ミッション・クリティカル・システムにおけるオートノミック・コンピューティング機能の構築

小宮 聖則

Implementation of Autonomic Computing Functions for Mission Critical Systems

Kiyonori Komiya

ミッション・クリティカル・システムは, 高可用性についての要件が厳しく, サーバーのフェイル・オーバー機能や, 過負荷をシステムに与えないための流量制御といった機能が必要である. オートノミック・コンピューティングの定義においては, それらは自己修復と自己最適化にあたり, 基本的な機能としては, 既にハードウェア, ソフトウェア製品で提供されている. 本論文は, ミッション・クリティカル・システムにおける, それら製品機能の補完によるオートノミック・システム構築事例に基づき, ミッション・クリティカル・システムに求められる, オートノミック機能の要件について明確化する. その結果として, ノードの監視方法や流量制御の考慮点などについて得られた知見を述べる.

Generally, the mission critical system needs to have fail-over functions for its servers and the flow control function to prevent overload system operation in order to meet high-availability requirements. In the definition of the autonomic computing system, these are defined as 'self-healing' and 'self-optimization'. The basic functions are already implemented in hardware and software products. This paper describes case references of autonomic computing systems in which these capabilities are built on top of existing product functions. The implementation experience suggests that the importance of autonomic computing for mission critical systems. Based on the results of experiences, some know-how such as how to monitor nodes, how to design the flow control function, etc. are explained.

Key Words & Phrases：オートノミック・コンピューティング, 高可用性, 自己修復, 自己最適化, 流量制御, WebSphere, Tivoli
Autonomic computing, high availability, self-healing, self-optimizing, flow control, WebSphere, Tivoli

1. はじめに

IBMの提唱するe-ビジネス・オンデマンド・オペレーティング環境（ODOE）において, インフラストラクチャー・マネージメント・ソリューション［1］を構築する重要な技術としてオートノミック・コンピューティングが位置付けられている. それを実現するための製品として, ソフトウェアの側面ではTivoli®製品群やalphaWorks®のオートノミック・コンピューティング（AC）ツールキット［2］などによる具体的なソリューションが発表され, 利用可能になってきている. 今後, ますます巨大化・複雑化するお客様のIT環境への対応として, オートノミック・システムは重要な位置を占めてくるであろう.

まずここで, 本論文で取り上げるオートノミック・コンピューティングの領域について明確化する. IBMの提唱するオートノミック・コンピューティングは次の4つの構成要素により定義されている［1］.

- **自己構成**
  環境の変化にダイナミックに適応する.
- **自己修復**
  システムの中断を防止するために,不具合を発見・診断し修正する.
- **自己最適化**
  IT資源を最大限に利用するために, 資源を調整したり,負荷分散したりする.
- **自己防御**

提出日：2004年08月31日　再提出日：2005年1月12日

予見や検知, 識別を行い危険に対処する. また, 攻撃に対する防御を行う.

本論文では, ミッション・クリティカル・システムにおいて必須である自己修復と自己最適化の機能について, 2つのインターネット・システムでの構築事例を中心に述べる. それらは, 実際の発生障害の対応策や, 厳しい業務要件を元に構築したフェイル・オーバー機能(障害のあるノードを切り離し, 正常なノードでサービスを継続すること. 自己修復機能)および流量制御機能(自己最適化機能)である. これら機能の構築において考慮すべき点について, 具体例を示すと共に, オートノミック機能における要件を明確化する.

以下, 2章においては, (個人顧客向け)インターネット・バンキング・システムでの自己修復と自己最適化機能の構築事例を示し, 3章においては, 同じく銀行の法人向けインターネット・システムで構築した, 複数のオートノミック・マネージャーによる協調型システムの事例を示す. 4章では, 本論文で示すオートノミック機能を, IBMのオートノミック製品であるTivoli, ACツールキット, WebSphere®をベースに構築する場合の実現可能性についてまとめ, 最後に, 一連のオートノミック・システム構築により得られた知見を述べる. (オートノミック・マネージャーとは, オートノミック・コンピューティング・アーキテクチャーで定義された制御ループを実装したコンポーネントのことである[3].)

以降説明する事例は, IBM WebSphere Application Server(以降WASと記述)およびWAS Edge Component製品の自己修復と自己最適化機能を補完するものである. 本論文中に示す製品仕様については, WebSphere v5.0以前のものであり, 今後の仕様変更により変わりうる. なお, オートノミック・コンピューティングの詳細については, 参考文献[1,2,3,4]などを参照されたい.

