは可用性を向上させ

† 富士通株式会社
FUJITSU LIMITED

た高可用クラスタシステムが導入されつつある．また最近では，インターネット上のファイアウォールやメールサーバなどの障害が致命的となるため，このような分野にも高可用クラスタシステムが求められている．

フェイルオーバ型クラスタでは待機系のノードは現用系に障害が発生するまでは稼働せず，余分な資源を用意しなければならない．つまり，たとえば 2 台のサーバでフェイルオーバ型のクラスタシステムを構築したとしても，1 台分しか稼働していないことになる．

ところで現在，我々は自律機能によるクラスタシステムの効率的な運用を実現するために，自律運用システム “Phantom” を開発している[4]．Phantom では，1 台の管理ノード上で動作するプロセスがクラスタシステム全体を管理しており，管理ノード以外のノードに障害が発生したとしても，障害が発生したノードを切り放すことでシステムの処理を継続できる可用性を備えている．しかしながら，管理ノード自身に障害が発生してプロセスが停止した場合にはシステムの運用を継続することができなくなる．これはいわゆる “Single Point of Failure” といわれている問題で，システムの構成要素を冗長化することで可用性を高めることが求められる．

そこで本稿ではこの問題を解決するためにフェイルオーバ型クラスタによる管理ノードの高可用化方式を提案する．本方式によってサービスを運用中のすべてのノードが管理ノードとなることができるため，余分な資源であるアイドル状態の待機系ノードを用意することなく，可用性を向上させることが可能となる．

以下，次章で本高可用化方式を実装した自律運用システム Phantom の構成を説明し，続く 3 章で Phantom の基本的な動作，4 章で提案する高可用化方式について詳述する．最後に本方式の評価を行う．

2. 自律運用システム “Phantom”

2.1 構成の概要

Phantom ではクラスタ内のノードを仮想化し，すべての通信を仮想化された IP アドレスによって行う．これにより，実際の計算機ノードとアプリケーションが操作する仮想的なノードとを切り離し，アプリケーションからノードの構成の変化や故障などを隠蔽することができる．またアプリケーションに割り当てるノード数を自律的に制御することによって，柔軟かつ効率的なリソース管理が実現される．

図 1 に Phantom の構成を示す．図中の各項目は以下のとおりである．

- 管理ノード
  クラスタの外からの仮想ノード宛のリクエストを受け取るノードで，後述の VNIC と組み合わせて，ゲートウェイ機能を提供する．
- 仮想ノード
  管理ノード上に生成された仮想的なノードで，ユーザアプリケーションに見せる仮想的な IP アドレス（VIP: Virtual IP）を提供する．VIP は Linux の IP エイリアス機構によって提供される．
- 実ノード
  物理的な計算機ノードで，各ノードには実際の IP アドレス（RIP: Real IP）が割り当てられている．管理ノードも実ノードの 1 つである．
- VNIC（Virtual Network Interface Card）
  IP 変換テーブルに従って，仮想ノード宛のパケットを実ノード宛に書き換えるカーネルモジュール．
- ノードコーディネータ
  仮想ノードと実ノードのマッピングを管理するデーモンプロセス．全実ノードの VNIC の IP 変換テーブルのコンシステンシを保つ．また各アプリケーションへのノード割当てを担当する．ノードコーディネータは管理ノード上で動作する．
- ND（ノードデーモン）
  ノードコーディネータからの指示に従って VNIC を設定するデーモンプロセス．
- IP 変換テーブル
  VIP から RIP を検索するための変換テーブル．

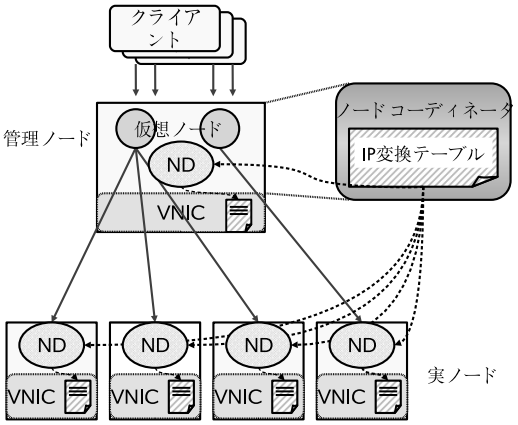

図 1 Phantom の構成
Fig. 1 Overview of autonomous management system: Phantom.

