オン・デマンド・ルーター適用によるサービス品質向上へのアプローチ

岩品 友徳　山口 崇　三浦 雅史

Approaches to Quality of Service Improvements Through Application of the On Demand Routers

Tomonori Iwashina, Takashi Yamaguchi and Masafumi Miura

Webシステムにおけるサービス品質の維持・向上をサポートする製品としてWebSphere® eXtended Deployment（XD）が存在するが，これはこれまで大規模システム向けの製品と考えられてきた．本論文はより小規模なシステムに対してもXDを適用することが可能なオン・デマンド・ルーター（ODR）のトポロジーを提示し，また従来のCBR（Content Based Routing）コンポーネントとの比較を通じてODRの機能的・パフォーマンス的優位性を示す．これにより今後増えるであろうEdge Componentからのマイグレーションに際し，ODR/XDのシステム適用への指針となることを目的とする．

WebSphere eXtended Deployment (XD) is a product which supports the maintenance and improvement of Quality of Service, but one that has been thought of as a product for large-scale systems. This article discusses the topology of applying extendible On Demand Routers (ODRs) to smaller scale systems, and demonstrates their functional and performance superiority through a comparison with conventional CBR (Content Based Routing). It thereby intends to provide a guide to the application of ODR/XD systems in terms of the issues that are likely to increase with the migration from Edge Component.

Key Words & Phrases：オン・デマンド・ルーター, WebSphere XD, サービス品質, サービス・レベル・アグリーメント, コンテンツ・ベース・ルーティング
on demand router, WebSphere XD, quality of service, service level agreement, content based routing

1．はじめに

システムとして提供可能なサービスを明確化・可視化し，システムの品質向上と運用の効率化を目指す動きの中で，システムの設計/開発，運用においてはサービス・レベル・アグリーメント（SLA）を作成し，システムが満たすべきサービス品質（QoS: Quality of Service）をシステム・オーナーと契約することが一般的となってきた［1］［2］．

Webシステムで求められるQoSの例を挙げると，コマース・サイトにおいては，大多数の閲覧ユーザーの閲覧処理により，少数だが重要な購入・決済処理を行っているユーザーの処理が阻害されることは販売機会の損失につながる．これを回避するため，エンド・ユーザーのシステム利用状況に応じた優先制御が求められる．また社内システムにおいても，重要度の異なるアプリケーションが同一システム上で運用されることが増え，アプリケーション毎に優先度を制御したいという要件が多く聞かれる．SLAではこれらのQoS要件を，それぞれのシステムが最低限満たすべきレスポンス・タイムやスループットなど，客観的に評価可能な数値目標として定義する．

従来，IBM製品ではWAS ND（WebSphere® Application Server Network Deployment）Edge Componentに含まれるCBR（Content Based Routing）コンポーネントがQoS維持機能を提供していたが［3］，SLAによるQoSの保証という社会要求の変化に対応したIBM製品としてWebSphere eXtended Deployment（XD）が登場した．XDは，WAS NDが提供する基幹システム構築に必要な機能を拡張し，システム全体としてのQoS向上をポリシー・ベースでの構成によりサポートする製品である．

今回XD v6.0のオン・デマンド・ルーター（ODR）の機能検証およびパフォーマンス検証を行う中で，これ

提出日：2007年3月16日　　再提出日：2007年7月12日

まで超大規模かつハイ・トランザクションなシステムのための製品として考えられてきたXDが, より小規模・中規模のシステムに対してもQoS向上に有効であることが分かった.

本論文では前提としてXDの核となるダイナミック・オペレーションに関する機能を簡単に紹介した上で, より小規模なシステムに対しても展開可能な新しいODRのトポロジーを提示する. その後, 従来高機能プロキシーとして利用されてきたCBRとの機能比較・パフォーマンス比較を通じ, ODRの機能的優位性を明らかにするとともに, これまで議論の的となってきたODR適用の懸念事項を解消する. これにより今後増えるであろうEdge Componentからのマイグレーションに際し, 本論文がODR/XDのシステム適用への指針となることを目的とする.