2. インターネット・バンキング・システムの自己修復と自己最適化

本章では, インターネット・バンキング・システムの典型的な構成を示し, その自己修復機能(フェイル・オーバー機能)を強化するために構築したマルチスレッドによる監視機能を説明する. 最後に自己最適化機能(流量制御)の拡張を行った事例を示す.

2.1 典型的なシステム構成

インターネット・バンキング・システムは, 入出金明細照会, 振込, 振替などの銀行取引を, インターネット上でサービスするシステムであり, 今や多くの銀行にとって, ROI増加につながる戦略的基幹システムとなっている.

このようなミッション・クリティカルなWebシステムを構築する場合, 通常, ブラウザーからのHTTPリクエストは, 負荷分散装置(もしくはここで示す例のようなソフトウェア型のサーバー)を経由して, クラスタリング構成されたWebサーバーに処理を振り分ける. これにより, 負荷分散機能(自己最適化機能)を実現すると同時に, 障害の発生したWebサーバーを検知し, 振り分け対象から除外するフェイル・オーバー機能を実現する. さらに一定以上の負荷をシステムに掛けない流量制御機能も実現する必要がある. その典型的なシステム構成を**図1**に示す.

図1は, IBM HTTP Server(以降 IHSと記述)およびWASからなるWebサーバー, アプリケーション(AP)サーバー, DBサーバー, そして勘定系システムへのゲートウェー(GW)・サーバーを, クラスタリングもしくは

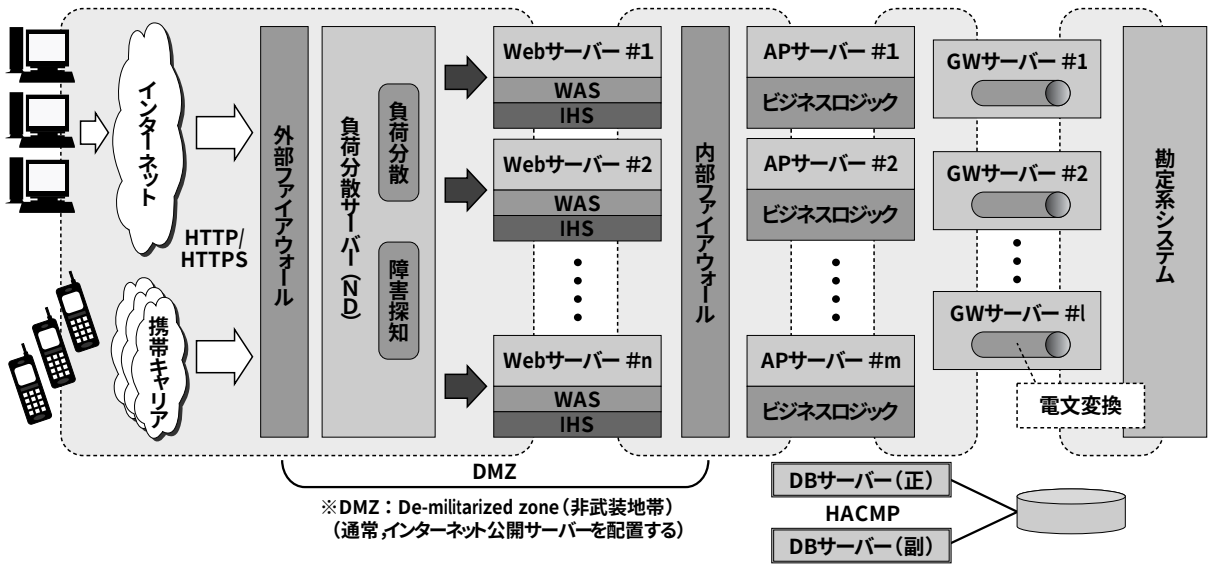

図1. インターネット・バンキング・システム概要

HACMP™（High Availability Cluster Multi-Processing：ノード障害、ネットワークアダプター障害などを検知し、正副切り替えにより自動復旧するシステム）による冗長構成としている. Webサーバーの負荷分散については, IBM WebSphere Edge ComponentのLoad Balancer Network Dispatcher（以降NDと記述）と, WASのセッション・パーシスタント機能（Webコンテナ間でHTTPセッションを引き継ぐためのHTTPセッション・データ保管機能）を用いたクラスタリング構成により実現している. 銀行勘定系の戦略的デリバリー・チャネルであるため, 半分に縮退運用した場合でも, 十分なキャパシティーの機器で構成している.