Phantom では，ユーザアプリケーションがクラスタ内の実ノードを直接操作することを禁止している．これはユーザアプリケーションには仮想ノードだけを提供し，実ノードを隠蔽することで実現される．仮想ノードは適切な実ノードにマッピングされ，ユーザア

プリケーションは実ノード上で実行される．マッピングは IP アドレスとポート番号の組によって管理する．仮想ノードへのアクセスは物理ノードへのアクセスに変換され，ユーザアプリケーションにはあたかも仮想ノード上で動作しているように見せている．

Phantom の特長を以下に示す．

- 仮想ノードと実ノードのマッピングを変更することができるため，ユーザは実ノードの構成や台数を知ることなく，アプリケーションを動作させることができる．
- 仮想ノードに実ノードをマッピングする場合には，1 つの仮想ノードに対して複数の実ノードをマッピングすることができる．つまり仮想ノードへのリクエストを複数の実ノードに分散することによってロードバランシング型のクラスタシステムを構成することができる．
- 複数の仮想ノードをユーザに対して提供することによって，複数ノードのクラスタシステムとして見せることができる．そのためユーザは Phantom 上で HPC 型のアプリケーションを動作させることが可能となる．

以下，各構成部について詳細を説明する．

2.2 ノードコーディネータ

ノードコーディネータは管理ノード上で動作するプロセスであり，IP 変換テーブルを管理している．IP 変換テーブルは以下のように VIP と port 番号の組に，1 つまたは複数の RIP を登録するためのテーブルである．

VIP，port	RIP 1	RIP 2	RIP 3	···

ノードコーディネータは IP 変換テーブルのマスタテーブルを操作できる唯一のプロセスである．アプリケーションを実行する場合に，ユーザはノードコーディネータに対して実ノードの割当てを要求する．ノードコーディネータは IP 変換テーブルのエントリを変更し，すべての実ノードに対して IP 変換テーブルのコピーを配布する．これによって IP 変換テーブルのコンシステンシを保つことができる．

IP 変換テーブルが更新されるたびにテーブル全体を実ノードに送ると，ネットワーク帯域を圧迫して本来のサービスの性能を低下させてしまうことが懸念される．そこで IP 変換テーブルに変化があったときに，その差分だけを送る．これによって Phantom が管理するクラスタシステムが大きくなって，IP 変換テーブルのサイズが増加したときにも対応できるようになる．

また，ノードコーディネータは実ノードの状態を監視して，仮想ノードが割り当てられた実ノードの負荷が増大すると，負荷を分散させるために新たな実ノードを割り当てる機能を持っている．反対に負荷が減少した場合には，割り当てていた実ノードを解放する．

2.3 ノードデーモン

各実ノードに常駐し，ノードコーディネータによって配布された IP 変換テーブルのコピーを受け取る．ノードデーモンは受け取った IP 変換テーブルを VNIC に伝達する．

また，ノードデーモンは自らのノードがサービスに割り当てられた場合に，VNIC によって転送されてきた VIP 宛のパケットを受け取るためのトンネルデバイスを設定する．自ノードがサービスから削除された場合は，トンネルデバイスの削除処理を行う．

2.4 VNIC

VNIC は管理ノードを含むすべての実ノードに備えられている．仮想ノードと実ノードとのマッピング情報である IP 変換テーブルによって，アプリケーションからの仮想ノード宛のアクセスが実ノード宛のアクセスに変換される．VNIC によって仮想ノード間の通信は実ノード間のインターコネクトを利用することが可能となる．

VNIC は大きく分けて 2 つの処理からなる．1 つはクラスタ外部からのパケットをクラスタ内部のノードに転送する処理である．これは WWW サーバなどの前段に設置されるロードバランサと同様の働きをする．他方はクラスタ内部のノードからのパケットをクラスタ内のノードに転送する処理である．これによってクラスタ内のノード間通信を行うことができる．

図 2 に VNIC のそれぞれの動作を示す．図 2 (a) はクラスタ外部のノードが送出したパケットをクラスタ内のノードに転送するところを表している．クラスタ外部からの VIP 宛のパケットはその VIP を提供する管理ノードが受け取る．パケットを受け取ったノードは，パケットの宛先アドレスとポート番号をキーとしてテーブルを検索する．検索にヒットするとエントリに登録されている RIP に対してパケットを送出する．ヒットしなければ通常のパケットとして受理する．1 つの VIP に対して複数の RIP を登録することによってロードバランサとして機能させることができる．

図 2 (b) はクラスタ内部のノード間通信を表している．クラスタ内の実ノードはユーザアプリケーションが動作している．ユーザアプリケーションがノード間通信を行うときには VIP によってノードを識別する．VNIC はこの VIP が割り当てられている実ノードに

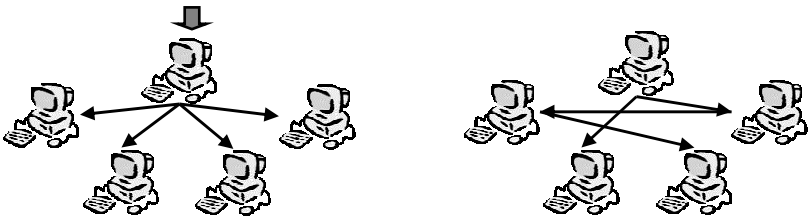

図 2 VNIC の動作
Fig. 2 Behaviors of virtual network interface card.

