「オンデマンド」時代に向けた大規模Webシステム設計に関する一考察

A Study of Large-Scale Web System Design in Response to the Era of "On Demand"

当論文は来るべき「オンデマンド」コンピューティング時代に向けて、現在実装可能な製品技術と設計手法を用いた大規模Webシステム設計に対する考察を行ったものである。「オンデマンド」コンピューティングの実現に向けては「仮想化」と「オートノミック」という要素がキーとなるが、それらの技術標準が未成熟な現在においても準備しておくべきシステム構成やシステム設計上の考慮点が幾つかある。当論文はその中でも負荷分散とシステムの可用性向上に焦点を当てたシステム設計手法を具体的に説明したものである。

This paper is concerned with the design of large-scale Web systems using product technology and design methods that are currently capable of implementation in response to the imminent age of "on demand" computing. Virtualization and autonomics are key factors in connection with realization of "on demand" computing, and there are a number of points relating to system configuration and design that need to be borne in mind at the present time, when technological levels in this area are still immature. In this paper I have attempted to offer a practical explanation of system design methods with the focus on workload balancing and on improving system usability.

日本アイ・ビー・エム株式会社 金融ソリューション・センター
保険第4ソリューション・サービス 第1システム部
ITスペシャリスト
IT Specialist
System No.1, Insurance Solution Service No.4
Delivery - Financial Services, IBM Japan, Ltd.

岡田 経明 Tsuneaki Okada

[プロフィール]
1985年、日本アイ・ビー・エム入社。1990年より保険会社のお客様担当SE部において大規模分散システムやクライアント/サーバー領域のインフラストラクチャー面での設計・開発・導入を海外IBMと協力しリードする。現在はその経験を生かし、大規模Webシステム領域におけるインフラストラクチャー設計・運用設計を担当している。

1. はじめに

e-ビジネスの発展に伴い企業内外で使用されるWebシステムが「オンデマンド」の段階を迎えようとしている。インターネット関連技術の発展や、そのビジネスに与える影響の大きさは広く世の中に認知されるようになり、Webシステムを活用したトランザクションを企業活動の中心とする企業も増加してきている。変化に対応できる「オンデマンド」システムの構築は企業にとって次の大きな技術的課題である。

「オンデマンド」時代に求められるシステムでのオペレーティング環境の特性としては以下の四つが挙げられている。

- 統合化（Integrated）
- オープン（Open）
- 仮想化（Virtualized）
- オートノミック（Autonomic）

「統合化」や「オープン」という特性は、異なるOS（Operating System）に対する一貫したアプリケーション・インターフェースを提供するベンダーのミドルウェア（WebSphere®など）を使用したサーバーをまたがった業務プロセスの統合や、XML（Extensible Markup Language）、SOAP（Simple Object Access Protocol）、OGSA（Open Grid Services Architecture）といったさまざまな業界標準の技術の発展により、その実現の可能性が高まってきている。しかしながら現在のベンダー製品や業界標準の発展度合いでは、「仮想化」や「オートノミック」という特性においては、理想とされるレベルと現在実現可能なインプリメンテーションのレベルにまだまだ大きな隔たりがあるとの感がぬぐえない。

振り返って、企業で現在稼働しているシステムにおいて来るべき「オンデマンド」時代に対応するための仕組みは、設計当初から考慮されていないことが多い。急激なトランザクションの増加や新規ビジネス・モデルに対応したアプリケーションの追加などに対しては、その都度IT（Information Technology：情報技術）部門で対応を検討し、個別にシステム構成を追加・変更したり、新規アプリケーション開発のための開発システム装備を購入したり、既存システムとは別に本番システム装備を導

入したりしているのが現状ではないだろうか。

ミッション・クリティカルなシステムにおいて、この「仮想化」と「オートノミック」はシステムの柔軟性・可用性を向上させるための重要なシステム特性であり、その対応の実現性について現段階で検討しておくことはシステム設計の面で非常に重要である。

当論文ではこういった現状を再認識し、現在の製品技術やシステム設計により可能な「仮想化」や「オートノミック」への準備と、それにより得ることのできるシステムの具体的なメリット、また真の「オンデマンド」コンピューティングの実現に向けて、どのような課題があるかについて述べてみたい。