2.2 自己修復機能に関する課題

図1のシステムの場合, 当初Webサーバー（IHS, WAS）の障害検知はND製品が提供する監視機能（custom advisor）により実現していた. それは, pingとHTTPサーブレット・リクエストに対するレスポンス・タイムにより行われるが[5], 複数のWASが同時に障害になると, 障害ノードへの振り分けの停止が遅延する場合がある. これは, NDのadvisorコンポーネントが, クラスター内の各Webサーバーをシーケンシャルに監視するためである. ネットワーク不通やサーバーの完全なダウンの場合はpingが失敗し, 早いタイミングで障害と判断できるが, WASのサーブレット処理スレッドがデッドロックでフリーズしているような状態ではTCPのポートは開いているため, ping成功の後, HTTPリクエストの送信にも成功し, レスポンスを待つことになる. そこで受信のタイムアウトになるとHTTPリクエストを規定回数まで再送し, リトライする. この間, 数十秒以上経過してしまうため, 同時に他のWASも障害になった場合, 障害ノードへもしばらくの間振りつづけてしまう.

インターネット・バンキング・システムは, 非常にピーク性が高く, クラスター構成の1台のWebサーバーに対して, ピーク時で秒あたり50を優に超えるサーブレット・リクエストが投入される. したがって, HTTPレスポンスのない（遅い）Webサーバーに対しては, NDからの振り分けを早期に停止させ, 正常なサーバーのみに処理を振り分ける必要がある. この制御が遅れると, 多くのユーザーのブラウザーに対して応答ができなくなり, ユーザーからコールセンターに問い合わせが殺到するという事態に発展する. このようなトラブルを未然に防止するため, 複数のサーバー（WAS）が同時に障害になった場合でも, 障害発生のノードに対しては, 早期に確実に振り分けを停止する機能の構築が求められた.

2.3 マルチスレッドによる自己修復機能の構築

この対応として, **図2**に示すヘルス・モニターを構築した. ヘルス・モニターとは, オートノミック・マネージャーの具現化システムの名称として読み進めてほしい.

これは, 複数のWebサーバーを同時に監視し, 障害ノードの検知時には運用監視オペレーターへ障害メッセージを通知すると同時にNDのコマンドを自動で発行し, そのノードへの振り分けを明示的に停止するものである.

当システムは, IBMオートノミック・コンピューティング・ブループリントが発表される前に構築したものだが, そのアーキテクチャーとしては, 以下に示すような, オートノミック・コンピューティング・アーキテクチャーの制御ループを実装した構造となっており, これは, オートノミック・マネージャーの実装例と言える[3]. 監視対象Webサーバー毎に監視（Monitor）スレッドを割り当て, それらが収集するHTTPレスポンス結果（レスポンス・タイムとHTTP処理結果）を, 別の分析（Analyze）スレッドが判断する. そして, 定義されたルー

