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

― 製品機能によるサーバーの自動アロケーションの実装 ―

Realizing Autonomic Computing with Tivoli®:

Automatic Server Allocation Using Product Functions

オートノミック―グリッド・コンピューティング機能を利用したソリューションとして「サーバー資源の自動アロケーション」がある。筆者は検証環境において、既存製品と通常のカスタマイズや開発により、「システム状況に応じたサーバー資源の自動アロケーション」を実装した。製品機能による実装であり、お客様の環境でもすぐに利用できる。当論文は、実際の経験を踏まえて、既存の製品機能を利用してサーバーの自動アロケーションを実装する際の仕組みや設計、考慮点について論ずる。

Automatic server-resource allocation is a solution using autonomic grid computing functions. The author achieved automatic allocation of server resources in accordance with system conditions through existing products and ordinary customization and development in a test environment. Implementation involved product functions and immediate use is possible even in customers' environments. In this paper I examine on the basis of actual experience the configurations, designs and points that need to be considered when implementing automatic server-resource allocation using existing product functions.

日本アイ・ビー・エム システムズ・エンジニアリング株式会社
システム・センター
主任ITスペシャリスト
Advisory IT Specialist
System Center, IBM Japan Systems Engineering Co., Ltd.

稲山 享伸 Kiyonobu Inayama

【プロフィール】
1995年、日本アイ・ビー・エム入社。同年、日本アイ・ビー・エム システムズ・エンジニアリング出向。以来、Tivoliなどのシステム/ネットワーク管理製品およびプロジェクトの技術支援を担当。主に金融や製造業などのお客様のシステム管理プロジェクトを経験し、その経験をベースにオートノミック・コンピューティング関連の技術支援を実施中。

日本アイ・ビー・エム システムズ・エンジニアリング株式会社
システム・センター
コンサルティング ITスペシャリスト
Consulting IT Specialist
System Center, IBM Japan Systems Engineering Co., Ltd.

宮本 良麿 Yoshimaro Miyamoto

【プロフィール】
1979年、日本アイ・ビー・エム入社。流通・サービス業界のお客様担当SEとして汎用機、MVSシステムを主に担当。1988年システム・センターに異動し、汎用機や分散系におけるシステム/ネットワーク管理製品、およびプロジェクトを担当。現在は、オートノミック・コンピューティング技術も踏まえ、運用管理全般のプロジェクトを支援。

1. はじめに

2002年10月に、IBMはe-ビジネス・オンデマンドを提唱した。e-ビジネス・オンデマンドに求められるIT（Information Technology：情報技術）環境として、ODOE（On Demand Operating Environment：オンデマンド・オペレーティング環境）があり、ODOEの特性の一つとしてオートノミックがある。今日では、オートノミック・コンピューティングという用語も定着してきている[参考文献1]。

今年の7月にはオンデマンド・オペレーティング環境ソリューションとして、3個のカテゴリーに整理されたオファリングが発表された（図1）。「統合化」「自動化」および「仮想化」のソリューションである。具体的なオファリングには、サーバーのプロビジョニングやIT資源の動的な割り当てを行うものがある[参考文献1]。

このうちの仮想化ソリューションとして、「IBM Webサーバー最適化オファリング」がある。これは、オートノミック・コンピューティングとグリッド・コンピューティングの技術を利用し、システムの稼働状況を監視し、SLA（Service Level Agreement）に基づき、サーバー資源を動的に割り当てようというものである。サーバーの負荷にピーク性があり、ピーク時以外におけるサーバー資源を有効利用しながら、ピークへの対応を動的かつ自動的に行いたいというお客様の要望に沿うソリューションである。このソリューションの原型は、オートノミック・コンピューティングのデモとしてHotRodなどの名称で公開されたものである[参考文献2]。

筆者は、「システム状況に応じたサーバー資源の動的かつ自動的なアロケーションは、既存の製品群の組み合わせでもある程度実現できる」と考えた。実際に、HotRodなどの特徴や狙いを踏まえ、現時点で利用できる製品群と一部のカスタマイズや開発により、検証環境においてソリューションを構築した。以下にソリューションの設計や機能の概要を紹介し、実環