2. 「仮想化」と「オートノミック」に求められるシステム要件と現在利用可能な製品技術

2.1. 「仮想化」に対するシステム要件と対応する製品機能

「仮想化」に求められるシステム要件としては、①サーバーを統合し、それによりグリッド・コンピューティングが実現できること、②キャパシティー・オンデマンドの提供により必要な資源を要求に応じて提供できるシステムであり、ユーティリティー・コンピューティング環境の実現に寄与できることが挙げられる。

現段階で上記システム要件を実現するための製品や機能としてはIBM @server®のCUoD（Capacity Upgrade on Demand）やGlobusプロジェクトによって開発されたOGSA準拠のGlobus Toolkitなどが使用できる。

以上のような仮想化は最終的な理想であるが、既存のWebシステムを本格的にそれに対応させるためには、Globus Toolkitのコマンドや API（Application Program Interface）を使用したシステムへの設計変更を必要としたり、CUoDに対応したハードウェアを購入したりといったように、Webシステムに本格的に取り組んでいるユーザーにおいてもその実現には時間がかかると思われる。

「仮想化」という意味を考えてみると組織、ユーザーまたはコンピューター・システムといった資源をバーチャライズし、物理的な制約にとらわれずに効率的に業務を行うことだともいえる。

1台のシステムを異なる論理区画に分割し、あたかも複数のハードウェアで異なるシステムを稼働させているかのようなLPAR（Logical Partitioning：論理分割）も、仮想化を実現している製品機能の一つである。

2.2. 「オートノミック」に求められるシステム要件と対応する製品機能

「オートノミック・コンピューティング」を構成する要素としてIBMは以下の四つを挙げている。

- 自己構成（Self-Configuring）
- 自己修復（Self-Healing）
- 自己最適化（Self-Optimizing）
- 自己防御（Self-Protecting）

四つの構成要素のうち「自己構成」および「自己防御」に関しては、DB2® Configuration AdvisorやTivoli® Configuration Manager（「自己構成」を実現する製品）、Tivoli Risk Manager（「自己防御」を実現する製品）などで、ハードウェアの自動構成とそれに対応したパフォーマンス・チューニングのためのパラメーターの自動設定、ソフトウェアの自動配布などが提供されたり、セキュリティー侵害に対する管理／対応機能が提供されている。

ミッション・クリティカルなシステムで優れた柔軟性・可用性を提供するための「自己修復」「自己最適化」については、IBM @serverのFFDC（First Failure Data Capture）機能やWAS（WebSphere Application Server）のWLM（Workload Management）機能など、既に幾つかの製品により実現されているものもある。しかし「自己構成」や「自己防御」のように、どちらかといえば製品提供機能を使用することで実現できるものと異なり、「自己修復」や「自己最適化」はアプリケーション・レベルの設計に影響を与える要素であり、それに対応した初期段階からの検討が必要である。

例えばWASのWLMには、IHS（IBM HTTP Server）のプラグインにより、複数IHS+複数WASの構成の場合に異なるIHSを経由してもsession情報に基づき前回処理を行ったWASのJVM（Java™ Virtual Machine）にルーティングされるsession affinityという機能や、複数WASのJVM間でsession情報を外部データ（XML形式ファイル）に保管してあるJVMに障害が発生してもユーザーのトランザクションに影響を与えないようにするsession persistenceという機能がサポートされているが、これらは製品機能をインストールするだけでは期待する効果は得られず、それを意識したアプリケーションやインフラストラクチャーの設計を行うことによって、初めて有用になるものである。

3. 「仮想化」と「オートノミック」を意識したシステム構成、システム設計とそれによるメリット

3.1. システム構成上の考慮点

3.1.1. 負荷分散機能の提供

システム設計上、「仮想化」と「オートノミック」をトランザクションという観点から意識するということは、「仮想化」された資源に「オートノミック」な自己最適化されたロジックに従ってトランザクションを分配することであるといえる。