2. XDのコア機能

XDではシステム・オーナーと取り交わしたSLAに基づいて, 各アプリケーションが満たすべきパフォーマンス・ゴールを, リクエストの重要度と目標応答時間からなるサービス・ポリシーとしてURLパターン毎に定義する. このサービス・ポリシーの達成/維持を可能とするために, XDは「仮想化」と「自動化」という特色を持ったシステムとなっている[4]. XDのダイナミック・オペレーションの基本となる仕組みを示したのが**図1**である.

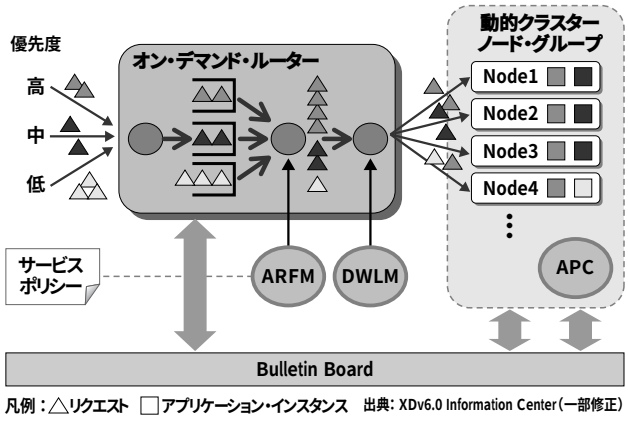


図1. XDのダイナミック・オペレーションの仕組み[5]

XDでは個々のアプリケーション実行環境（アプリケーション・サーバー）はサーバー・プールとして仮想化され, 複数ノードにまたがるノード・グループ上の動的クラスターとして定義される. 動的クラスターの前段には, 高機能プロキシー・コンポーネントであるODRが配置され, 以下の3つのオートノミック・マネジャーが連携し, 自律的なシステムとしてリクエストの処理を行う.

- ARFM（Autonomic Request Flow Manager）

ARFMはODR上で稼動するオートノミック・マネジャーである. あらかじめ定義したサービス・ポリシーに従って, ODR内のキューにリクエストを分類し, 優先順位付けおよび流量制御を行う. またそのデータをプロファイリングすることで, これから必要とされるマシン・リソースの需要予測を行う.

- DWLM（Dynamic Work Load Manager）

DWLMもODR上で稼動するオートノミック・マネジャーであり, アプリケーション・サーバーの負荷状況を動的に判断し, より最適なアプリケーション・サーバーへと割り振りを行う.

- APC（Application Placement Controller）

APCはノード・グループ内のいずれかのプロセスで稼動するオートノミック・マネジャーである. ARFMから渡される需要予測と各ノードのキャパシティを基に, 必要とされる各アプリケーションのインスタンス数と実行ノードを判断し, 動的クラスター内でアプリケーション・サーバーのインスタンス数を必要に応じて動的に変化させる.

3. ODRの非XD環境への適用構成

これまでXDの典型的トポロジーとしては, **図2**で示した構成が紹介されてきた[6].

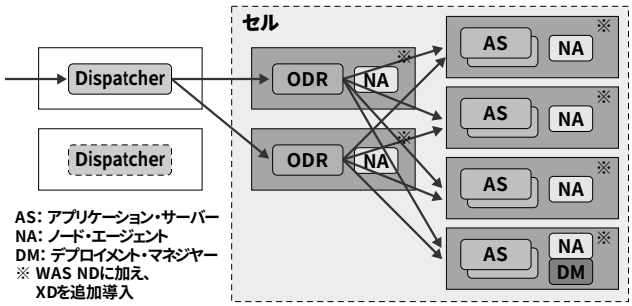


図2. XD基本構成

ODRおよびその割り振り先となるアプリケーション・サーバーにはXDを追加導入し, WAS NDの機能を拡張する. 各ノードではODRおよびアプリケーション・サーバー以外に, ノード・レベルの管理コンポーネントであるノード・エージェント（NA）が稼動する. 各ノードは集中管理プロセスであるデプロイメント・マネジャー（DM）管理のセルに統合され, ODRやアプリケーション・サーバーはノード・エージェント経由でデプロイメント・マネジャーにより管理される. XDの核となるODRは可用性を確保するため複数台配置され, その前段にODRへの割り振りコンポーネントとして, WAS ND Edge Componentに含まれるDispatcherコンポーネントが高可用性構成で配置される. 前節で示した

オートノミック・マネジャーのうち, ARFMおよびDWLMはODRプロセス上で, APCはXDの導入されたノード上のノード・エージェントまたはアプリケーション・サーバーのプロセス上で稼動する.