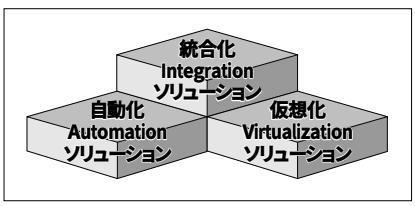

図1. ODOEのソリューション

境で構築する際の考慮点や考察を論ずる。

2. サーバーの自動アロケーション

まず、「システム状況に応じたサーバー資源の動的かつ自動的なアロケーション」の概念的な仕組みを説明する。ソリューションの目的は、システムの負荷に変動がある場合に負荷に応じてサーバー資源を動的に追加／削除し、サーバー資源をより効率的に利用することである。重要度の高いWebアプリケーションに優先的にサーバー資源を割り当てながら、負荷が軽い場合にはサーバー資源を別のアプリケーションで活用する。

オートノミック・コントロール・ループに則して考えると以下のようになる(図2)[参考文献3]。

- **監視：** システムの稼働状況を把握する。監視項目には、CPU(Central Processing Unit：中央演算処理装置)使用率やメモリー使用量などのシステム負荷に関する項目、スループットやトランザクション量などのシステム処理量に関する項目、応答時間などのユーザーが体感できる項目などがある。
- **分析：** 現状のシステムの稼働状況を分析し、対応が必要か否かを判断する。対応の必要性を判断する際の手法としては、あらかじめ設定したしきい値との比較、さらにしきい値超えの回数や傾向を考慮するオカレンス／ホールの手法、現状のデータからの予測結果に基づく将来のしきい値超えのチェックなどがある。また、サーバー資源の追加／削除を実施する場合には、追加／削除すべきサーバーの候補を洗い出し、対象サーバーを決定する。そのためには、システム全体の構成情報や個々のサーバーの稼働状況も把握する必要がある。例えば、何らかの障害を起こしているサーバーは追加できない。このように、サーバーの追加／削除が必要であると判断されると、さらに必要な情報を取得し、これらの追加情報やシステムの稼働情報に基づき、追加／削除すべきサーバーが決定される。
- **計画：** 実際にサーバー資源の追加／削除をシステムに対して指示する。具体的な指示の内容や方法は、環境により異なる。例えば、EdgeサーバーやWAS(WebSphere® Application Server)サーバーに対するコマンドの実行指示となる。
- **実行：** 指示に従い、サーバー資源の追加／削除を行う。その結果、追加されたサーバー資源でトランザクションの処理が行われるようになったり、削除されたサーバー資源でほかのアプリケーションが稼働できるようになったりする。
- **知識：** 現状分析やそれに伴う判断などのロジックは知識に当たる。必要に応じた追加情報の取得や追加情報に基づくサーバー資源の決定、さらに構成情報や性能情報なども知識といえる。

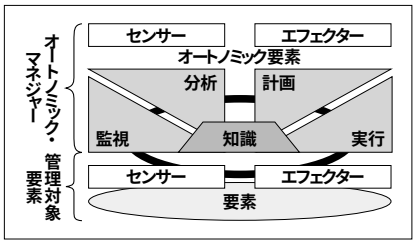

図2. オートノミック・コントロール・ループ(MAPE)

また、オートノミック・コンピューティングには進化の5段階(基礎・管理・予測・適応・自律)があり、今回のソリューションはレベル4(適応)を目指すものである。今回のソリューションにおけるオートノミック・コントロール・ループと製品機能やカスタマイズ、開発機能との対応は、4章に記述する。

3. サーバーの自動アロケーションの実施例

「システム状況に応じたサーバー資源の動的かつ自動的なアロケーション」を行うソリューションとしては、デモ・システムとしてHotRodなどがある。ここで、簡単にその特徴を紹介する。