Webサーバーおよびアプリケーション・サーバーを構築するに当たってIHSおよびWASを使用する場合は、IHS-WAS間ではプラグインによりWeightedラウンドロビン（異なるWAS JVMへの割り振り済みセッション数の増減により重みを付けたラウンドロビン）による負荷分散機能があらかじめ提供されている。ただし、当機能は複数のWAS JVMがクラスタリングされた環境が前提であるため、WAS/ND（WebSphere Application Server Network Deployment）あるいはWAS/EE（WebSphere Application Sever Enterprise Edition）が必要である。

一方、Webサーバーやアプリケーション・サーバー以外も負荷分散機能を提供しないと、Webブラウザーからのリクエストが特定のWebサーバーに集中したり、アプリケーション・サーバーからのデータベース・レコードに対する入出力が特定のデータベース・サーバーに偏ったりすることになり、システム全体としてみると、すべてのコンポーネントで最適な負荷分散が提供できていることにはならない。そのためにIHSの前にIBM Network DispatcherやF5社のBIG-IPなどといった負荷分散機能を持つ製品を導入したり、データベース・サーバー側ではIBM Sysplex Distributorなどを導入したりするのが一般的である。

3.1.2. サーバーの冗長構成

ハードウェアの冗長構成およびそれを利用したソフトウェア機能による障害対応設計は、まず最初に検討すべき可用性向上のための手法である。「自己修復」により可用性を向上させる技術を使用するに当たり、サーバーの冗長構成は前提ともいえる。

ハードウェアの冗長性と一口にいってもサーバーのボックス単位からDASD（Direct Access Strage Drive）やNIC（Network Interface Card）に至るまで、そのレベルはさまざまである。当然のことながら細かいレベルまで冗長構成を適用するに従って、システムの投資コストや設計の複雑さは増大する。ここではWebサーバーおよびアプリケーション・サーバーにIBM @server pSeries®、データベース・サーバーにIBM @server zSeries®という大規模Webシステムを構築するときの典型的なハードウェア構成を前提にして、適用できる可用性向上の技術を列挙してみる（表1）。

表1. サーバーごとの可用性向上技術

サーバー	レベル	技術／手法	備考
pSeries	システム全体	N+1構成	LPAR or 複数ボックス
	CPU	動的縮退機能	SMP/3way以上
	メモリー	Chipkill ECC	
	DASD	RAID5/Mirror	
	NIC	複数NICの導入	OSPF設定により有効
		NIB（Network Interface Backup）	
	WebSphere	WASによるCloningとsession affinity	
	AIX® TCP	OSPF	複数NICが前提
zSeries	システム全体	Sysplex	
	OS/390® TCP	OSPF	

これらの技術を提案システムに適用することによって、サーバーやネットワークを含めたシステム全体としての障害設計をCFIA（Component Failure Impact Analysis）を通じて実施することで、ハードウェア／ソフトウェアおのおののコンポーネントに障害が発生した場合のアプリケーションやインフラストラクチャーへの影響を事前に確認することは、特にシステム構成が複雑な大規模Webシステムにとっては極めて重要である。

3.2. システム設計上の考慮点とそのメリット

負荷分散機能を持つハードウェアやサーバーの冗長構成を確保したとしても、それを生かすアプリケーション／インフラストラクチャー設計がなされていないとシステム全体の柔軟性・可用性は低下してしまう。以下に幾つかの例を示して、筆者が検討した設計上の工夫や考慮点について説明する。

(1) アプリケーション・サーバー障害発生時／障害復旧後のIHSプラグインによるセッション割り振りロジックの制御

前述した通り、IHS→WASへのセッション割り振りの制御はWeightedラウンドロビンにより行われているが、省略時の設定では以下のようなシナリオで問題が発生する可能性がある（図1参照）。

この問題については［参考文献1］に記載されている。

① WebサーバーのWASプラグインは、ラウンドロビン・ルールに従ってクローンへHTTP（HyperText Transfer Protocol）セッションの割り振りを行おうとする。プラグインによる割り振り対象のクローンは初期weight値=2が設定されており、割り振りを行うと-1を行うため、IHSにレスポンスを返していないクローンのweight値は1となる。

