情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2009-EVA-30 No.4
2009/11/16

仮想化システムにおけるファジィ制御理論を用いた資源管理

二星　翔[†1]　大井　仁[†1]

近年，多くの企業が仮想化技術によってシステムの統合を果たしている．仮想化によるシステム統合は物理的なシステムの台数・管理コストを削減し，計算資源の有効活用を可能にする．このような統合された仮想化システム内では一つの物理システムを複数のドメインで共有しているため，資源管理の手法によっては仮想化システム全体の性能を損なう可能性があり，仮想化資源の配分方法に有効な手法が求められる．

しかし，既存の仮想化システムではドメインの状態に応じて資源配分をおこなっているため，さらなる効率的な資源の配分を行うためにはアプリケーションの状態を元にした資源管理が必要となる．現在，仮想化によって集約されたシステムにおいて，ファジィ制御[6)7)]を用いたアプリケーションの状態を反映させる動的な資源管理を開発している．本報告ではシステムの概要および現在までの計測結果について報告する．

Application of Fuzzy Control Theory to Resource Management in a Virtualized System

Sho Niboshi[†1] and Hitoshi Oi[†1]

In recent years, companies choice virtualization technology for the integration of computer systems. The virtualized system has effective use of computer resources by reducing the number of computers and financial costs for administration. In a virtualized system, all resources must be controlled by resource manager and it allocates shared resources to each domain. Since this control method affects an entire virtualized system, allocation method needs to achieve more effective.

However, the resource manager in current systems only utilizes the information from the operating system on each domain and lacks the running state of the application running on it. We are developing the resource controller that adapt to dynamic workload and application states on a virtualized system. The relationship between the applications' running states and their required resources are often vague, complex and empirical. To represent such relationship, we use fuzzy control theory to maximize the total system performance. This technical report describes the brief of our controller and its experiment results.

1. はじめに

仮想化技術は複数のシステムを一つの物理的なシステムに統合することを可能にする技術である．複数のシステムで一台の物理システムを共有することにより，システムの構築に本来必要なコンピュータの台数や管理および維持コストの削減が可能となり，有効なシステムの運用が実現される．このように複数システムによる資源の共有は仮想化システムの特徴であり，そのための仮想資源管理は仮想化システムの非常に重要な仕組みである．よって仮想化システムの性能向上や大規模なシステム統合を可能とするためには，仮想化資源の配分方法に効率的な方法が求められる．

一方，仮想化システム上での一般的な共有資源の配分方法は各ドメインおよびその上で実行されるオペレーティングシステム（OS）からもたらされる情報によって決定されるため以下のような状況を招く可能性がある．

不必要な資源確保： ドメインに本来必要な量を超える過剰な資源分配

偏った運用： 一部のドメインが十分な性能を発揮しているのに対して，他のドメインでは低い性能しか発揮できていない

これらの状況を招く原因は，OS 側からではドメイン上で動くアプリケーションの状態を把握することが難しいためである．通常，アプリケーションは性能指標として Quality of Service （QoS） を持つが，これはアプリケーションの特性や負荷の状態など様々な要因によって変化するため OS では QoS を推測することは困難である．

このような資源管理の問題に対応するため，各ドメインの負荷や QoS に応じて動的な資源配分を行う新しい資源管理を開発している．本研究では資源管理の制御手法としてファジィ制御を取り入れた．ファジィ制御は数式を利用せず経験的な知識や直感的な思考を入出力とし，非線形の数値や複数の入力を必要とする制御において特に有効的な制御手法である．また，経験則や直感思考に沿った情報を元に制御を行えるため，比較的容易に制御構造を作り出すことが出来る．

本報告では各ドメインの QoS と連携した資源管理の概要と仕組み，現段階での資源管理の検証について述べる．

†1 会津大学
Aizu University

2. 関 連 研 究

仮想化システムにおける資源管理の研究は様々なアプローチで進められている．

たとえば，1) は仮想化されたシステムにおける CPU 資源の管理に関する研究である．この研究ではノード間にまたがったアプリケーションとデータベースの 2 層サーバーシステムを対象として，数式モデルを利用した資源管理の制御を行っている．各ドメインが一定の CPU の使用率を維持するように CPU 資源の配分を決定しており，資源配分が衝突するような高負荷時にはドメイン間の QoS の比率に応じた配分に変更している．