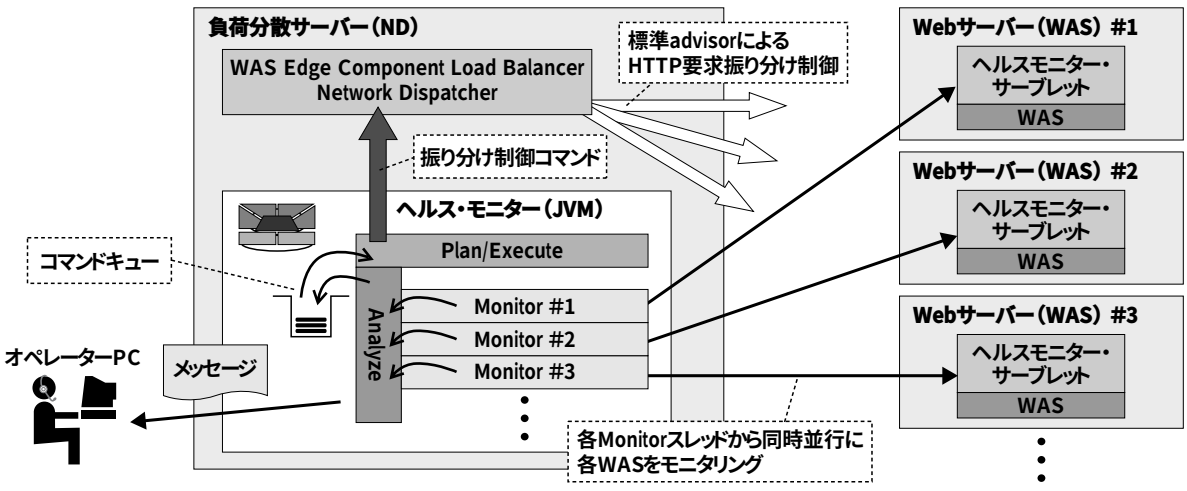

図2. ヘルス・モニター概要

ル(後述のRule#1)に基づき, 障害と判断するノードが存在する場合, 計画・実行(Plan/Execute)スレッドにキューを経由してNDの操作コマンドを通知する. Plan/Executeスレッドは, キューからのコマンドの取得・実行を繰り返す.

Webサーバー(WAS)を障害と判断するルールについては以下のようにし, 容易に調整できるように設定数値は定義ファイルのパラメータとしている. (n, m の数値)

(Rule #1):
過去連続 n 回のHTTP要求が, すべてしきい値m秒以上のレスポンス・タイムであった場合, 振り分けを停止する.

バックエンド・システムの一時的な処理遅延や, インターネット・システム特有のトラフィック・スパイクにより, サーブレット・リクエストが一時的にWASにたまり, レスポンス遅延が発生する場合などがあるため, 上記のnは通常2回以上に設定するべきである. また, 復旧(振り分け再開)についての自動化は, レスポンスを監視し続け, 回復を判断することにより技術的には実現可能であるが, 運用監視員による手動復旧が適している場合もあり, ニーズに応じて実装すべきである.

この仕組みにより, 複数のWASで同時に障害が発生しても, 早いタイミングで検知し, 該当ノードを振り分け対象から外すことが可能となった. このように, クラスタリング構成された監視対象ノードに対するオートノミック・モニタリングは, 各ノードを並行処理にて同時に監視する必要がある. つまり, オートノミック・マネージャーの『監視』コンポーネントは, 対象ノード毎に並行して動作するように実装する必要があるということである. 特に, タイムアウトやリトライが発生しうるセンサーの場合は重要である. (センサーとは, オートノミック・マネージャーに対して, 監視対象エレメントの情報を渡す手段・コンポーネントである[1].)

2.4 自己最適化機能の構築

本節では, バックエンド・システムの処理遅延事象の対策として実施した, ヘルス・モニターへの自己最適化機能の拡張について述べる. これは, ITITO (IBM Tivoli Intelligent ThinkDynamic Orchestrator) やWebSphere XD (eXtended Deployment) [6]で実現するような自動プロビジョニング機能ではないが, システムのキャパシティー超過による全業務サービス停止を防止し, システムを救うための流量制御機能である.

Webシステムにおいて, バックエンド・システムの処理時間が遅延すると, WAS内部のサーブレットの占有時間(処理時間)も遅延し, レスポンス・タイムが悪化し, 最悪の場合はシステムの全面障害になりうることは, 既に多くのWebシステム構築経験者は認識していることであろう. 通常, 一定以上の負荷がかかった場合にバックエンド処理遅延が発生しやすい. このような場合, フロントからの流量制御により, 一定のレスポンス・タイムを維持する方法が考えられる. この流量制御であるが, 負荷分散装置(ソフトウェア型も含む)の標準機能であるHTTPセッション数のしきい値超えにより, Webサーバーに投入するHTTPリクエストを無差別に制限してしまうと, 複数画面の遷移を経て完結する取引に影響が出てしまう. 例えば振込処理の場合, 出金口座選択画面, 入金口座銀行・支店選択画面, 金額入力画面, というように画面遷移して, 一つの銀行取引が完結する. そのため, 取引中のユーザーを考慮する必要がある.

本オートノミック・システムでは, ヘルスモニターによる, レスポンス・タイム悪化検知の際に, ログオン要求のHTTPリクエストのみを制限し, ログオン済のユーザーのHTTPリクエストは受け付ける方針で実現した. 分析(Analyze)機能が判断する‘ログオン休止モード’ON/OFFの判定ルールは, 次のとおりである.

(Rule #2):
過去連続 p 回のHTTPリクエストが, すべてしきい値 q 秒以上のレスポンス・タイムであった場合,‘ログオン休止モード’に設定する.

(Rule #3):
‘ログオン休止モード’であり, かつ過去連続 r 回のレスポンス・タイムが s 秒以下に回復した場合,‘ログオン休止モード’を解除する.