パケットを転送する．ノード内で生成された VIP 宛のパケットは，図 2 (a) の場合と同様に VIP とポート番号をキーとしてテーブルを検索し，パケットの宛先 RIP を得ることができる．この場合にも複数の RIP を登録することによってノード間通信にもロードバランシングを適用することができる．つまり，各ノードは独立してロードバランサとしての機能を提供することができる．

VNIC の実装の詳細については文献 4) を参照されたい．

3. Phantom の基本動作

3.1 サービスの登録

Phantom ではオペレータによってサービスが初期化される．このとき，ノードコーディネータに対してサービスの登録およびノードの割当て，サービスの起動リクエストが送られる．

サービスの登録の処理フローを**図 3** に示す．登録リクエストを受け取ったノードコーディネータは，サービスに VIP を割り当てて，サービスリストとしてメモリ上に蓄える．サービスの登録にはサービス記述ファイルと呼ぶ XML 文書を指定する．**図 4** にサービス記述ファイルの例を示す．

この例では，3 行目と 4 行目に，“web” という名前のサービスが TCP の “80” 番ポートを使用していることが記述されている．また 5 行目はサービスを起動/停止するためのスクリプトで，“/etc/init.d” ディレクトリ内のスクリプトと同様に引数に，“start” や “stop” を指定して起動される．さらに 6 行目はリソースの競合が発生した場合の優先順位を決定するための優先度である．

さらに 7 行目からの svc: SLA は自律制御の動作を指定するための記述で，データ種別とその閾値を条件式の形で表したものである．例ではノード数と CPU 使用率がデータ種別として示されている．それぞれの条件式はレベル付けされており，レベルが低い方から順に評価していく．評価した結果が “偽” となった場合は，その直前のレベルをサービスレベルとする．複数のアプリケーションが登録されている場合には，すべてのアプリケーションで “真” となるサービスレベル