整然とした構成ではあるが, 一方で, 常にこの構成でODRの優先順位付け/流量制御とアプリケーションの動的配置の双方を行うことが当然であるかのように考えられ, XDは超大規模かつハイ・トランザクションな環境のためだけの製品だと考えられがちであった. このような構成をとるためには, Dispatcherノードを除く全てのWAS NDノードに対してXDライセンスを追加で購入する必要があり, 金額的理由からもXDが敬遠されてきたというのが現状である.

しかしQoSの維持, システム・オーナーと合意したサービス・レベルの保証は, システムの規模を問わず常に求められるものである. そこで本論文では, まず**図3**のようなシンプルなトポロジーでのXDの適用を検討する.

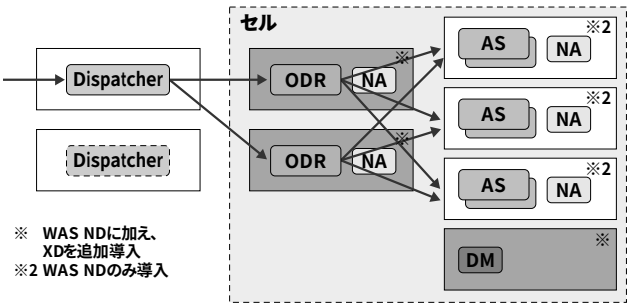


図3. WAS NDへの割り振り構成

XDのDM配下には, XDノードだけではなく, 通常のWAS NDのノードも統合することが可能である. このトポロジーにおいては, ODRおよびDMの存在するノードにのみXDを導入し, アプリケーションの実行環境はWAS NDの通常の静的クラスターとする.

この構成ではAPCによりアプリケーションを動的に配置し、ピーク時負荷に対応するという機能は享受できなくなるが, ODRの提供するQoS維持のための優先制御および流量制御の仕組みは引き続き使用可能である. 割り振り先のアプリケーション・サーバーはODRと同じセル内に所属するため, 管理の一元性も維持できる. なおセキュリティ的に許容されるならばDMをODRと同一ノードに配置し, さらにライセンスを圧縮することもできる.

XDは, **図4**に示したようにWebLogicなど他社製J2EE(Java™ 2 Enterprise Edition)サーバーやApache Tomcatなど非WebSphere環境への割り振りも, 汎用(はんよう)サーバー・クラスターとしてサポートしており, この構成においてもODRのメリットを享受することができる. この構成では, ODRは自らを通過したリクエストの状

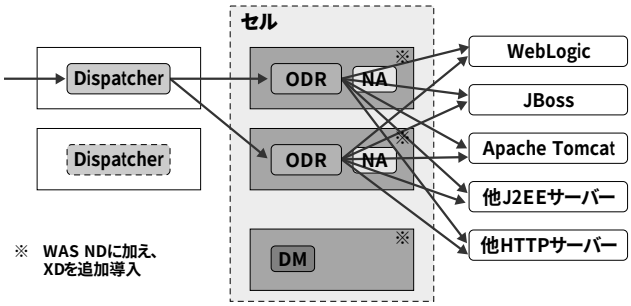


図4. 非WebSphere環境への割り振り構成

況から割り振り先サーバーのサービス状況を判断して割り振りを行うことが基本であるが, 割り振り先サーバーのCPUの速さや使用可能メモリといった情報をODRに登録することで, それらを割り振りの判断ロジックに加えることもできる. XDと同一のセルに統合されない別セル環境のWAS NDに対しても, この汎用(はんよう)サーバー・クラスターの機能を利用して割り振りを行うことが可能である. この機能を利用することで, 既存システムに手を入れたくない場合でも, QoS向上のためにXDを採用することが可能である.