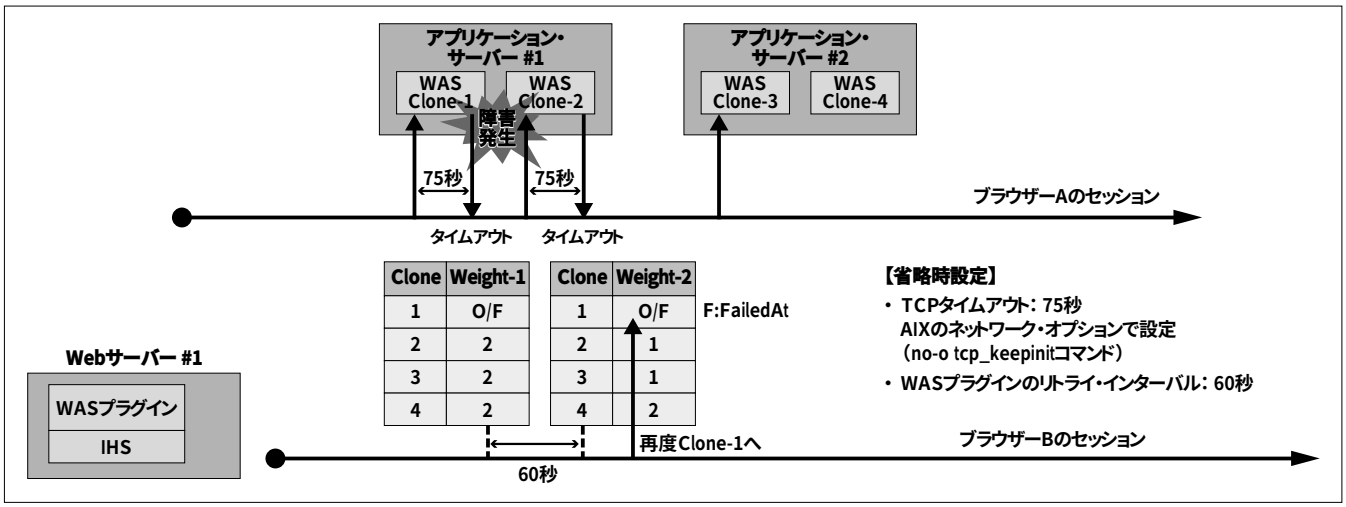

図1. IHSのWASプラグインを使用したセッション割り振りの仕組み

② アプリケーション・サーバー#1が障害中であるため、プラグインはClone-1へHTTPリクエストを割り振ろうとするが、TCPタイムアウトの75秒でタイムアウトする。Downとマークされた Clone-1はweight値が0になり、FailedAtにタイム・スタンプが記録される（図1のWeight-1の状態）。

③ 次にClone-2へ割り振ろうとするが、これもアプリケーション・サーバー#1が障害中であるため75秒でタイムアウトする。Clone-2もweight=0, FailedAtにタイム・スタンプが記録される。

④ 次にClone-3への割り振りが成功する。この時点でWASプラグインが最初のHTTPリクエストを受け取ってから75×2=150秒が経過している。

⑤ 別のブラウザーからHTTPリクエストが到着した場合、Clone-1はFailedAtからリトライ・インターバルの60秒が経過していると再度割り振りの対象に組み込まれるが、Clone-1およびアプリケーション・サーバー#1はまだ復旧していないため、タイムアウト発生の75秒が経過するまでHTTPリクエストは次の使用可能なクローンへの割り振りが行われない。これは結果的にユーザーから見るとブラウザーでの長時間waitという事象ととらえられる。

上記のようなシナリオが発生すると、IHSにHTTPリクエストが到着した時点でクローンが障害中であっても、FailedAtのタイム・スタンプからリトライ・インターバルが経過したクローンに対する使用可能かどうかのチェックが行われるために、あまり好ましい動作にはならない。WASのクローンと通信できないという状況はアプリケーション・サーバーの障害による場合が多く、その復旧にも時間がかかることから、リトライ・インターバルを1日といった長時間に設定することも極端に言えば可能である。しかしその反面、アプリケーション・サーバーが復旧したとしても即座にその状況はIHSのWASプラグインに認識されないため、サーバーのキャパシティーが不足している時間も長期化する可能性がある。この辺りの設計は、お客様から求められているサーバーの可用性に関する要件を考慮しながら、テストにより十分検証する必要がある。