3.1. HotRodによる実現方法

HotRodの主な特徴を簡単に整理する[参考文献2]。監視項目としては、トランザクション量に注目する。分析としては現状値からの予測を行い、近い将来の予測値に基づいた分析を行う。分析結果に基づいた対応としては、サーバー資源の動的な割り当てを実行する。ソリューションの位置付けとしては、デモであり、実際のお客様環境への提供については、個別の対応が必要となる。

3.2. 今回のソリューション

— Tivoli製品(ITMTPなど)を利用した実現方法 —

システムの状況に応じて、動的かつ自動的にサーバー資源を割り当てる仕組みとしては、Tivoli®製品のITMTP、ITM、TECなどを利用した方式も可能である。詳細は、4章で記述するが、主な特徴は以下の通りである。

- Tivoliの標準製品機能と通常のカスタマイズ・開発により実装されている。
- システム状況の監視項目としてシミュレーターによる応答時間を利用している。
- 監視項目のしきい値監視に関連して、オカレンス／ホールの考え方を取り入れて、きめの細かい判断ができる。
- 追加すべきサーバーを決定する際に、その追加候補のサー

バーの稼働状況を考慮している。

- お客様の要件に合わせて、判断ロジックなどを追加することが比較的容易である。

4. Tivoliによるサーバーの自動アロケーション

既存製品を組み合わせたサーバーの自動アロケーションの検証システムを図3のように構築した。以下に考慮点や考察を交えながら構成や処理の概要および検証結果を解説する。

使用した製品を表1にまとめる［参考文献4］。括弧内の製品略称は図3の表記と一致させており以後の説明でも使用する。

4.1. 構成概要

当ソリューションはWebシステムと監視システムの二つから成る。Webシステムはロードバランサーと複数のWebサーバーによって構成される。ロードバランサーは負荷分散を行うとともに、Webシステムへのサーバーの追加／削除を制御する。「未使用」のサーバー群は予備サーバーであり、高負荷時にのみシステムへ追加される。複雑性を排除するため、バックエンドのデータベース・サーバーなどは除外した。一方の監視システムはシミュレーターを用いてWebシステムの応答時間を計測し、その結果に応じてWebシステムの構成を自動調整する。例えば、監視システムは応答時間の劣化を検知した場合、予備のサーバーを自動追加してWebシステムの処理能力を向上させ、サービス・レベルの維持に努める。その際に追加可能なサーバーのCPU使用率を性能情報として蓄積しておき、最も負荷の少ないサーバーを追加する。反対に応答時間が正常化したときにはサーバーの削除を行う。

監視システムは異機種混在環境にも対応しており、Webシステムが IBM製品でなくとも軽微な変更で利用することができる。ただし、ロードバランサーがコマンドなどによる外部からの制御機能を持っていることが前提となる。

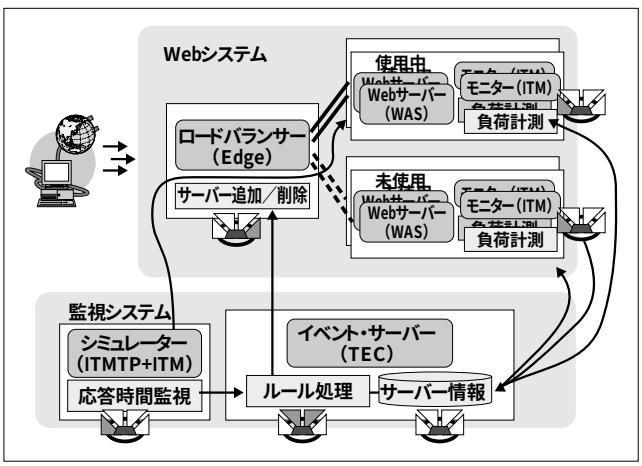

図3. 検証システム構成図

表1. 使用製品一覧

システム	製品名	機能概要
Web システム	WebSphere Edge Server Load Balancer 5.0(Edge)	ロードバランサー
	WebSphere Application Server 5.0(WAS)	Webアプリケーション・サーバー
監視 システム	Tivoli Framework 3.7.1	Tivoli稼働環境 リモート・コマンド実行環境
	IBM Tivoli Monitoring for Transaction Performance 5.1(ITMTP)	応答時間計測機能
	IBM Tivoli Monitoring 5.1.1(ITM)	リソース・モデルによるサーバー監視機能 リソース・モデルの開発環境
	IBM Tivoli Enterprise Console® 3.8(TEC)	統合イベント管理コンソール ルールによるイベント自動処理