なお, 図3および図4いずれの構成においても, 割り振り先サーバーのノード上にXDのリモート・エージェントを追加導入し, 直接CPU使用率などのマシン・リソース情報を取得し, ODRのルーティングをより的確なものとすることも可能である.

4. ODRの機能とパフォーマンス

これまで見てきたようにODRはWebSphere環境如何を問わず, QoS向上のためのコンポーネントとして幅広く適用することが可能である. 本章では非XD環境へODRを適用した場合の機能検証結果およびパフォーマンス検証結果を基に, ODRの機能的優位性を明らかにし, ODR適用における懸念事項を解消する.

4.1 検証構成

今回の検証では、既存の割り振り先システムには手を入れずに、これまでCBRで行っていた優先制御機能をODRにマイグレーションすることを想定して検証を行った. このため, ODRと割り振り先サーバーの存在するセルは完全に切り離して構成している. 検証トポロジーとしては, 図4の非WebSphereへの割り振り構成と同様, 汎用(はんよう)サーバー・クラスター機能を使用して2台のWAS ND環境への割り振りを行う**図5**の構成とした.

検証サーバー環境にはRedHat Enterprise Linux™ V4 ASが稼動するBladeCenter® HS20 を3台, 負荷ツール(JMeter v2.2)が稼動するクライアントにはWindows® XP SP2が稼動するThinkPad T60を2台使

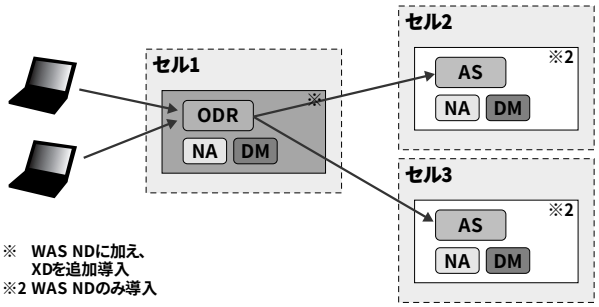


図5. 検証トポロジー

用した. 検証にはXD v6.0.2およびWAS ND v6.1.0.2を使用し, 比較の対象となるCBRもODRと同じ筐体にv6.1を導入した.

4.2 ODRの機能

本節では従来のCBRコンポーネントとの比較を通じ, ODRの優先制御およびルーティング機能の優位性を示す.

(1)優先順位付けと流量制御

これまでCBRで行ってきた優先順位付け/流量制御とODRの優先順位付け/流量制御には根本的な発想の違いがある.

CBRでは接続数ベースのルーティング・ルールを設定し, 接続数が一定数を超えた場合には特定のURLグループ以外のリクエストは全てはじいてしまうという発想であった.「関所」となる接続数ルールにより, システム過負荷時には重要ではないリクエストは全てSorryサーバーへと転送される. サービス・レベルの異なるURLグループが複数ある場合には, 接続数ルールとコンテンツ・ベース・ルールを組み合わせて定義し, 何段階にもこの「関所」を用意する必要があった.

この方法はシステム負荷が高まった際に特定のユーザーのリクエストを守るという点では非常に有効である. 一方でそれ以外のユーザーのリクエストをはじいてしまうことになるため, 負荷が下がりシステムを低優先度のユーザーに再開放した際に, 再びリクエストが集中しかねないという問題点を抱えていた.

ODRの流量制御は, 優先度の低いリクエストをはじくのではなく, キューイングして「待たせる」という点に, これまでのプロキシー・サーバーにはない新規性がある. もちろん, システムの負荷が高まった際に, より優先度の高いリクエストをより多く通すよう流量制御は行われる. しかし, 優先度の低いリクエストを全てはじいてしまうのではなく, ODR内にキューイングして待たせつつ, 比率は少ないながらも割り振りを行い続ける. これにより, システムの開閉塞を繰り返すことによって逆にリクエストのバーストが生まれる可能性を回避している.

ODRの優先順位付けと流量制御機能の挙動を確認するため, 負荷ツールによる検証を行った. この検証では同一の処理を行うサーブレットを異なる複数のURLで呼び出し, URL1に対しては最高の優先度を, URL2に対しては最低の優先度を持つサービス・ポリシーを定義した.