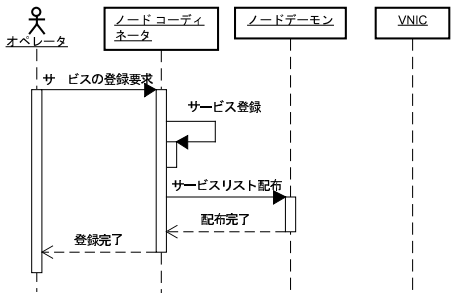

図 3 サービスの登録
Fig. 3 Sequence diagram of service registration.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <svc:config>
3   <svc:name>web</svc:name>
4   <svc:socket proto="TCP" port="80"/>
5   <svc:script>/etc/init.d/httpd</ptm:script>
6   <svc:priority>8</svc:priority>
7   <svc:SLA>
8    <svc:cond level="1">
9              NODE    ==    1   </svc:cond>
10   <svc:cond level="2">
11             CPULOAD &lt; 95 % </svc:cond>
12   <svc:cond level="3">
13             CPULOAD &lt; 90 % </svc:cond>
14   <svc:cond level="4">
15             CPULOAD &lt; 85 % </svc:cond>
16  </svc:SLA>
17 </svc:config>
```

図 4 サービス記述ファイル
Fig. 4 Example code of the service description file.

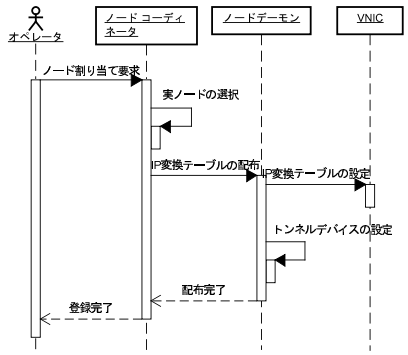

図 5 ノード割当て
Fig. 5 Sequence diagram of node allocation.

が決定される．ノードコーディネータはサービスレベルが最大となるように，サービスに割り当てる実ノード数を制御する．

3.2 ノードの割当て

図 5 にノード割当ての処理フローを示す．オペレータは，ノードコーディネータに対して実ノードの割当てを要求する．割当て要求はサービス名と割り当ててほしい実ノード数で指定される．ノードコーディネータは割当て可能な実ノードのリストから要求された数だけの実ノードを取り出して，IP 変換テーブルに記録する．IP 変換テーブルはすべての実ノード上の VNIC の IP 変換テーブルを更新するためにコピーが配布され，サービスリストと同様にメモリ上に蓄積される．

3.3 ノードスケジューリング

ノードコーディネータはサービスの起動要求を受け取ると割り当てられた実ノード上でサービスを起動してサービス提供状態に入る．

サービス提供状態に入ると，ノードコーディネータはすべての実ノードと仮想ノードの状態を監視して，与えられた SLA の条件式に従ってノードの割当てを行う．これをノードスケジューリング機能と呼ぶ．ノードの負荷が変化した場合には，ノードスケジューリング機能によって実ノード数の割当てを自律的に変更してシステム全体で最適な状態を保つように動作する．WWW サーバのようなスケールアウト型アプリケーションではノードスケジューリング機能を利用することができる．

ノードスケジューリング機能は IP 変換テーブルに登録された RIP の数を変更することで実現される．RIP を増加させるときには，ノード割当て可能な実ノードを選択して IP 変換テーブルに割り当てる RIP を追加する．このとき，すでにサービスが動作している VIP の実ノードを増やす場合には，そのノード上でサービスを起動する．反対に実ノードを削除する場合には，そのノード上のサービスを停止した後で IP 変換テーブルに登録されている RIP を削除する．

4. 高可用化方式

ノードコーディネータは自律機能を制御している唯一のプロセスであるため，ノード故障などで停止した場合には運用を継続することができなくなるという問題がある．本章ではこれを解決するために実装した高可用化方式について説明する．

4.1 提 案 方 式

フェイルオーバ型クラスタの構成方式は様々な方式が提案されており，以下のように分類することができる．

(1) 片方向スタンバイ（アクティブ/スタンバイ構成）
同じ構成のシステムを 2 系統用意しておき，片方を現用系として動作させて他方は同じ動作を行いながら待機状態にしておく．待機系は現用系と同じ状態を保っておき，現用系に障害が発生したときに待機系に処理が引き継がれる．

(2) 双方向スタンバイ（アクティブ/アクティブ構成）
待機系で他の処理を実行することで，お互いが他方の待機系となる．障害が発生したときには，自らのノードで動作している処理に加えて，障害が発生したノードの処理を引き継ぐ．

(1) は高速にフェイルオーバを行うことができるが，現用系に障害が発生しなければ待機系は稼働せず，余分な資源を用意しなくてはならない．それに対して (2) では現用系と待機系を効率的に利用することができるが，障害が発生した場合に他方のノードが過負荷になってしまうことが考えられる．

そこで，本高可用化方式では (2) を応用して，別のサービスを提供していた実ノードをノードコーディネータが動作する管理ノードに切り替える方式を提案する．本方式では新たに管理ノードとなったノードで提供していたサービスを停止して，他の実ノードにサービスを移動させること（サービスマイグレーション）で過負荷な状態となるのを回避している．これによって効率化と高可用化を両立させることが可能となる．

4.2 本方式の構成