4.2. 処理概要

図3における自動アロケーションの処理群を紹介する。基本的な処理の流れは、「応答時間監視」→「ルール処理」→「サーバー追加／削除」という順である。「サーバー情報」は各処理に共通の知識となるサーバーの構成情報や性能情報を集中管理している。Webサーバーにおける「負荷計測」は「サーバー情報」の性能情報を蓄積する処理である。これらの処理は複数のサーバー間で連携しながらオートノミック・コントロール・ループを構成する。応答時間監視が「監視」、ルール処理が「分析と計画」、サーバー追加／削除が「実行」、サーバー情報が「知識」に当たる。負荷計測は知識であるサーバー情報を補完する。それぞれの処理の実装について説明する。

(1) 応答時間監視

シミュレーターがロードバランサー経由でトランザクションを実行し、応答時間をしきい値監視する。製品の記録機能によりトランザクションを容易に作成・編集することが可能であり、実際の利用者の操作に近づけることができる。監視結果として通知されるイベントには応答時間劣化イベントと応答時間正常化イベントの二つがあり、これらは次工程のルール処理へ送られる。この処理を実現するため、ITMTPとITMを組み合わせて利用する方法を考案した。ITMTPが応答時間を計測し、ITMのリソース・モデルがその結果をしきい値監視する。ITMリソース・モデルのしきい値監視機能を利用した背景は、応答時間監視の有用な知見となるので以下に詳しく説明する。

一般的に利用者全体の平均的な応答時間は高負荷時に劣化するが、シミュレーターによる計測結果は平均と比較してばらつくことがある。よって1回のしきい値超えだけで対応を実施するのではなく、何回かの監視結果を総合的に評価して対応することが望ましい。特に自動アロケーションでは、むやみにシステムを変更させないためにもこうした考慮が必要になる。この課題を解決する方法として、ITMリソース・モデルのオカレン

ス／ホールの活用が有効と考えた。オカレンス／ホールは、簡単にいうとしきい値超えの回数を考慮する機能である。オカレンスはしきい値違反状態の上限値、ホールはしきい値正常状態の上限値を表している。連続したしきい値超えの回数がオカレンスに指定された数値に達するとイベントが生成される。頻繁に変動する指標であるということを念頭に、複数回の監視結果を総合し「応答時間が劣化した状態が一定の時間連続したらサーバーを追加する」という方法を実現している。

(2) ルール処理

Webシステムの現状を分析し、必要な対応を決定する。状況分析には複数のサーバーの情報を必要とするため、中央の管理サーバーが情報を集中管理して分析する方法が適切と考えた。この機能を実現するにはルールによる統合的なイベント処理機能を持つTECの利用が有効である。当ソリューションでは、TECのルールと開発プログラムを組み合わせることにより、応答時間の変化に応じてサーバー追加／削除などの具体的な対策を決定する方法を実装した。ルール処理としては、応答時間劣化イベント用と応答時間正常化イベント用の二つを開発したが、ここでは図4を用いて応答時間劣化イベント用ルール処理を説明する。

最初にルール処理はロードバランサーと連携して現状把握を行う。「イベント情報の取得」で応答時間が劣化しているWebシステムの識別情報（代表ホスト名）を取得して、「使用中サーバーの把握」でロードバランサーから割り振り対象のサーバー一覧を取得する。次に「追加サーバーの決定」では、前工程で得られた割り振り対象のサーバー一覧と「サーバー情報」の最大構成サーバー一覧を比較し、その差分から追加可能なサーバーを選択する。その際CPU使用率を選択基準として利用する。追加するサーバーが存在した場合、ロードバランサーへ「サーバー追加コマンド実行」を指示する。ルール処理の結果はTECイベントとして操作員へ通知される。