Rule #2のp, q は, Rule#1の n, m よりも小さい値で設定するべきである. つまり振り分け停止に至るほどのレスポンス遅延になる前に, ログオンの休止によりシステム負荷の軽減を試みて, レスポンス・タイムの回復を期待することが可能だからである.

この‘ログオン休止モード’のON/OFF設定については, ヘルス・モニターの実行(Execute)機能から, 該当Webサーバーへの HTTP要求により, 切り替えの専用サーブレットを呼び出す方式としている. また, アプリケーションのログオン用サーブレットで,‘ログオン休止モード’のON/OFFを検査し, ON(ログオン休止中)の場合に現在混雑中の画面を出すように対応している.

この機能により, ピーク時にバックエンド処理遅延が発生する状況でも,‘ログオン休止モード’の

ON/OFFを何度か繰り返すことにより, バックエンドへの負荷を調整しつつ, ログオン済ユーザーの取引を処理して行き, ピークが過ぎ去るまでしのぐことが可能となる.

このように流量制御は, アプリケーションの特性を考慮した上で実装し, 場合によってはアプリケーション側の対応も必要である.

3. 法人向けインターネット・システムの自己修復と自己最適化

本章では, 個人向けに比較して多額の取引が行われ, 高可用性についての要求が極めて高い法人顧客向けインターネット・システムでの事例を示す. これは, オートノミック・マネージャーの協調処理による適用例である.

3.1 システム概要と要件

Webシステムにおいて, 業務ロジックをWebコンテナで動作するJava™で実装する場合, **図3**のようにWebサーバー(IHS)と, アプリケーション・サーバー(WAS)を分離させる構成は一般的である. この例は, インターネットからのHTTPリクエストを, 負荷分散サーバー(ND)経由で2台のWebサーバーに負荷分散し, さらにIHS側のWAS plug-in機能にて4台のアプリケーション・サーバー(WAS)に振り分けている. つまり, IHSが1に対してWAS(Webコンテナ)が2のクラスタリングである. WASのアプリケーションからは, DBサーバーや他システムと連携するエンタープライズ・システムである.

前章の例は, IHSとWASが同一筐体のシステムであるため, NDで動作するオートノミック・マネージャー(ヘルス・モニター)により, WAS内部の稼動状態の監視, つまり業務サービスのレベルでの障害検知が可能であった. しかしながら, このケースはIHS側のWAS plug-inから, 2台のWASに振り分けているため, NDの位置からの監視では, どのWASに振られるか不定であるため, 制御は困難である。

WASのIHS側plug-inモジュールは, Webコンテナに対しての負荷分散およびセッション・パーシスタントを前提としたフェイル・オーバー機能を標準で備えている[7]. WASに対しての疎通確認を行い, 障害の発生しているWebコンテナ(WAS)には振り分けをしないフェイル・オーバー機能である. 疎通による障害検知であるため, Webコンテナ内での業務アプリケーションの不正や, Webコンテナから先のバックエンド・シ