WAS復旧をIHSに早期に認識させるためには、プラグイン関連のもう一つの設定であるリフレッシュ・インターバル（プラグインがplugin-cfg.xmlファイルをreloadする間隔）を短くすることによりplugin-cfg.xmlに更新があった場合は、weight値やFailedAtの値は初期値に戻るため、アプリケーション・サーバーの復旧後にこのファイルをtouchコマンドなどで更新することにより、プラグインでのクローン復旧認知時間を短縮することも可能である。

参考ながら、セキュリティー要件からIHSをDMZ上、WASをセキュアー上のノードに分割して設置する場合は、ファイアウォールの障害（ハードウェア障害やsession persistence timer値のexpireによるTCPセッション断）によっても影響を受けることに注意したい。

なお、WAS5.0.1まではWeightedラウンドロビンによるセッション数での制御しか提供できなかったため、データベース入出力や複雑な業務ロジックによりJVMを占有しても、プラグインはそれを加重することができなかったが、WAS/EE 5.0.2のダイナミックWLMによりCPU使用率や応答時間を考慮した加重配分でのセッション制御が可能になっており、より「自己最適化」された負荷分散が可能になっている。

(2) アプリケーション障害発生時、アプリケーション変更管理実施時のセッション制御

システム障害はハードウェアやシステム・ソフトウェア、ミドルウェアなどの障害だけで発生するものではなく、お客様の業務アプリケーションの不具合によっても発生し、これも「計画外停止」の時間帯として測定される。

業務区分	強制制止フラグ	開始時間	終了時間	
0	0	8:00	21:00	←8時～21時の間使用可能
1	0	8:00	21:00	←8時～21時の間使用可能
2	0	8:30	21:30	←8時30分～21時30分の間使用可能
3	0			←24時間使用可能
4	0			←24時間使用可能
5	1	8:00	21:00	←サービス停止
6	1	8:00	21:00	←サービス停止
7	0	5:00	20:00	←5時～20時の間使用可能
⋮				
98	1			←サービス停止
99	1			←サービス停止

- **業務区分:** 0～99、業務区分に対応
- **強制制止フラグ:** 障害やモジュール入れ替えなどで、当該業務を一時的に使用不可にするフラグ（1:サービス停止、0:サービス開始）
- **開始時間:** 当該業務が使用可能になる時間。24時間表示で、終了時間とともにブランクの場合は制限なし（24時間使用可能）
- **終了時間:** 当該業務が使用不可になる時間。24時間表示で、開始時間とともにブランクの場合は制限なし（24時間使用可能）

図2. メニュー制御ファイル

当然のことながら、アプリケーションに不具合がある場合は修正モジュールの本番環境への反映が必要であるが、WebSphere上で稼働するサーブレットやJSP（Java Server Pages）といったアプリケーション・モジュールを入れ替えるためには、オート・リロードの仕組みを使用するよりも、モジュールの入れ替え後にWAS JVMのリスタートを実行した方が、EAR（Enterprise Application aRchive）やWAR（Web Application aRchive）の整合性を確保するためには適切である。しかしWAS JVMのリスタートでは、修正対象でない業務アプリケーションのユーザーにとってもシステム停止時間が生じてしまい、目的であるシステム可用性の向上には寄与できない。また業務アプリケーションの期案件ごとの追加・変更といった「計画停止」の時間帯も、システムとしては全面停止になってしまう。

以下はこれを解決し「計画外停止」「計画停止」の双方に対応可能とするための設計上の工夫である。

（a）メニュー制御ファイルでの運用制御

業務ごとに運用時間帯変更や停止制御を行うために、図2のメニュー制御ファイルを提供する。個々の業務アプリケーションには「業務区分コード」「照会コード」といった識別情報をあらかじめ定義しておき、業務アプリケーションを呼び出す際には共通のフレームワークによりこのメニュー制御ファイルをチェックした後に業務アプリケーションを呼び出すようにする。