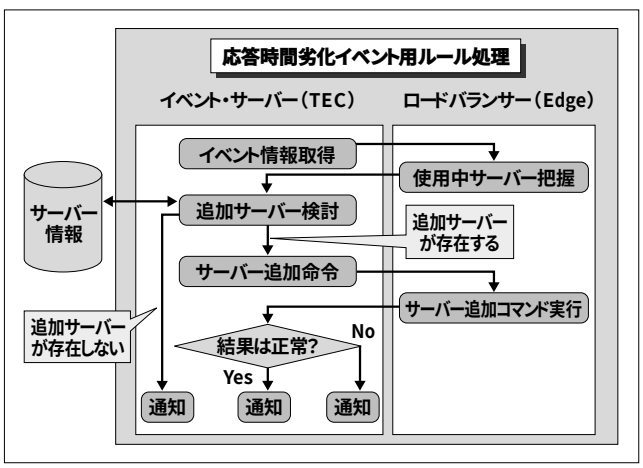

図4. 応答時間劣化イベント用ルール処理

(3) サーバー追加／削除

ルール処理の指示により、ロードバランサーでサーバー追加／削除を実行する。この機能を実現するためにEdgeサーバーの管理コマンド“ dscontrol ”を利用したシェル・スクリプトを開発した。サーバー削除の際に既存接続の強制終了を避けるため、“ manager quiesce ”というコマンドを用いた。このコマンドはサーバーの重みを0にすることで、既存接続を残したまま、新規接続の割り振りを停止する。サーバー削除を実施する際にはこのような既存利用者への配慮が重要である。ルール処理とシェル・スクリプトを連携させるためにはリモート・コマンド実行環境が必要になる。SSH（Secure Shell）やTivoli Frameworkなど幾つかの方法が考えられるが、セキュリティー・ポリシーなども考慮して最適な手段を選択すればよい。

(4) サーバー情報・負荷計測

各処理に共通の知識として、サーバーの構成情報や性能情報を集中管理する。構成情報はWebシステムの最大構成サーバー一覧や最小構成サーバー一覧を管理している。構成情報にはWebシステムの識別情報（代表ホスト名）を付加して、複数のWebシステムを集中監視できるよう考慮した。一方の性能情報は、負荷計測の処理で集められたサーバーのCPU使用率を管理している。性能情報はルール処理の追加サーバー選択の判断基準として利用される。負荷計測にはITMのCPU監視リソース・モデルを利用した。さらに、性能情報を拡張してサーバー単体の応答時間などの指標を組み合わせて利用することも考えられる。ITMやITMTPの標準機能で多くの監視項目が得られるので、負荷計測部分の機能拡張は比較的容易に実装できる。

4.3. 検証結果

負荷ツールを用いて自動アロケーションの効果を検証した。表2はテスト・ケースと検証結果のサマリーと監視条件である。図5はサーバー台数と応答時間の相関グラフである。

検証結果について以下に解説する。なお、“ AC（Autonomic Computing）機能 ”という用語を今回実装した自動アロケーション機能の総称として使用する。

ケース1は通常時の状態である。Webシステムはロードバランサーと Webサーバー1台の最小構成で機能する。シミュレーターが計測する応答時間は500m秒前後である。ケース2ではAC機能を停止した状態で負荷ツールを用いて高負荷にした。最小構成での運用を継続し、平均応答時間は9秒程度に劣化する。ケース3では高負荷の状態を維持し、AC機能を使用可能にしてサーバーの自動追加を可能にした。図4でオカレンスの数だけ応答時間の劣化が発生した後、サーバー追加が実施さ