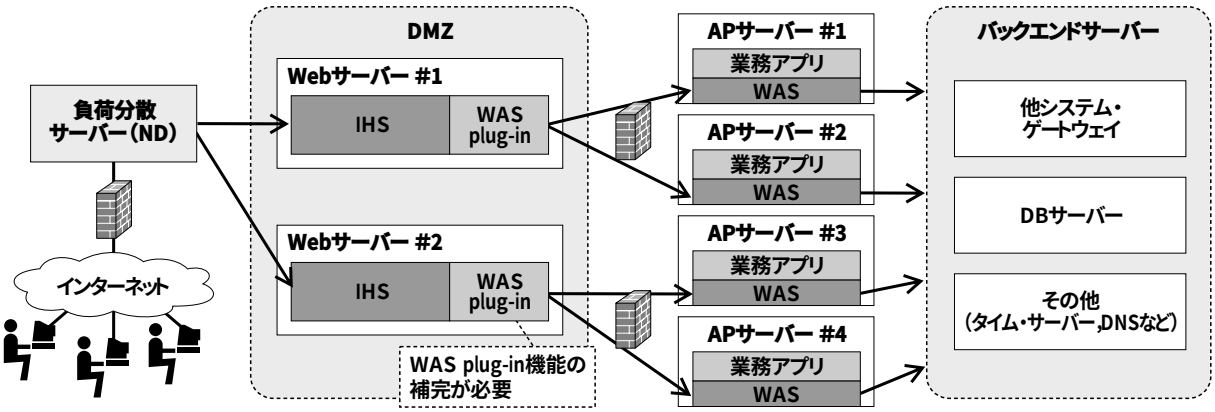

図3. IHS,WAS分離型システム

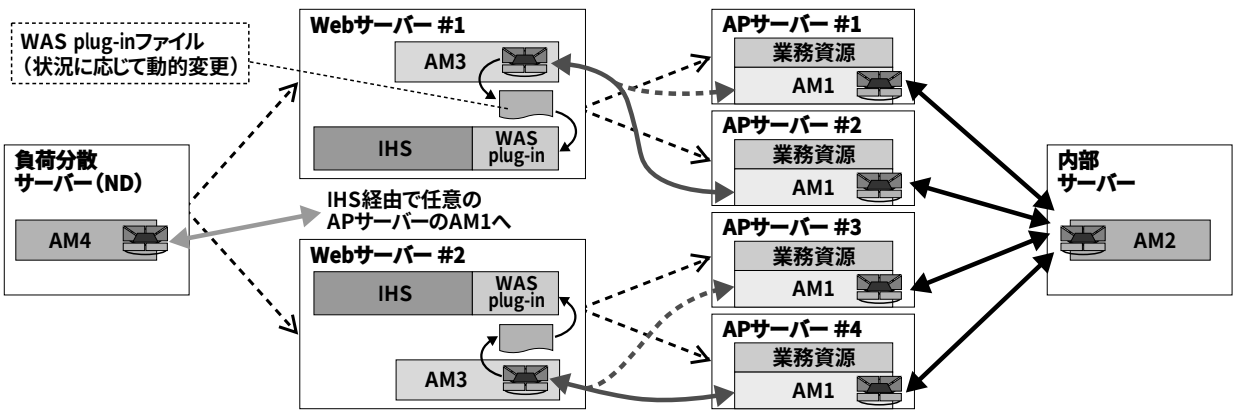

図4. 協調型オートノミック・マネージャーのシステム

ステムのネットワーク障害などの検出はできない（WebSphere v5現在）. この機能を補完するため, 次に示す要件のオートノミック・システムが必要である.

オートノミック機能要件：

各Webコンテナ内のアプリケーションのサービス稼動状況およびバックエンド・システムとの連動状況を監視し, 不具合の発生しているWebコンテナに対しては, IHS側のWAS plug-inの構成を動的に変更することにより振り分けを制御する. さらに, Webサーバー（IHS）が接続するWebコンテナ（2つ）すべてが障害の場合, 当該WebサーバーへのNDからの振り分けを停止する.

3.2 実装例

上記の機能要件を実現するにあたり, **図4**に示す複数のオートノミック・マネージャーの連携処理による方式を考案し, 実装した.

内部の4台のAPサーバー（WAS）に対して, HTTPにより直接アクセスできる内部のサーバーを中心として, 各Webコンテナ内の業務サービス状況を伝播（でんぱ）し, 負荷分散サーバー（ND）からでも全APサーバーの状況に応じた制御が可能な方式となっている. **表1**にそれを実現する4つのオートノミック・マネージャーについてまとめる.

表1. 協調型オートノミック・マネージャーの各機能

オートノミック・マネージャー名／稼動サーバー／（連携先）	機　能
AM1／Webコンテナ／（AM2,3）	Webコンテナ内部のアプリケーション・サービス状態やバックエンド・システムとの接続状態を監視し, Webコンテナ内部のデータ・オブジェクトに格納する. AM2から他Webコンテナの状況が送られてくるのでそれもデータ・オブジェクトに保管する. AM2およびAM3からの問い合わせに対して, データ・オブジェクトに格納した情報を提供する.
AM2／内部サーバー／（AM1）	4つのWebコンテナ（WAS）に対して, 他WebコンテナのAM1の状況を伝えると同時に当該WebコンテナのAM1状況を問い合わせる. つまり, Webコンテナ間のAM1状況伝播を行う.
AM3／Webサーバー／（AM1）	WebコンテナのサーブレットをIHS経由で呼出して, AM1状況（AM2が伝播したもの）を取得する. これには全Webコンテナの状況が含まれている. 自サーバーのIHSの振り分け対象Webコンテナに不具合が発生している場合, そのWebコンテナに振り分けをしない設定でplug-in構成ファイルを更新する. 更新されたplug-in構成ファイルは, WAS plug-inの標準機能により自動的にリロードされ, 振り分け設定が動的に変更される. （plug-inの定義で更新チェック間隔を指定可.） また, WebコンテナへのHTTPリクエストのレスポンス・タイムやAM1が提供するWebコンテナ内部状況により, 前章で提示したような自己最適化機能（ログオンの制御）を実装することも可能である.
AM4／ND／（AM1）	Webサーバー（IHS）の振り分け先Webコンテナがすべて障害の場合, NDのコマンドを発行し, 当該Webサーバーへの振り分けを停止する. 一つでも稼動している場合は, AM3によりplug-inファイルが適切に更新されている（はずである）ため, NDからの振り分けは続行する.