優先順位付けによる流量制御はサービス・レベルが満たされていない状況でより顕著に効果が現れるため, 検証用にいずれのURLに対しても達成不可能なレベルの目標を設定した(**表1**).

表1. 設定したサービス・ポリシー

サービス・ポリシー	優先度	目標時間
SP1(URL1)	最高	10msec
SP2(URL2)	最低	30msec

リクエスト数1000件を固定とし, 同時アクセス・ユーザー数にあたるスレッド数を500として, 負荷ツールより2つのURLに対して負荷をかけた場合のODRの流量制御の挙動を示したものが**図6**である.

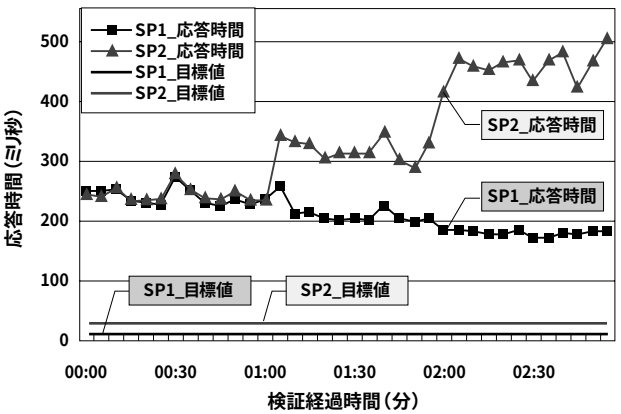


図6. ODRの流量制御の挙動

負荷をかけ始めてすぐは, ODRは双方のURLに対して均等な割り振りを行っているが, 1分ほど過ぎたところでODRは目標がいずれも満たされていないことに気付く(ARFMのプロファイリング・データがデフォルト1分間隔で集計されているため). この状況に対応するため, ODRはより優先度の高いSP1が目標時間を達成できるようSP2の要求を待たせて, SP1の要求を優先的に割り振っている. さらに1分ほど経過すると, 前回のサイクルでの優先制御では優先度の高いSP1が目標レスポンスを達成することができていないため, よりSP2の要求を待たせるようにして優先的にSP1の割り振りを行い, SP1の目標達成を支援している.

このようにODRはあらかじめ定められたサービス・レベルの維持に向けて自律的に判断し, 優先順位付け／流量制御を行う. 接続数で全てをはじいていたCBRに比べ, ODRではサービスの目標達成状況に応じた, より柔軟かつ的確な優先制御が可能となっている.

(2) ルール・ベース・ルーティング

CBRとODRの対応しているルーティング・ルールは, **表2**の通りである. 以下にODRとCBRのルール・ベース・ルーティングの差異の特徴的な点をまとめる.

表2. 対応しているルール・タイプ

タイプ	CBR	ODR
コンテンツ・ベース	○	◎[1]
IPベース	○	◎[1]
接続数ベース	○	×
時間ベース	○	×

1：CBRよりも多くのパラメータが利用可能な点およびAND条件での設定が可能な点を評価し◎とした.

CBRでのコンテンツ・ベース・ルールでは, URLもクッキー値・パラメータなども全て同列の扱いであった(必ずしもURLに結び付けてルールを記述しなくてよい). 一方ODRではURLのグループ(Work Class)毎にルールを設定するため, 全てのルーティング・ルールはURLにひも付けて作成される. このためODRではIPアドレス・ベースのルーティング・ルールもURLにひも付けられる. あるURLに対して特定IPアドレス(例えばテスト環境マシンのIPアドレス)から要求が来たら, テスト用アプリケーションに割り振るといったことがODRでは可能であり, 柔軟なルーティング制御ができる. これはコンテンツ・ベース・ルールとIPアドレス・ベース・ルールのAND条件ができなかったCBRにはできなかった割り振りロジックである(CBRの場合は, 特定IPからの要求は全て特定サーバーに割り振るという形になる).