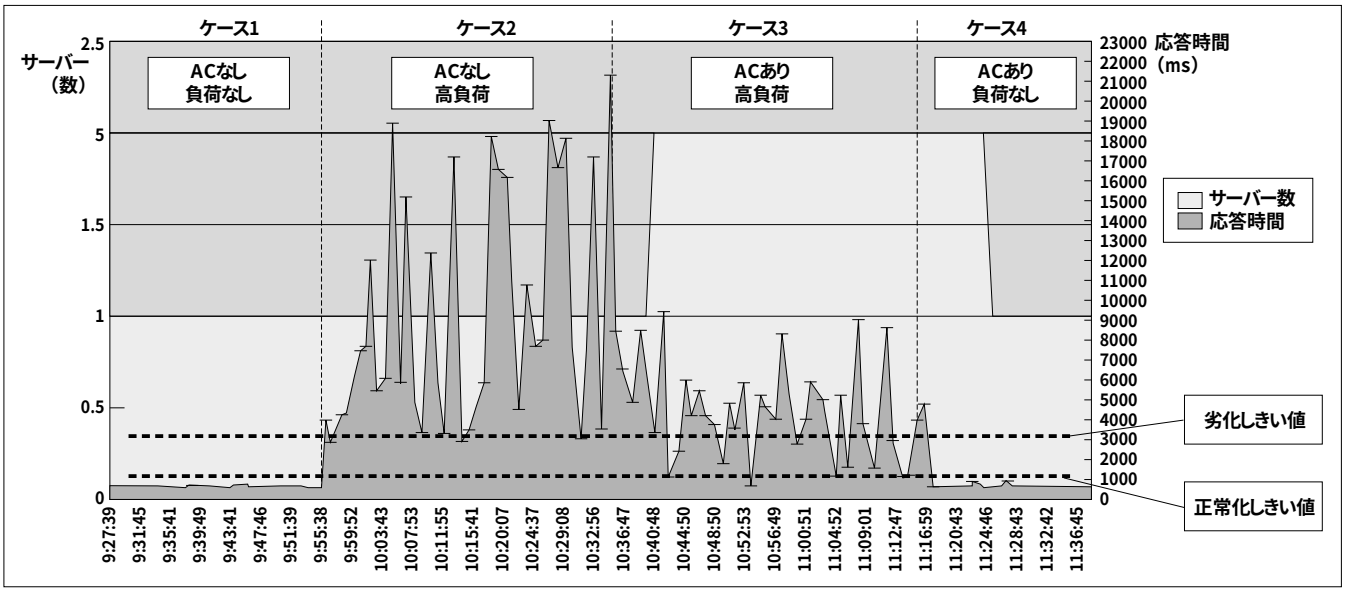

図5. サーバー台数と応答時間の相関グラフ

表2. テスト結果サマリー

ケース	内容	平均応答時間(ms)
ケース1	通常負荷環境における計測結果(ACなし)	470.41
ケース2	高負荷環境における計測結果(ACなし)	9094.63
ケース3	高負荷環境における計測結果(ACあり)	4224.44
ケース4	通常負荷環境における計測結果(ACあり)	466.27

しきい値名	条件	オカレンス	ホール
応答時間劣化しきい値	3秒以上	5	1
応答時間正常化しきい値	1秒以下	5	1

れている。このケースでは平均応答時間は4秒程度まで改善されており、自動アロケーションの効果が見られる。まれに応答時間が正常化しきい値を下回ることがあるが、オカレンスに達しないので無視され、サーバー削除は実施されない。このことからも応答時間監視にオカレンス／ホールを取り入れた効果が実証されている。ケース4ではAC機能を使用可能にした状態で負荷を通常状態に戻した。応答時間が正常化しきい値をオカレンスの数だけ下回った後、サーバー削除処理が実行されサーバー台数が1台の最小構成に戻る。検証システムという限定された環境だが、ソリューションの効果は確認できたと考える。

5. 考察

当ソリューションに関する考察を以下に述べる。

(1) 監視項目(負荷指標)

2章で述べたが、Webシステムの負荷の指標には、システム負荷状況・システム処理量・応答時間がある。システム負荷状況は実際のユーザー・サービスに直結するとはいえない。また、スループットやトランザクション数などの処理量の指標は必ずしもシステム負荷を表すとはいえない。トランザクションには重いもの／軽いものがあり、処理量が一定でも重いトランザクションが増えればシステム負荷は高まる。これに対して、応答時間はユーザーが体感できる指標であり、把握に技術的な困難さがあっても、ほかの指標よりも適切な監視項目であるといえる。