2) は 1) をさらに進めた研究で，より複雑な形態の仮想化システムにおいて CPU と I/O 資源の管理を行っている．

3) では CPU 資源ではなくメモリ資源の動的な資源配分を仮想化システム上で行う研究である．こちらの研究では各ドメインの Page Fault を利用し，ドメインへのメモリ資源の割当を動的に行っている．

本研究ではファジィ制御を用いることで複雑な数式モデルを必要としない資源管理を開発している．

3. ファジィ制御を用いた資源管理

3.1 制御の概要

本研究では主要資源として特に CPU 資源の管理を行った。開発中の資源管理システムの構成を **図 1** に示す．各ドメインへの資源配分は以下の 3 段階を経て決定される．

(1) 各ドメイン上の QoS の計測： 各ドメイン上で実行されるアプリケーションの動作を示す QoS を選択し，状態を定義する．

(2) QoS に基づいた資源要求量の決定： (1) で定義し計測された QoS をもとに，アプリケーションが十分に性能を発揮する資源要求量を決定する．

(3) ドメイン間での要求量の調停： (2) で決定された資源要求量の総和はシステム全体を上回るケースがありうる．その場合，各ドメインの資源要求量を用い，その比率に応じて資源を分配するように要求量を調停する．

3.2 ファジィ制御を用いた制御手法

ファジィ制御ではメンバーシップ関数とファジィルールによって制御を決定する．メンバーシップ関数は値の状態を定義するために使われ，ファジィルールは制御ルールを記述することで制御アルゴリズムを決定する．ファジィルールは通常“IF A -THEN B”の形で記述され，各ルールでは入力状態に対する制御出力状態が示されている．

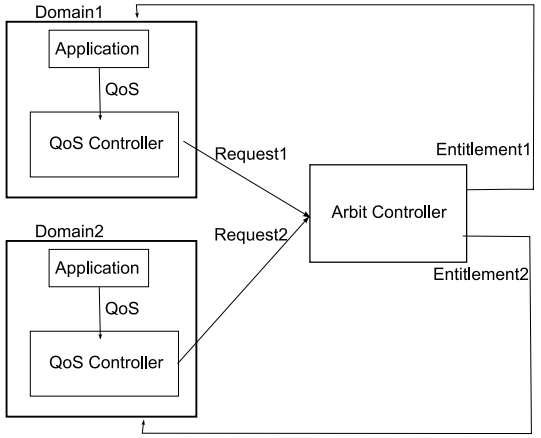

図 1 資源管理の概要

本研究の資源管理ではファジィ制御を各ドメインの QoS から資源要求量を決定することに利用しており，ファジィ制御は入力として与えられた QoS の状態をメンバーシップ関数によって取得，ファジィルールを参照することで QoS の状態に対する要求資源量を決定する．

3.3 調 停 制 御

しかし，ドメインの資源要求量が仮想化システム全体の許容を超えている場合，例えばドメイン 1 が 60%の資源を，ドメイン 2 が 70%の資源を要求した場合には両者の要求を調停する仕組みが必要になってくる．このような衝突が起きた場合，資源管理システムはドメインの資源要求量の比率に応じて資源を分配する．

4. 資源管理の実装

本研究では資源管理システムの開発のため二つのドメインを作成し，各ドメインをメールサーバー，Java サーバーとして動かした．仮想化システムには Xen Hypervisor[8] を用い，OS は各ドメインともに CentOS 5.2 を用いた．仮想化に用いた物理マシンの環境は **表 1** に示す通りである．各資源はドメイン間で共有されているが，2 台の HDD の内 1 台はメールサーバーが占有して使っている．