一方, ODRでサポートされないルーティング機能も存在する. 接続数ベースで要求をはじくためのルールがODRではサポートされない. これは4.2-(1)優先順位付けと流量制御で述べた通りである. またシステムの開閉局に用いるような時間ベースのルーティング・ルールもサポートされていない. これを補うためには, システム全体の開閉局にはODRの前段に配置するDispatcherの時間ベース・ルールを用いることができる. 特定アプリケーションのみ開閉局を行いたい場合には, wsadminコマンドを用いて動的にODRのルーティング・ルールの割り振り先を変更することとなる.

4.3 ODRのパフォーマンス

本節では, ODRがJVM(Java Virtual Machine)で稼動することで常に懸念事項とされてきた, ODRの負荷分散コンポーネントとしてのパフォーマンスおよびガーベージ・コレクション(GC)がパフォーマンスに与える影響について検証結果を基に議論する.

検証では, まず純粋な負荷分散コンポーネントとしてのパフォーマンスを確認するため, 優先制御を行わない単純ルーティングでのパフォーマンス検証を行い, 次にODR本来の用途である優先制御のサービス・ポリシーを設定してパフォーマンス検証を行った.

(1) 単純ルーティング

まず優先制御を行わない単純ルーティングの際のパフォーマンスを測定した. ODRではURL別のサービス・ポリシー作成はせず, 全てデフォルトのサービス・ポリシーを使用した. CBRでもルールは作成していない. 各スレッドのリクエスト数を1000件に固定とし, 同時アクセス・ユーザー数にあたるスレッド数を変化させた場合の結果が**図7**である.

着目すべきはODRが競合度の高いケース(600スレッド以上)において安定的に4000リクエスト/秒レベルのスループットを出している点である. 750スレッド以上ではCBRのスループットを上回ってさえいる.

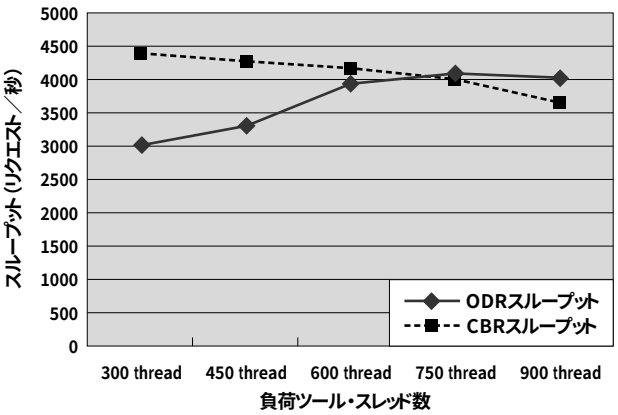


図7. 単純ルーティング・スループット比較

Javaであっても繰り返し呼び出されるメソッドはJIT(Just In Time)コンパイラーによりあらかじめコンパイルされ, ネイティブと遜色ない速度での応答が可能である. 常に懸念されてきた「Javaだから遅い」という発想は当たらないといえる.

競合度が少ないケースでODRのスループットが伸びていないのは, 負荷ツールによる突然のリクエストに対してODRの緊急スロットルと呼ばれる安全弁機能が働いているためである. これは急激にきたリクエストの割り振りを絞ってODRでキューイングを行い, 割り振り先サーバーへの急激な負荷を回避する機能

である. ODRは割り振り先サーバーの状況を確認した上でこの緊急スロットルの解除を行う. スレッド数(同時アクセス・ユーザー数)の少ないテスト・ケースでは, 総処理リクエスト数が少なく, 全体の検証時間が短い. このため緊急スロットルの効いている時間が検証時間に占める割合が高く, スループットの平均値を下げている. 一方, CBRは特にそのような安全装置機能は持たないため, 負荷が少ないうちは素直に高いスループットを出すことができているが, 負荷が高まるに連れ, 競合によりスループットが漸減している.

ODRは, JVM上で稼動するため常にGCの影響が議論の対象となってきた. しかし今回の検証結果からはGCが原因と考えられる顕著な応答時間の遅延は見受けられなかった.

補足検証として, 同一の負荷状況(600スレッド)でヒープ・サイズの変更を行い, ヒープ・サイズがODRのGCに与える影響も確認した. しかし, いずれのケースにおいてもGC時間は短く, テスト・ケースによるスループットの差異はほとんど見受けられなかった(**表3**).