(2) 応答時間の把握方法

今回のソリューションは、応答時間をシミュレーターにより計測する。ユーザーや処理内容を登録し、ダミー・トランザクションを発生させ、その応答時間を計測する。これは実トランザクションではなく、ユーザーが直面する応答時間とは異なる可能性があり、正当性を疑問視する声もある。Edgeサーバーなどのロードバランサーを利用したWebシステムでは、コネクション数や応答時間などによりバックエンド・サーバーの負荷を考慮し、最適なサーバーへ処理が割り振られる。シミュレーションの応答時間も、この割り振りの結果であり、全体の負荷を表す指標になると考える。また、図5の検証結果でもWebシステムの負荷集中とシミュレーションの応答時間に相関が見られ、シミュレーターの有用性は立証されている。

一方、現実には実トランザクションの応答時間の適切な把握は困難な場合が多い。ARM API(Application Response Measurement - Application Program Interface)の実装が普及し、実トランザクションの応答時間が収集されれば、今回のソリューションでもその応答時間が利用できる[参考文献5]。

(3) オカレンス／ホールの考え方

4章で述べたが、頻繁に変動する指標に対しては指標値の傾向把握のためにオカレンス／ホールの手法が有効である。

(4) さらにきめの細かい分析／計画への対応

当ソリューションを基に、応答時間監視機能やルール処理機能を拡張し、さらにきめ細かい監視・分析・対応の実装も可能である。例えば、ITMTPのシミュレーターにはトランザクション実行ユーザーの情報が付加できる。ユーザー属性に応じた監視・分析・対応が可能であり、優先すべき特定ユーザーの状況把握も可能となる。また、ITMTPはARM APIを用いてアプリケーション内部の応答時間を計測する機能もある。ARM APIの実装が普及すれば、その計測結果により応答時間劣化の原因となるアプリケーションを特定し、より効果的な対応が可能である。

また、今回はCPU使用率を考慮して追加サーバーを決めているが、別の選択も可能である。例えば、各サーバーに直接ダミー・トランザクションを実行させ、応答時間の短いサーバーを選択するなどである。異機種混在環境のように複数サーバーの機器構成が異なる場合には、CPU使用率などのシステム負荷状況の単純な比較はできず、応答時間の比較の方が有効である。

(5) サーバーの削除

今回のソリューションでは、不要になったサーバー資源を動的に削除している。その際に、サーバーを使用中のトランザクションが存在する可能性もあり、考慮が必要である。削除方法には、「単純なサーバーの削除」「該当サーバーへの新規トランザクションの割り振り停止」「削除すべきサーバー資源の操作員への通知」などが考えられる。「単純なサーバーの削除」は処理中のトランザクションへの影響を考慮する必要がある。「新規割り振りの停止」は終了しないトランザクションなどへの対応が別途必要となる。「操作員への通知」は人手による介入が必要となる。現実的には、「新規割り振りの停止」と「操作員への通知」を併用し、新規割り振りを停止後も長く実行されるトランザクションがあれば、操作員に通知するなどの対応がある。

(6) サーバー追加／削除時の排他制御

優先度の高いアプリケーションに新しいサーバーを割り当てる際には、優先度の低いアプリケーションにサーバーの利用をやめさせることが必要になる場合もある。また、サーバー追加時に、これまで稼働していなかったソフトウェアを起動する必要があるかもしれない。これらの割り当て時の高度な処理を実現するには、排他制御を考慮したサーバー情報のデータ構造の検討や、ルール処理へのプロセス起動処理の追加などが必要となる。アプリケーション側では優先度が下げられたときに強制停止されることを想定したデザインが必要になる。