各ドメインへの負荷はメールサーバーに対しては SPECmail2001[4] を，Java サーバーに

対しては SPECjbb2005[4] を用いた．SPECmail2001 は SMTP と POP3 プロトコルによるメールシステム，SPECjbb2005 は 3 層のクライアント・サーバーシステムのエミュレートによるサーバー側の Java システムに対するベンチマークである．各ベンチマークに対してドメインの CPU 使用量が 50%となる負荷を想定負荷 $wMedium$ と定め，この負荷に対して想定を若干下回る負荷 90%程度を $wLow$，若干上回る負荷 110%程度を $wHigh$ と定め，これら三つの負荷の元で実験を行った．ただし，SPECjbb2005 に関しては負荷 100%と 110%では結果にわずかな違いしか見受けられなかったため，負荷を多少調整している．

各ドメインへの CPU 資源の割当には Xen の Credit Scheduler を用いた．Credit Scheduler にはドメインごとに使用できる CPU 量の上限を定める Cap 機能が備わっている．本研究ではこの機能を利用して CPU 資源の割当を行った．

4.1 QoS の選択

最初に各ドメインにおける QoS を決定する．通常，メールサーバーはメッセージ配送を行う SMTP サーバーとメッセージの受信を行う POP3 サーバーの二つアプリケーションから構成されているためメールサーバーの QoS として SMTP と POP3 双方のアプリケーションから QoS を取得した．様々な QoS を検証した結果，SMTP サーバーからは $DeliveryTime$（1 SMTP メッセージあたりの平均処理時間），POP3 サーバーからは $LoginRate$（POP3 セッションの平均ログイン成功率）を QoS として選択した．Java サーバーでは Java アプリケーションのみが動作しているので $TransactionTime$（1 トランザクションの平均処理時間）のみを QoS として設定した．

図 2，**図 3** および **図 4** は各ドメイン上で 20%から 80%の CPU 量を与えた時の $wLow$，$wMedium$，$wHigh$ の三種類の負荷に対する QoS の変化を示したグラフである． 図 2 と図 3 はメールサーバーにおける QoS，$DeliveryTime$ と $LoginRate$ をそれぞれ示している．$DeliveryTime$ は CPU 量を下げるごとに上昇していくが一度でピークを迎えた後徐々に減少していくのが見える（図 2）．一方，図 3 では $LoginRate$ は一定の CPU 量までは 1 を保っているが，それ以下に CPU 量を下げると値が急激に減少する．$LoginRate$ が 1 を切る CPU 量は $DeliveryTime$ のピークとほぼ重なっていることから，メールサーバーでは以下のような状態になっていると考えられる．$DeliveryTime$ のピーク時までは SMTP サーバーと

表 1 実験環境

CPU	Athlon64 X2 (Dual Core) 2000 MHz
Memory	4GB
HDD	SATA 60GB ×2

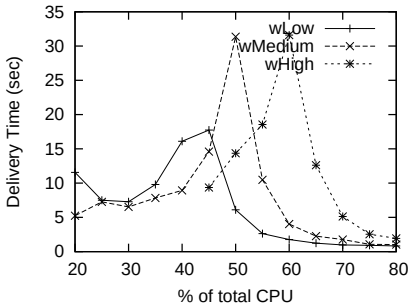

図 2 DeliveryTime

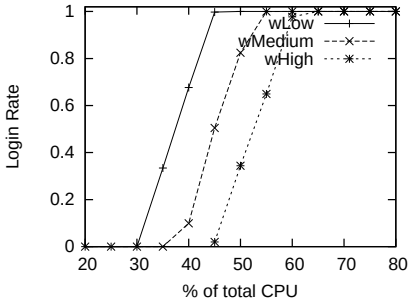

図 3 LoginRate

POP3 サーバーは双方が順調に動作しているが，それ以降では POP3 サーバーは処理を落とし，最終的に機能しなくなる．POP3 サーバーが機能しなくなるまでの過程で，POP3 の処理数が減少したため SMTP サーバー側に供給される CPU 量が増え $DeliveryTime$ は減少を始めるが，最終的に POP3 サーバーがダウンしてした後では再び SMTP サーバーも処理を落とし始め，結果として $DeliveryTime$ が上昇している．$DeliveryTime$ が山形をとっているのはこのためである．このように選択した二つの QoS，$DeliveryTime$ と $LoginRate$ がメールサーバーの性能を示していることがわかる．