表3. ヒープ・サイズとGCの傾向

ヒープ・サイズ(最小-最大)	GC間隔	GC時間(ミリ秒)
50-256 (デフォルト)	2秒弱	90-110
256-512	4秒弱	120-150
512-1024	9-12秒	150-200

今回の検証結果より, ODRのパフォーマンスはCBRと比しても遜色がなく, むしろ競合度が高い状況ではCBRよりも優れたパフォーマンスを発揮することが分かった. またGCの影響でクライアントへのレスポンスが遅延するといった従来の懸念も払拭(ふっしょく)された.

さらにODRを配置した際の特徴として, 割り振り先サーバーのCPU使用率が安定した曲線を描くことがあげられる(**図8**). これはCBRの割り振り先サーバーのCPU使用率が振れの大きいグラフを示す点(**図9**)と対照的である. これはODRがリクエストを一旦キューイングし, 状況を確認しつつ割り振るためであるが,

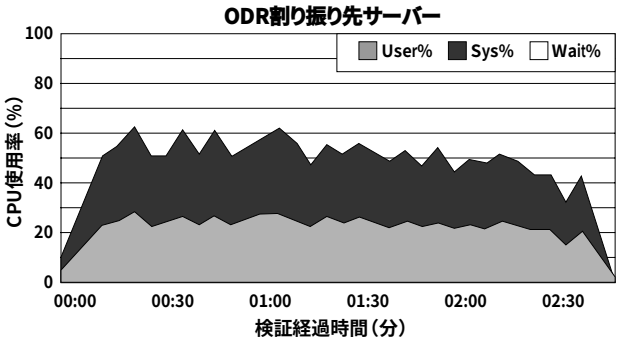


図8. ODR割り振り先サーバーCPU使用率

突発的な負荷がかからないという点はシステム安定稼動のためには非常に有益である.

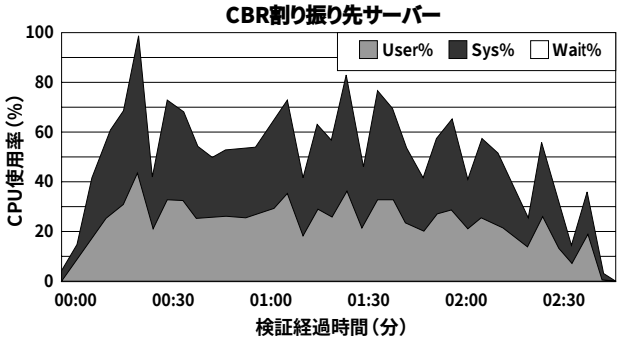


図9. CBR割り振り先サーバーCPU使用率

(2) 優先制御ルーティング

次に優先制御ルールを設定した場合のスループットへの影響を見るために, 4つのアプリケーションに対して, それぞれ異なる優先度レベルの割り振りルールを設定して検証を行った. この検証結果が**図10**である. 優先制御を行った場合は, ODRの方が常に良いパフォーマンスを示した.

CBRで必要となるルール数は, 優先度レベルx{初回ルーティング用ルール(1個)+アフィニティ用ルール(割り振り先台数分)+接続数制限ルール(1個)}で求められ, この検証ではCBRでは16個のルールを定義する必要があった. 負荷が高まった場合には、マッチング処理のオーバーヘッドの影響が大きくみられ, スループットが伸びていない. 一方ODRでは優先度レベル毎のサービス・ポリシーのみ指定すればよく, アフィニティはルールとして設定する必要がない. このため, 優先制御用の設定(サービス・ポリシー)は4つと少数のままである.