このシステムの一つ一つのオートノミック・マネージャーは, それぞれ独自の責務を果たしながら, 他のオートノミック・マネージャーと連携している. これは, オートノミック・マネージャーのコラボレーション（協調）[4]により成り立つ実装例と言える.

4. IBMオートノミック関連製品による実現

これまでに述べてきたオートノミック機能について, IBMのオートノミック関連製品を利用した実現可能性について、**表2**にまとめる.

文献[9]にあるように, Webサーバーのレスポンス監視とNDの操作によるオートノミック機能はITMTP, ITM, TEC といったTivoli製品で実現可能である. しかしながら, 現バージョンのITMTPは, Webサーバーの監視はシリアル処理される仕様となっている（2005年1月現在）. 今後の機能拡張に期待したい. ACツールキット AME（Autonomic Management Engine）のリソー

表2. IBM製品によるオートノミックの実現

機能	実現製品	説明／備考
Webサーバーレスポンス監視	ITMTP（IBM Tivoli Monitoring for Transaction Processing）	Webサーバーの疎通・レスポンス・タイムのモニタリングが可能.（現バージョンはシリアル処理）
	ITM（IBM Tivoli Monitoring）	リソース・モデル（java, shellでカスタマイズが可）を定義し, 各種性能情報モニタリングが可能. リソース・モデルはインスタンス毎に同時並行稼動.
	AC ツールキットAME（AMEそのものにはレスポンス監視機能はなく, リソースモデルで実現）	
	WebSphere XD	Dynamic Operations機 能の運用ポリシーで, 平均レスポンス・タイムを設定可能.
しきい値監視／オートノミック・エンジン	ITM	リソース・モデル定義に従い, しきい値の監視を行う.
	ACツールキットAME	
	WebSphere XD	Dynamic Operations機 能によるしきい値監視と動的リソース拡張／縮小.
全体ルール処理／イベント通知	TEC（IBM Tivoli Enterprise Console®）	操作員メッセージ通知やコマンド発行を行う.
アプリケーション内部状態監視	WAS v5（JMX, MBeanによる開発）*1	コンテナ内のアプリケーション状態をカスタムMBeanで取得. 情報収集はJMX APIで取得する.
流量制御	WAS Edge Components Load Balancer	ログオン制御による流量制御. CBRの動的ルール変更により実現.
	WebSphere XD	オンデマンド・ルーター機能によりリクエスト・タイプ毎の制御が可能.

注）*1 JMX（Java Management Extensions）, MBean は, J2EE1.4より標準として採用されたシステム管理・監視機能（API）の仕様. WAS v5においては, 既にWAS管理機能で採用済[8].

ス・モデルのインスタンスは, 同時に並行稼動する仕様であるため, 本論文で主張した複数ノードの同時並行監視が可能と考えられる[10].

オートノミック・マネージャーのコラボレーションの例として提示したアプリケーション内部状態監視の機能については, JMXとカスタムMBeanによる実装が, 今後の標準であると考えられる[8]. AMEやITMはJMXインターフェースでのリソース・モデルがサポートされる.

流量制御においては, NDのCBR機能(URLのパターンで振り分けを制御する機能)のルールを動的に変更するコマンドを, TECのイベント処理から発行することで実現できる. つまり,‘ ログオン休止モード ’では, ログオンURLのパターンの場合にSorryサーバーに振るルールを動的に設定する[7, 11].

WebSphere XDは, パフォーマンスとHTTPリクエストの重要度が考慮されたきめ細かい流量制御機能を提供し, さらに, 負荷に応じて動的にアプリケーションをデプロイする自己最適化機能も有する[6].

オートノミック・システムを構築するにあたり, 本論文で示したような, 業務特性やシステム要件に応じた製品補完機能の開発, あるいはカスタマイズは依然必要ではあるが, これらIBMオートノミック関連製品を利用して, より容易に実現できる土台はそろっていると言える.