また個々のアプリケーション・プログラムの入れ替え時のために、以下のように照会コードINHIBITファイルを準備して切り替え対象アプリケーションの照会コードを追加登録することにより、運用制限を行うことも可能である。

```
menu.inhibit=0000000;1111111;1234567;
```

（b）A系／B系片方ずつの変更管理

（a）の設計だけではシステム全体がこのメニュー制御ファイルや照会コードINHIBITファイルにより影響を受けてしまう。そこで、システム全体をA／B系の2系統に分割することでシステムが全面停止になる時間帯を可能な限り短縮させることが必要であり、それを実現するために以下の技術を使用している。

（a）負荷分散装置BIG-IPでのセッション・クッキー読み取りによる新規セッションの判断とルールによるIHSへの振り分け。

（b）A／B系に分割した複数サーバーをそれぞれ別のクラスターとして定義したWAS/ND構成。

BIG-IPでは、以下の図3のような定義を事前に設定することで、セッション・クッキーに基づいたIHSへのセッション割り振りが可能になる。

図3の定義によりシステムに対して変更管理を実施する場合は、以下のようなフローにて作業を実施することにより24時間運用にも対応できる設計が実現可能である。

① A系に対する変更管理の実施を計画する。

② 新規セッションはBIG-IPのルール設定の変更によりB系に割り振る。

③ A系で継続しているセッションは、一定時間がたつとメニューから新規に業務が選択できないように、A系のメニュー制御ファイルを更新する。

④ A系セッションはメニューに戻った時点でセッション情報の消去が実施可能な状態であるので、一定時間の経過後にA系サーバーのWAS JVMを停止する。

⑤ A系サーバーに対して変更管理を実行したのち、A系の

```
rule RuleHttp {   if (exist http_cookie "A_SideCookie") {use (PoolHttpCookieExistA)
                  }
                  else if (exist http_cookie "B_SideCookie") {use (PoolHttpCookieExistB)
                  }
                  else {use (PoolHttpCookieNotExist) }
pool PoolHttpCookieNotExist {
                  persist_mode simple
                  member Webサーバー#A1のIP-Address:80
                  member Webサーバー#A2のIP-Address:80
                  member Webサーバー#B1のIP-Address:80
                  member Webサーバー#B2のIP-Address:80
                  fallback SorryサーバーのIP-Address
}
pool PoolHttpCookieExistA {
                  persist_mode simple
                  member Webサーバー#A1のIP-Address:80
                  member Webサーバー#A2のIP-Address:80
                  fallback SorryサーバーのIP-Address
}

pool PoolHttpCookieExistB {
                  persist_mode simple
                  member Webサーバー#B1のIP-Address:80
                  member Webサーバー#B2のIP-Address:80
                  fallback SorryサーバーのIP-Address
}
```