管理ノードに障害が発生したときには，ノードコーディネータが保持しているデータを引き継がなければならない．引き継ぐべき情報はクラスタを構成するの

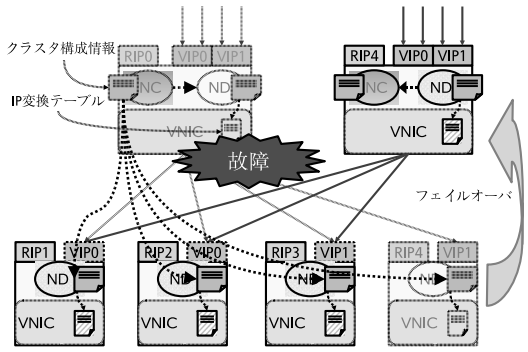

図 6 高可用化方式の構成
Fig. 6 Structure of the high availability mechanism.

表 1 クラスタ構成情報のサイズ
Table 1 Size of cluster composition information.

クラスタ構成情報	サイズ
サービスリスト	360 バイト × サービス数
タスクリスト	128 バイト × タスク数

表 2 更新リクエストと配布タイミング
Table 2 Update request and distribution timing.

更新リクエスト	配布タイミング
SERVICE_REGISTER	サービスの登録
SERVICE_DELETE	サービスの削除
NODE_ALLOC	サービスへの実ノードの割当て
NODE_FREE	サービスからの実ノードの解放
SERVICE_START	サービスの起動
SERVICE_STOP	サービスの停止

に必要な情報であり，以下の項目からなる．

- サービスリスト
  オペレータによって登録されたサービスのリストで，サービス名，起動スクリプト，SLA の条件式などで構成されている．
- タスクリスト
  現在起動しているサービスのリストで，現在のサービスレベル，負荷情報などから構成されている．
- IP 変換テーブル
  割当て済みの実ノードと仮想ノードの組を保存したリスト．

Phantom では IP 変換テーブルをすべての実ノードに配布する機能を持っている．そこで，この機能を拡張することによって，IP 変換テーブル以外のクラスタ構成情報もすべての実ノードに配布するような構成とした．**図 6** に本方式の構成を示す．図にあるようにノードコーディネータが保持しているクラスタ構成情報はすべての実ノードのノードデーモンにコピーされる．これはどの実ノードでも管理ノードとなりうることを意味しており，管理ノードに障害が発生してノードコーディネータが停止した場合には，いずれかの実ノード上で新たにノードコーディネータを起動するだけでよい．

ノードコーディネータが起動されたノードは，ノードデーモンから受け取ったクラスタ構成情報からクラスタ構成を再構築して，新たな管理ノードとなる．

本方式によってサービスを運用中のすべてのノードが管理ノードとなることができるため，余分な資源を用意することなく，可用性を向上させることが可能となる．

ノードコーディネータによって配布される構成情報は各実ノード上で動作するノードデーモンが受け取ってメモリ上に保持する．

本方式によって各ノードデーモンは，IP 変換テーブル以外のサービスリストおよびタスクリストを新たに保持する必要があるために，従来よりも多くのメモリを消費することになる．そこで今回の実装で増加するメモリ量を試算した．各エントリのサイズを**表 1** に示す．

たとえば，10 ノードのクラスタシステムにおいて 3 種類のサービスを提供する場合を考えると，新たに増加するメモリ量は次のように計算される．

$$360\,\mathrm{Byte} \times 3 + 128\,\mathrm{Byte} \times 10 \simeq 2.3\,\mathrm{KByte}$$

すなわち，本方式を採用することによって新たに増加するメモリ使用量は，ノードデーモンあたり約 2.3 KB であり，非常に小さいことが分かる．さらにこれらをノードコーディネータから転送するためのコストも低く抑えられているということができる．

4.3 クラスタ構成情報の更新

すべての実ノードでコンシステンシを保つために，クラスタ構成情報が更新されたときに各実ノードに配布する．更新された内容は，その種類を示す更新リクエストとともに送られる．具体的な更新リクエストと配布タイミングの一覧を**表 2** に示す．

構成情報の配布方式にはブロードキャストによる方式と，各ノードデーモンに対して TCP コネクションを張って point-to-point に行う方式の 2 通りが考えられる．

前者は各ノードデーモンがブロードキャストを受け取って VNIC 内の IP 変換テーブルを更新する処理を並列に行うことができるが，すべての実ノードがノードコーディネータと同じセグメント内に存在していなければならない．また確実にデータを受け取ったことを保証しなければならないため処理が複雑になる．

一方，後者は多少のオーバヘッドはあるが，異なる

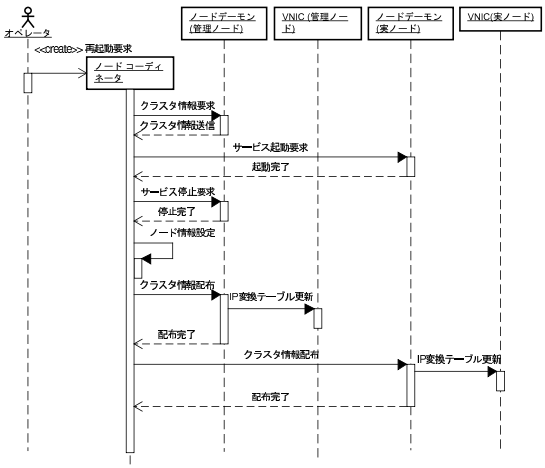

図 7 クラスタ構成の再構築
Fig. 7 Sequence diagram of restructuring of cluster composition.

セグメントに存在する実ノードにも確実に IP 変換テーブルのコピーを送ることができるため，今回の実装では後者を採用している．

リクエストを受け取ったノードデーモンは，リクエストの内容を解析して自らが保持している構成情報に変更を反映する．

IP 変換テーブルの情報は，カーネルモジュールである VNIC に通知される．VNIC への通知には Linux カーネルの Netfilter 機構[5] が提供する以下のインタフェースを用いる．これによって，`setsockopt()` 関数や `getsockopt()` 関数が呼ばれたときに呼び出されるコールバック関数を設定することができ．コールバック関数内でノードデーモンからのデータを受け取って IP 変換テーブルを設定する．