5. おわりに

本論文では, ミッション・クリティカル・システムでの, 高可用性にフォーカスしたオートノミック・システムに求められる機能についての構築事例を示した. 最後に, 構築を通して得られた次の4点の知見について, 結論としてまとめる.

(1) ノードの監視は並行処理で行う

クラスター構成された複数ノードが同時に障害になった場合でも, フェイル・オーバーを遅延させないため, オートノミック・マネージャーの監視(Monitor)コンポーネントは, ノード毎に同時並行して監視できるように実装すべきである.

(2) 流量制御は業務特性を考慮する

システムへの過剰な負荷により全面停止に至る前に, 流量制御によりシステムを救うことが必要. ただし, 複数画面をまたがる仕掛かり中の処理を優先的に処理するなど, アプリケーションの特性を考慮し, サービス率を極力下げない工夫が必要.

(3) 業務サービスの監視が必要

バックエンド・システムと連携するようなミッション・クリティカル・システムにおけるフェイル・オーバー機能では, ノード・ダウンやプロセス・ダウンのみならず, 業務サービスのレベルでの障害検知が必要である.

(4) オートノミック・マネージャーの協調処理を検討

複雑なシステム構成の場合など, 単一のオートノミック・マネージャーだけでは要求される機能の実現が困難な場合, 複数のオートノミック・マネージャーの協調処理(連携処理)により実現する方式を検討する.

本論文で提示したこれらの知見が, 今後オートノミック・システムを構築する開発者にとって参考になることを願う.

謝辞

本論文を執筆するにあたり, 多くの情報をIBMソフトウェア事業部, IBMシステムズ・エンジニアリング株式会社のWebSphere, Tivoli技術サポート関連の方々より入手した. 関係諸氏に対し, ここに深く感謝の意を示したい.

参考文献

[1] オートノミック・コンピューティングガイドブック, http://www-6.ibm.com/jp/autonomic/pdf/guidebook.pdf, 2004/7/20

[2] Autonomic computing Toolkit http://www-106.ibm.com/developerworks/autonomic/probdet.html#gla?Open&ca=daw-ac, 2004/7/20

[3] オートノミック・コンピューティング・アーキテクチャーに関するブループリント, http://www-6.ibm.com/jp/autonomic/pdf/acbp2_2004_jp.pdf, 2005/01/21

[4] L. Stojanovic, Th. Lumpp, et al.., *The role of ontologies in autonomic computing systems, IBM Systems Journal* vol.43, No 3, 598-616, 2004

[5] IBM WebSphere V5 Edge of Network Patterns, http://www.redbooks.ibm.com/redbooks/pdfs/sg246896.pdf, 2004/08/30

[6] WebSphere Extended Deployment, http://www-6.ibm.com/jp/software/websphere/ft/was/xd/index.html, 2004/08/10

[7] IBM WebSphere V5.1 Performance, Scalability, and High Availability WebSphere Handbook Series, http://www.redbooks.ibm.com/redbooks/pdfs/sg246198.pdf, 2004/08/30

[8] 日本アイ・ビー・エム システムズ・エンジニアリング, WebSphere V 5.0 開発者必携ガイド1 J2EE入門

[9] 稲山 享伸, 宮本 良麿, Tivoliによるオートノミック・コンピューティング機能の実現 (PROVISION Winter 2004 No.40), http://www-6.ibm.com/jp/provision/no40/pdf/40_ppr1.pdf, 2004/08/30

[10] IBM alphaWorks, AME Developer's Guide, p10, http://www-106.ibm.com/developerworks/autonomic/probdet1.html, 2004/8/31
[11] Load Balancer 管理ガイド バージョン5.1(GC88-9356-01), http://www-306.ibm.com/software/webservers/appserv/doc/v51/ec/infocenter-jp/edge/LBguide.htm, 2004/8/31ISBN4-8171-6061, 1998

日本アイ・ビー・エム株式会社
ITアーキテクト
小宮 聖則 Kiyonori Komiya

[プロフィール]
1987年 日本アイ・ビー・エム株式会社入社. 金融機関のお客様を担当するSEとして配属され, 米国IBMでの金融ミドルウェア開発, 帰国後各種C/Sシステム開発プロジェクトを経験. 1999年より某都市銀行様担当として, 分散系, Web系のプロジェクトに携わる. システム基盤, ミドルウェア、アプリケーションを中心としたアーキテクチャー構築に従事.

KOMIYA@jp.ibm.com