(7)「製品機能＋通常の開発」の利用

今回のソリューションは、製品の標準機能に基づいている。ITMTPのシミュレーター機能、ITMのしきい値監視と判断機能、TECのイベント処理機能、Edgeサーバーのサーバー割り振り機能などである。ITMのリソース・モデルやTECルール、TECが起動するシェルなどの開発を伴うが、システム管理では通常に実施される内容である。製品機能を利用する利点としては、第1に、機能拡張やサポートへの安心感がある。障害時の対応や将来の機能拡張や製品移行に際しても、ソリューションに依存した特別な配慮は必要ない。第2に、お客様や社内に蓄積されたスキルや技術情報、ノウハウが豊富で容易に利用できる。例えば、応答時間のITMのオカレンス／ホールによる監視、TECのルール処理による分析・計画の実施など、の内容はお客様環境ですぐにでも利用できる。第3に、短期の開発期間で実装できる。当論文の検証システムは、HotRodなどの概念を参考にし、約1週間で開発できた。実環境でも現実的な開発工数で実装できるものである。

(8) 監視システムの信頼性向上

オートノミック・コンピューティングでは監視機能が制御機能と密に連動する。監視システムの信頼性が重要で、以下の対策が必須である。

- 単一障害点の排除：シミュレーターやイベント・サーバーの冗長化による監視システムのハードウェア障害などへの対応。
- 監視システムの監視：監視システム自身の監視。ダミー・イベントの定期的な発行による一連の制御ループの実行テストなど（制御ループとは別の仕組みによる監視が必要）。

(9) ソリューションの応用

4章で述べたが、今回のソリューションは、以下の環境への拡張を想定して設計している。

- 複数システムへの対応：一つの監視システムから複数のWebシステムの監視が可能。
- 異機種混在環境への対応：IBM製品以外のWebシステムの監視が可能。

(10) オートノミック・コンピューティングの進化レベル

今回のソリューションは、システム状況に応じて動的かつ自動的にサーバーの追加／削除を行うものであり、オートノミック・コンピューティングの進化の段階ではレベル4（適応）に当たるものといえる。システム状況に応じた対応を操作員への通知にとどめることにより、レベル3（予測）として実装することもでき、お客様の環境や要件に合わせたレベルでの実装ができる。

6. おわりに

当論文では、オートノミック・コンピューティング関連のオファリングとしても注目されている「システム状況に応じたサーバー資源の動的かつ自動的なアロケーション」を紹介した。システム状況の指標として応答時間に注目し、既存の製品機能と通常のカスタマイズや開発により、稼働状況に応じたサーバー資源の動的かつ自動的なアロケーションが実現できることを示した。オートノミック・コンピューティングの進化の段階ではレベル4（適応）に相当する。

検証環境ではあるが約1週間で構築できている。お客様環境で実装するには、5章で述べた考慮点もあり、整備すべき文書も多いが、一つのソリューション・モデルと考えられる。

この分野の技術革新は激しく、本論文執筆中にもIBMが買収したThinkDynamics社のテクノロジーを用いた新製品IBM Tivoli Intelligent ThinkDynamic Orchestrator（ITITO）によるサーバー・アロケーション事例を耳にした。技術的に検討してみると、本論文における実装や考察は、ITITOのコンセプトと多くの共通点が見られることが分かった。今後ITITOを活用することで、サーバー・アロケーションをより効率的かつ高度に実装できると考えられる。本論文のソリューション・モデルとITITOの融合は、今後取り組んでいきたい課題である。

当論文が、オートノミック・コンピューティングを既存の製品で実現するソリューションの例として役に立てば、幸いである。

[参考文献]
[1] 日本アイ・ビー・エム http://www.ibm.com/jp/e-business /technology/ 2003年8月26日
[2] 日本アイ・ビー・エム http://www.ibm.com/jp/autonomic/casestudies/hotrod.html 2003年8月26日
[3] 日本アイ・ビー・エム「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」2003年4月
[4] 日本アイ・ビー・エム http://www-6.ibm.com/jp/software/ 2003年8月26日
[5] Open Group http://www.opengroup.org/management/arm.htm 2003年8月26日