図 4 は Java サーバーにおける QoS である $TransactionTime$ と CPU 量の関係を示したグラフである．$TransactionTime$ は CPU 量の減少に対して，非線形ではあるが単調減少している．このことから Java サーバーにおいては $TransactionTime$ が性能を示していることがわかる．

4.2 ファジィ制御の実装

本研究における資源管理の目標は最低限の CPU 量で QoS の状態を $Medium$ に保つ事である．そのための資源管理は，QoS が低ければ CPU 量を上げ，高ければ CPU 量を下げ，最適な状態であれば現在の CPU 量を維持するように CPU 量の調整を行う．本実験では両ドメインともに CPU 量のメンバーシップ関数として共通の定義（**図 8**）を利用する．図 8 における状態名はそれぞれ，$NegativeLarge(NL)$，$NegativeMedium(NM)$，$NegativeSmall(NS)$，$Zero(ZE)$，$PositiveSmall(PS)$，$PositiveMedium(PM)$，$PositiveLarge(PL)$ を表す．

メールサーバーにおける QoS の状態を定義する．図 2 において $DeliveryTime$ の変移を追っていくと，$DeliveryTime = 4$ の前後でグラフの傾きが大きく変化しているのが見られ

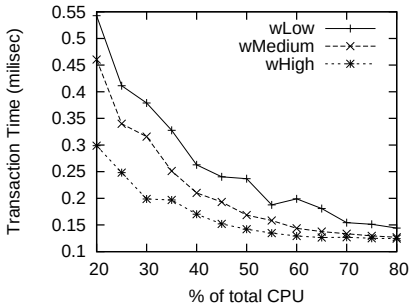

図 4　TransactionTime

る．この変曲点を $DeliveryTime$ における $Medium$ として決定し，同時にメールサーバーにおける $Medium$ として定義する．図 3 からこのような $DeliveryTime = 4$ を満たすためには $LoginRate$ は常に 1 を保たなければいけない．従って，$DeliveryTime$ のメンバーシップ関数を **図 5**，$LoginRate$ のメンバーシップ関数を **図 6** のように定義すると，最小の CPU 量でメールサーバーの QoS が $Medium$ である条件は，$DeliveryTime$ が $DeliveryTime$ が $Medium$ かつ $LoginRate$ が $High$ の場合である．従ってメールサーバーのファジィルールは **表 2** のように記述される．

他方，図 4 において $TransactionTime$ に明確な変曲点がみられなかったため $TransactionTime = 0.25$ を $TransactionTime$ の $Medium$ と決め，これを Java サーバーにおける $Medium$ として定義した．従って，資源管理システムは $TransactionTime$ が $Medium$ な状態に保つように CPU 量を増減させる．$TransactionTime$ のメンバーシップ関数を **図 7** のように定義すると，ファジィルールは **表 3** のようになり，$TransactionTime$ が低ければ CPU 量を減らし，高ければ CPU 量を増やすことで状態の調節を行う．