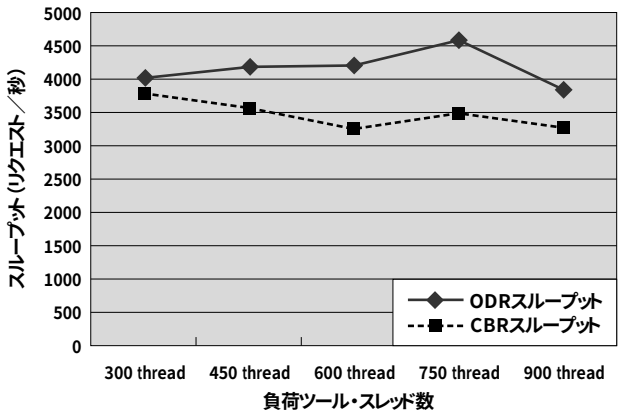


図10. 優先制御ルーティング・スループット比較

CBRでは割り振り先サーバー数に比例してルール数が増えるため, より多くの割り振り先サーバーがある環境では, マッチング処理のオーバーヘッドがより

顕著に出ると考えられる. この点からCBRに比べODRの方が拡張性・メンテナンス性に優れているといえる.

5. おわりに

今回の検討・検証により以下の知見が得られた.

1) XDは大規模システムに限らず, より小規模なシステムに対しても適用可能であり, 小規模システムにおいても十分にQoS向上の効果が得られる.
2) ODRではCBRに比べ, より柔軟な優先制御・ルーティングが可能であり, 拡張性・メンテナンス性にも優れている.
3) 従来懸念されてきたODRのパフォーマンスはCBRと比べても遜色がなく, 高い負荷でも安定してスループットを出すことができる. またGCのレスポンスへの影響も特に問題とならない.
4) 全てをXDにせずともその効果は得られる. まず足がかりとして, ODRのみのシステムへの適用から始めることも十分に検討対象となりうる.

本論文の知見を踏まえることで, 今後増えるであろうEDGE Componentからのマイグレーションに際して, ODR/XDをシステムに適用しサービス品質の向上を図る強い動機付けと指針が示せたと確信する.

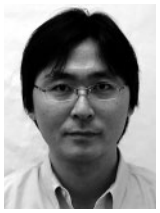
参考文献

[1] 社団法人 電子情報技術産業協会(JEITA) ソリューションサービス事業委員会 : 民間向けITシステムのSLAガイドライン第三版, 日経BP社, ISBN 4-8222-6205-7 (2006).
[2] 独立行政法人情報処理推進機構 : " 情報システムに係る政府調達へのSLA導入ガイドライン, ". http://www.meti.go.jp/policy/it_policy/tyoutatu/sla-guideline.pdf (2004.3)
[3] 小宮聖則 : " 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, "PROVISION, No.45, pp67-74 (2005).
[4] High Performance On Demand Solutions: *Virtualization and Automation: Best Practices for a WebSphere Application Server Environment*, IBM White Paper (2005).
[5] WebSphere XD v6.0 Information Center: http://publib.boulder.ibm.com/infocenter/wxdinfo/v6r0/ (2006.5.20).
[6] C. Satdler et al: *Best Practices for Implementing WebSphere Extended Deployment*, IBM Redbook, SG24-7343 (2007).

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
ワークプレース
ITスペシャリスト

岩品 友徳 Tomonori Iwashina

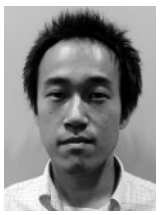
[プロフィール]
2002年, 日本アイ・ビー・エム システムズ・エンジニアリング株式会社 入社. 入社以来, WebSphere製品の技術サポート・チームの一員として, Webフロント・システムを中心に, 一貫してWebシステム基盤の構築を担当.



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
ワークプレース
主任ITスペシャリスト

山口 崇 Takashi Yamaguchi

[プロフィール]
1994年, 日本アイ・ビー・エム株式会社 入社. アジア各国を対象としたWebSphere製品の技術サポート・チームの一員として, 主にWebSphere XDなどの高トランザクション・高可用性の求められるエンタープライズ・システムのデザインおよび構築を担当.



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
ワークプレース
ITスペシャリスト

三浦 雅史 Masafumi Miura

[プロフィール]
2002年, 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 入社. WebSphere製品の技術サポート・チームの一員として, 携帯端末用Webサイト, 企業社内ポータルなどWebシステムのデザインおよび構築を経験.
現在は, 主にWebSphere XDを使用したパフォーマンスと可用性が求められる基幹系システムのデザインおよび構築を担当.