```
int nf_register_sockopt(struct nf_sockopt_ops*);
```

ここで `nf_sockopt_ops` 構造体はコールバック関数を登録するためのものである．

4.4 クラスタ構成の再構築

図 7 に本方式のクラスタ構成の再構築の処理フローを示す．何らかの原因でノードコーディネータが停止した場合には，他の実ノード上で新たにノードコーディネータを起動させる．起動されたノードコーディネータは自ノード（ローカルノード）のノードデーモンに対して構成情報を問い合わせる．

ノードデーモンは構成情報の問合せに答えるための専用のポートを開いており，ノードコーディネータからの問合せを待っている．ノードコーディネータがこのポートに接続して構成情報を問い合わせると，保持している構成情報を，サービスリスト，IP 変換テーブル，タスクリストの順に送信する．ノードコーディネータは受け取った構成情報からクラスタ構成を再構築する．

再構築処理が完了すると，ローカルノードで動作していたサービスを停止して，IP 変換テーブルからローカルノードの RIP を削除する．このとき，ローカルノードが再び割り当てられないようにローカルノードの設定を変更する．

実ノードを管理ノードに切り替えるために，サービスを提供している実ノード数が減少する．これによってシステム全体の性能は一時的に低下するものの，サービスの提供を継続することが可能となる．低下した性能は，新しいノードコーディネータの自律機能によって最適化されることが期待される．

サービスを提供しているノードがローカルノードだけであった場合には，ローカルノード上のサービスを停止する前に，他の実ノード（リモートノード）を割り当ててサービスの起動要求を送る．これによってサービスをマイグレートすることができ，サービスを提供できない状態を排除することができる．

サービスを停止する場合に，毎回サービスマイグレーションを行うとノードコーディネータの再起動時間が延びてしまうことが懸念される．そこで同一のサービスを提供している実ノード数によってサービスマイグレーションを行うかどうかの判定を行っている．

ノードコーディネータはローカルノード上のサービスの停止完了通知を受け取ると，トンネルデバイスの削除と，外部からのパケットを受け取るための VIP の作成（IP エイリアス）などの管理ノードの設定を行う．

管理ノードとしての設定が終了すると，障害ノードを除いたすべての実ノードに新しいクラスタ情報を送出する．これが完了するとサービスを提供することが可能な状態となる．

4.5 障害検出

本来の高可用化機構では処理の継続に問題をきたす障害を検出して，フェイルオーバを実行する．しかしながら本稿ではノードコーディネータのフェイルオーバに焦点を当てているため，今回の実装では人手によって障害の発生を判断し，ノードコーディネータのフェイルオーバをコマンドラインから指示を与えるようにしている．

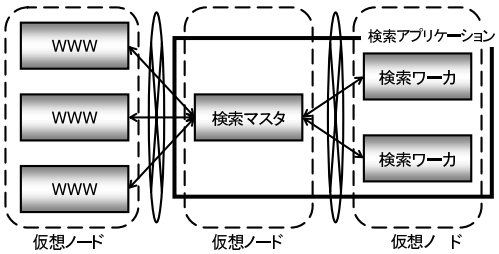

図 8 検索アプリケーションの構成
Fig. 8 Structure of full-text search application.

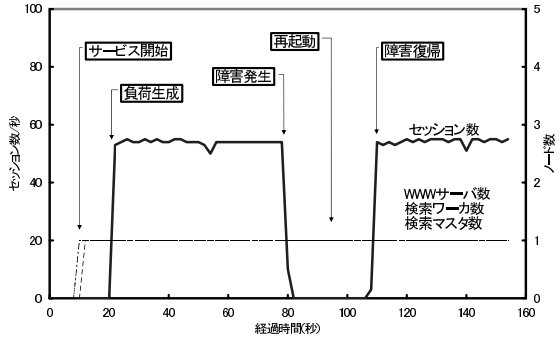

図 9 実験結果 A（低負荷時）
Fig. 9 Experimental result of scenario A.

5. 実　　験

5.1 実験環境

実験にはブレード型 IA サーバである富士通 PRIMERGY BX300 の 20 ノード上に本高可用化方式を実装した．このうちの 1 つのノードを管理ノードとして，ノードコーディネータを動作させた．

実験にはアクセラテクノロジ社製の検索アプリケーション eAccelaBizSearch[6] を使用した．これは WWW サーバに対して検索リクエストを送ると，あらかじめ用意されたテキストデータベースを検索して，その結果を返す検索エンジンである．この検索アプリケーションは**図 8** に示すように検索マスタ/検索ワーカ・プロセスから構成されている．WWW サーバに届いた検索リクエストは検索マスタに投げられ，マスタプロセスは実際に検索処理を行う検索ワーカに処理を依頼する．処理が終了すると結果を検索マスタに返す．検索マスタが整形した結果は WWW サーバ経由でクライアントに返される．

上記の BX300 とは別のノード上に，擬似的な検索リクエストを発生させる検索クライアント群を用意して，1 秒間に応答したセッション数と各サービスに割り当てられている実ノード数を計測した．

5.2 実験シナリオ

今回の実験で行った 2 通りの実験シナリオを示す．

実験 A 1 ノードの検索クライアントから約 55 セッション/秒の一定の負荷をかけた状態で，管理ノードにノード故障を発生させた後でフェイルオーバの操作を行う．フェイルオーバによって，サービスの提供を継続できることを確認する．

実験 B 検索クライアントから約 200 セッション/秒の負荷をかけて実験を開始する．安定状態に入った後で，管理ノードにノード故障を発生させ，フェイルオーバの操作を行う．その後，負荷を約 400 セッション/秒に増加させて，新たに起動したノードコーディネータが負荷変動に反応して自律運用を継続できることを確認する．

前述したように，障害の検出は人手で行い，ノードコーディネータのフェイルオーバをコマンドラインから手動で行った．ノード故障は管理ノード上のノードコーディネータを停止し，VIP を削除することで行った．またフェイルオーバ操作として新しいノードコーディネータを実ノード上で起動して VIP の設定を行った．

このときノードコーディネータを再起動するノードは，サービスを提供している実ノードであればどこでもよく，今回の実験では無作為に抽出した．