メールサーバー，Java サーバー双方ともファジィ制御の出力は前回の CPU 量に対する増減（Δ_{cpu}）で表されるため，最終的な CPU 資源の要求量は

$$Request = PreviousEntitlement + \Delta_{cpu} \quad (1)$$

となる．

4.3　調停制御の実装

調停制御は各ドメインの CPU 資源要求の合計が仮想化システムの許容量である 100%を

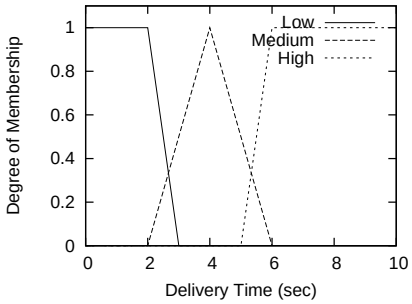

図 5　メンバーシップ関数：DeliveryTime

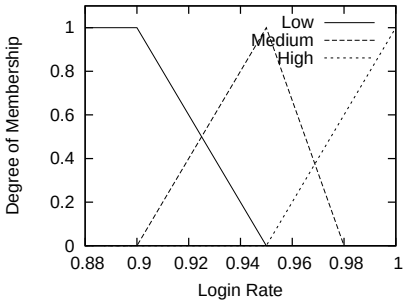

図 6　メンバーシップ関数：LoginRate

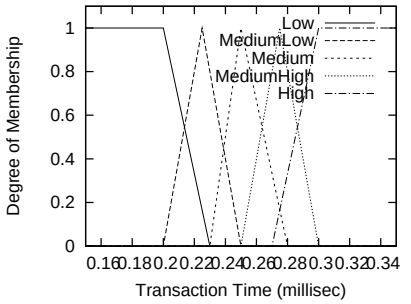

図 7　メンバーシップ関数：TransactionTime

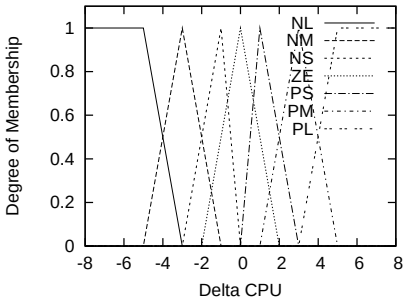

図 8　メンバーシップ関数：Δ_{cpu}

超えていれば，下式のように資源を配分量に基づき比例配分する．

$$Entitlement_i = (Request_i / \sum_{j=1}^{n} Request_j) \quad (2)$$

5. 実験結果

4節で記した環境の下，二つのドメイン，メールサーバーと Java サーバーにおいて資源管理の動作検証を行った．資源管理システムは 10 秒の周期で動かし，実験には **表 4**，**表 5** で示す二つ負荷のシナリオを用いた．各シナリオは最初 10-190 秒では各ドメイン共に $wMedium$ の負荷を与えるが，その後シナリオ 1 ではメールサーバーの負荷を，シナリオ 2 では Java サーバーの負荷をそれぞれ $wHigh$ に変化させている．その後，そのまま負荷を 590 秒まで継続した後，600 秒目で負荷を再び $wMedium$ へと戻した．実験結果は次の三つの観点から評価を行った．

資源使用率： 各ドメインが与えられた資源に対してどれだけの資源を使用しているか

各システムの性能： 各ドメインがある最低限のサービスを提供しているのか

仮想化システム全体の性能： 各ドメインが均等な性能を出し，偏ったシステム運用を行っていないかどうか

表 2 ファジィルール：メールサーバー

DeliveryTime	LoginRate	Δ_{cpu}
Low	Low	PositiveLarge
Low	Medium	PositiveMedium
Low	High	NegativeSmall
Medium	Low	PositiveLarge
Medium	Medium	PositiveMedium
Medium	High	Zero
High	Low	PositiveLarge
High	Medium	PositiveMedium
High	High	PositiveLarge

表 3 ファジィルール：Java サーバー

TransactionTime	Δ_{cpu}
Low	NegativeLarge
MediumLow	NegativeMedium
Medium	Zero
MediumHigh	PositiveSmall
High	PositiveMedium

表 4 負荷のシナリオ 1

Interval	Mail	Java
10-190	wMedium	wMedium
200-590	wHigh	wMedium
600-790	wMedium	wMedium

表 5 負荷のシナリオ 2

Interval	Mail	Java
10-190	wMedium	wMedium
200-590	wMedium	wHigh
600-790	wMedium	wMedium