図3. BIG-IPのrule/pool設定定義ファイル

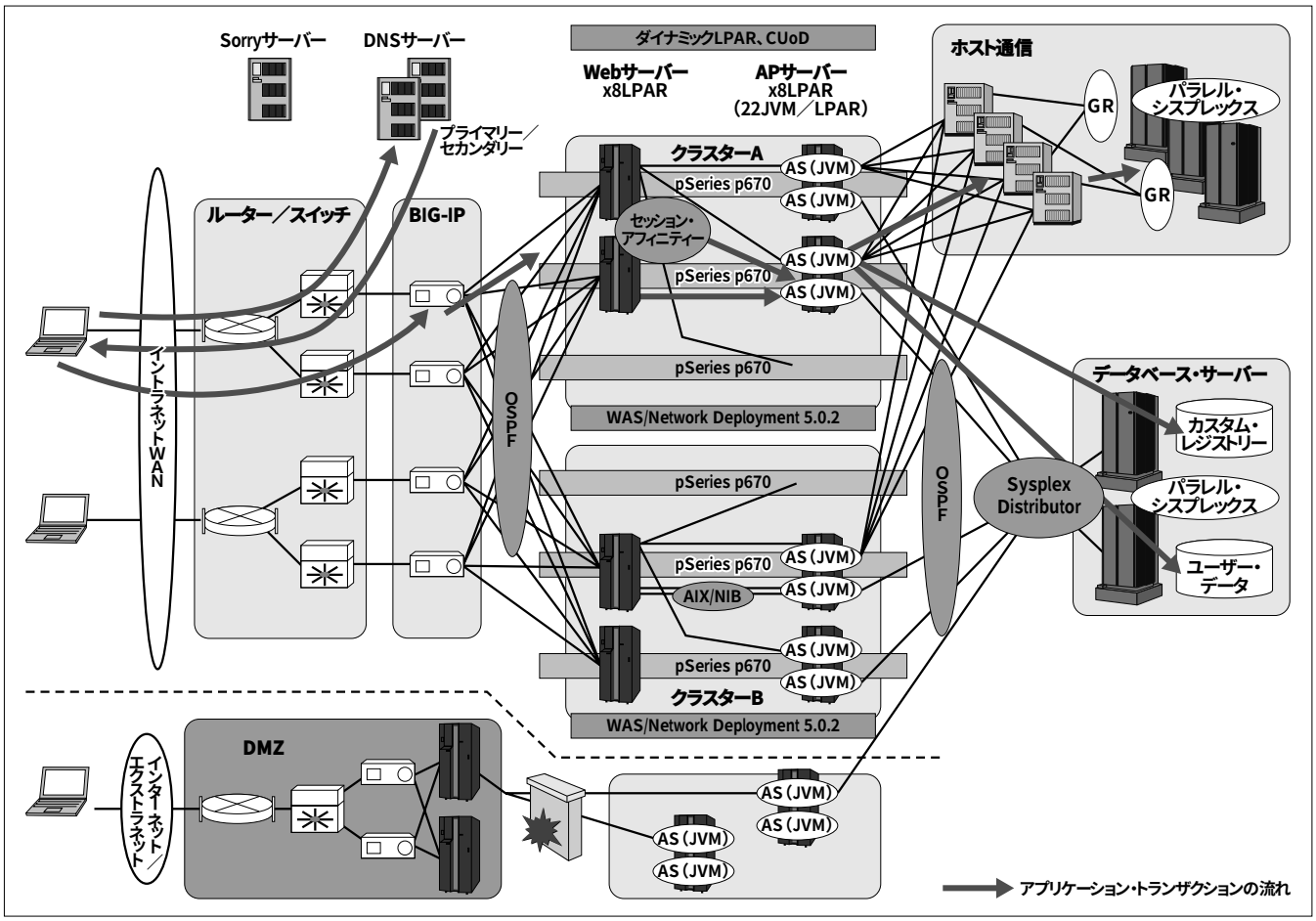

図4.「仮想化」と「オートノミック」へ実用技術を適用したシステム設計

サーバーを再始動する。

⑥ B系を変更管理対象として②～⑤の操作を行う。

これらのシステム設計上の手法と現在使用可能な技術を組み合わせたものが、あるお客様における大規模Webシステムの基盤となっている（図4）。

3.3. システム設計によって生まれた副次的効果

上記のシステム構成は、A／B系に仮想的に2分割された複数サーバー群とそれに対する負荷分散やセッション制御の仕組みを、製品と設計により最大限活用している。当システムの設計は、本番業務が稼働するシステム構成を念頭においたものであったが、A系を本番業務、B系をテストで使用することにより本番システムを仮想的にテストでも使用可能であり、テスト用装備コストの削減にも寄与できるのではないかと考えている。

4. おわりに

現時点で真の「仮想化」と「オートノミック」を実現する製品技術はまだ開発段階であり、さまざまなハードウェア／ソフトウェアがそれに対応しようと計画されている。IBM WebSphereがダイナミックWLM機能やOGSA対応を発表したことにより、グリッド・コンピューティングやオートノミックへの動きが加速する中で製品がいかに進化しようとも、それを最大限活用するとともに不足機能を補完するシステム設計を行うことが、技術者の真骨頂であることは疑いもない。

当論文がそうした技術者の一助となれば幸いである。

[参考文献]

[1] Hao Wang & Mark Bransford, 'Server Clusters For High Availability in WebSphere Application Server Network Deployment Edition 5.0', 2003.1.21