5.3 実験結果

実験 A の結果を**図 9** に示す．横軸が実験開始からの経過時間，縦軸が応答したセッション数および WWW サーバ，検索マスタ/ワーカのそれぞれに割り当てられた実ノード数である．

グラフから約 10 秒後にサービスが開始され，各サーバの初期値である 1 ノードずつが割り当てられたことが分かる．さらに約 20 秒後に検索クライアントから負荷を発生させている．負荷が発生すると，応答したセッション数はほぼ一定の値を示しており，このときの負荷が非常に低く初期状態の構成で十分にリクエストに応答できていることが分かる．

約 80 秒後にノード障害が発生していることが分かる．これによって VIP あてのリクエストが処理されずに，セッション数が 0 となっている．約 90 秒後にフェイルオーバの操作を行った．これによって構成情報が再構築されて，サービスが提供可能な状態に移行する．このとき，構成情報からサービスを提供しているサーバの台数を検出して，これが 1 台だけだったときにはサービスのマイグレートを行った後にサービスを停止する．この処理に掛かるコストは，ノード台数の検出およびノードの割当て，サービスの起動，サービスの停止の処理時間の合計となる．

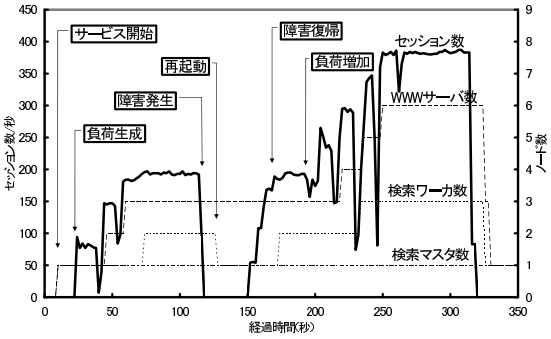

図 10 実験結果 B（負荷変動時）
Fig. 10 Experimental result of scenario B.

ノード台数の検出は構成情報の大きさによるが，今回の実験での構成程度では非常に軽い処理である．さらに実ノードの割当ては，割り当てられるノードの情報を構成情報に追加する処理であるため，構成情報の大きさによらず高速に処理を行うことができる．またサービスの停止/起動処理はサービスの種類によって異なる．たとえば WWW サーバでは起動に約 3 秒，停止に約 1 秒の時間を要した．したがって WWW サーバのサービスマイグレーションは約 5 秒以内で完了することができる．

グラフによると約 110 秒後に検索リクエストに対する応答を開始して障害復帰しており，ノードコーディネータが再起動してから約 20 秒のタイムラグがあることが分かる．これは，クライアント群がキャッシュしている arp テーブルのエントリが更新されるまでの時間である．arp テーブルは IP アドレスから MAC アドレスを検索するためのテーブルである．障害が発生したノードから新しいノードに切り替わるときには，VIP を引き継いでおり，MAC アドレスは引き継いでいない．arp テーブルは一定時間ごとに更新されて新しいエントリに書き換わるため，この時間が現れたものと考えられる．これを解消するためには Gratuitous ARP[7] と呼ばれるパケットをブロードキャストすることによって arp テーブルを更新すればよい．

次に実験 B でのセッション数と各サーバに割り当てられたノード数の時間変化を**図 10** に示す．

グラフは約 10 秒後にサービスが開始され，約 20 秒後に 5 ノードの検索クライアントからの負荷を発生させた状態を示している．負荷を発生させると，初期値である 1 ノードではリクエストを捌ききれず，ノードコーディネータのノードスケジューリング機能によって WWW サーバと検索ワーカに割り当てている実ノード数が増加して，それぞれ 3 ノードと 2 ノードとなったところで安定状態に入ったことが分かる．

ここで，約 115 秒後に管理ノードに障害を発生させ，約 125 秒後に検索ワーカが動作していたノードでノードコーディネータを再起動させた．これは検索ワーカに割り当てられている実ノード数が 2 から 1 に減少していることからも分かる．減少した検索ワーカは新たに起動されたノードコーディネータの自律機能によって，約 170 秒後に元の 2 ノードに増加されて障害前の状態に復帰していることが確認できる．

さらに約 200 秒後に検索クライアントの数を 10 ノードに増加した．新しいノードコーディネータはこの負荷変動に追従して，WWW サーバを 6 ノード，検索ワーカを 3 ノードにまで増加させることで，安定してリクエストに応答できる最適な状態に移行していることが分かる．このことからも再起動されたノードコーディネータの自律機能が正しく機能していることが分かる．

以上により Phantom の管理ノードに障害が発生した場合でも，本高可用化方式によってシステムを止めることなく処理を継続することができることが確認された．

6. 関 連 研 究

LinuxVirtualServer（LVS）[8] は Linux カーネル 2.4.23 から正式採用された，Linux カーネルによるロードバランシングソフトウェアである．LVS はインターネットから届いたパケットを 1 台の LVS で受け取って，後続のサーバノードに転送するものであり，VNIC が持っているノード間通信の仮想化は行っていない．

Ultra Monkey [9] は LVS の高可用化を目的としたプロジェクトである．Ultra Monkey では複数の LVS によるロードバランサ用意して，これらをアクティブ/アクティブ構成で運用することによって，可用性を向上させて Single Point of Failure を解決する方式を提案している[10]．これはすべてのロードバランサに同一の IP アドレスと MAC アドレスを割り当て，1 つの接続には 1 つのロードバランサだけが動作するようにフィルタリングすることで実現している．同様な手法に文献 11)，12) などがあげられる．また文献 13) では TCP ルータを冗長化し，ラウンドロビン DNS によってこれを決定し，負荷状況に応じて他のノードにリクエストをフォワーディングする手法を提案している．これに対して本研究では通常の実ノードを管理ノードに切り替えることで高可用化を実現しており，すべてのノードを均一に扱うことでノードの用途をあらかじめ決めておく必要がなく，より柔軟な構成をとることができる．

他の方式として，文献 14)，15) では現用系と待機系を用意して，待機系でパケットをモニタリングすることによって TCP の状態を現用系と同期させるアクティブ/スタンバイ構成を採用している．それに対し，本研究では待機系のノードを用意することなく，可用性を向上させる方式を提案している．

7. ま と め

本稿では我々が開発を進めている自律運用システム Phantom の可用性を向上させるための高可用化方式を実装した．本方式によって，システムで唯一であるノードコーディネータに障害が発生した場合に運用を継続することができなくなるという問題を解決した．IA ブレードサーバ上での実験から，ノードコーディネータが他の実ノード上に移動することによって，サービスの提供を継続できることを確認した．

さらに Phantom が管理しているクラスタ構成情報が再構築されて，負荷変動によるノード数を動的に変化させる自律運用が継続されていることを確認した．

今回は手動で行っている障害検出機構を開発して，管理ノードのフェイルオーバを自動化する必要があり，今後の課題と考える．また，引き継ぐ構成情報の論理検証を行うことによって，データの正当性を保障することも重要な課題である．

謝辞 本研究は新エネルギー・産業技術総合開発機構基盤技術研究促進事業「高信頼・低消費電力サーバの研究開発」によって行われた．

参 考 文 献

1) PRIMECLUSTER. http://primeserver.fujitsu.com/primepower/products/soft/primecluster/
2) ClusterPerfect. http://www.toshiba.co.jp/
3) CLUSTERPRO. http://www.nec.co.jp/
4) 鈴木和宏，松原正純，勝野 昭，クラスタ計算機システムにおけるノード仮想化方式，並列/分散/協調処理に関するワークショップ（SWoPP2003 松江），CPSY2003-6〜18, pp.67–72 (2003).
5) Netfilter Core Team. http://www.netfilter.org
6) eAccelaBizSearch. http://www.accelatech.com
7) Stevens, W.R.: *TCP/IP Illustrated, Volume 1: The Protocols*, Addison-Wesley, Reading, Massachusetts (1994).
8) LinuxVirtualServer プロジェクト. http://www.linuxvirtualserver.org
9) Ultra Monkey プロジェクト. http://ultramonkey.org
10) Horman, S.: LVS: Active-Active Servers and Connection Synchronisation, *Proc. Linux Conference Australia* (Jan. 2004).
11) Leite, F.O.: Load-Balancing HA Clusters with No Single Point of Failure, *Proc. 9th International Linux System Technology Conference* (Sept. 2002).
12) 瀧ヵ平健，Sedukhin, S.：マルチキャストによるクラスタ Web サーバの構築，インターネットカンファレンス 2001 (2001).
13) Bestavros, A., Crovella, M., Liu, J. and Martin, D.: Distributed Packet Rewriting and its Application to Scalable Server Aarchitectures, *Proc. 6th International Conference on Network Protocols* (Oct. 1998).
14) Aghdaie, N. and Tamir, Y.: Clinet-Transparent Fault-Tolerant Web Service, *Proc. 20th IEEE International Performance, Computing and Communications Conference*, pp.209–216 (Apr. 1997).
15) Law, K.L.E., Nandy, B. and Chapman, A.: A Scalable and Distributed WWW Proxy System, *Proc. International Conference on Multimedia Computing and System* (June 1997).

(平成 17 年 1 月 24 日受付)
(平成 17 年 4 月 25 日採録)

鈴木 和宏

昭和 41 年生．平成 2 年横浜国立大学工学部情報工学科卒業．平成 4 年東京大学大学院工学系研究科情報工学修士課程修了．同年（株）富士通研究所入社．クラスタシステム，自律コンピューティングの研究に従事．

松原 正純（正会員）

昭和 48 年生．平成 8 年筑波大学第三学群情報学類卒業．平成 13 年同大学大学院工学研究科博士後期課程修了．同年（株）富士通研究所入社．博士（工学）．クラスタシステム，自律コンピューティングの研究に従事．

勝野 昭

昭和 60 年大阪大学工学部卒業．同年（株）富士通研究所入社．コンピュータアーキテクチャ，自律コンピューティングの研究開発に従事．