5.1 計測結果：シナリオ 1

図 9，**図 10**，**図 11** および **図 12** はシナリオ 1 における各 QoS と CPU 量の時間変化を示したグラフである（各グラフのデータ名 WC，WoC はそれぞれ資源管理システムを利用した場合，しなかった場合のデータを指し示す）．なお $LoginRate$ のグラフは 1 から値が変化していなかったため省いている．10-190 秒では WC，WoC ともにメールサーバーに対して十分な CPU 資源を配分しており QoS が $Medium$ に保たれているが，200 秒に入るとメールサーバーの負荷が増えたことで WoC の方では CPU 資源が不足状態になり $Medium$ の状態を保てず，時折 $DeliveryTime$ が 40 秒を超える場面も見られた（図 9）．一方，WC では 10-190 秒間は平均 55%程度の CPU 資源が配分していたのに対して，増加したメールサーバーへの負荷により $DeliveryTime$ が高くなったあとでは分配する CPU 量を 63%程度まで増やしている（図 11，図 12）．Java サーバーにおいて、WoC が 200 秒前後で $TransactionTime$，CPU 資源量に対して変化を築いていないのに対して，WC では 200 秒以前は平均 42%与えられていた CPU 資源が 200 秒の後では 34%程度にまで減らされ，最低限 $Medium$ を保つための CPU 量のみが分け与えられている．このように，各ドメインの QoS を見ると，WoC は Java サーバー側に偏った資源分配を行っているのに対して，WC では資源管理システムが個々のドメインが均等な QoS を得られるように CPU 資源を調節しているのが読み取れる．

5.2 計測結果：シナリオ 2

図 13，**図 14**，**図 15** および **図 16** はシナリオ 2 の実験結果を示したグラフである．$LoginRate$ はシナリオ 1 と同様に 1 から変化がなかったためグラフを省いている．シナリオ 2 も最初の 10-190 秒は両ドメインとも $wMedium$ の負荷が与えられ，WC，WoC ともに落ち着いた QoS の移動を取っている．しかし，200 秒に入り Java サーバーの負荷が $wHigh$ に増加したことにより，WoC の側では増加した処理に対応できず $TransactionTime$ が大きく増加している（図 14）．一方，WC では Java サーバーの負荷を $TransactionTime$ 資源管理システムが検知，Java サーバーの QoS を $Medium$ に保つために資源配分を Java サーバーの方に少し傾けている．Java サーバーの CPU 資源が増量したことに対して，メー

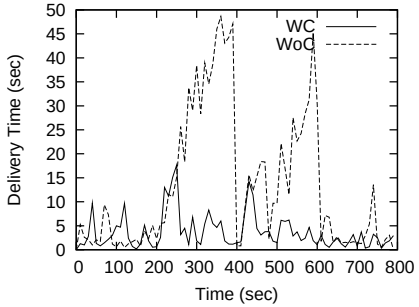

図 9 Scenario1: Delivery Time

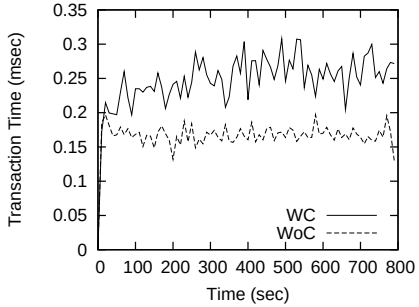

図 10 Scenario1: TransactionTime

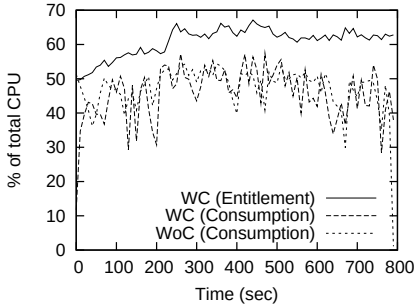

図 11 Scenario1: CPU Entitlement and Consumption on Mail Server

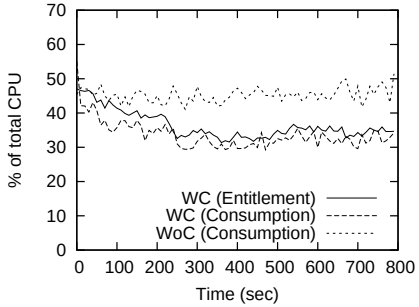

図 12 Scenario1: CPU Environments and Consumption on Java Server

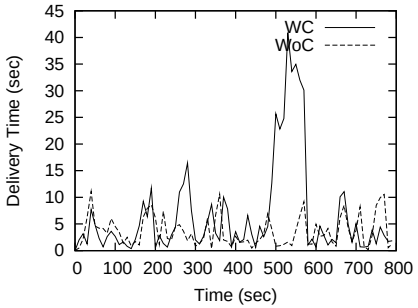

図 13 Scenario2: Delivery Time

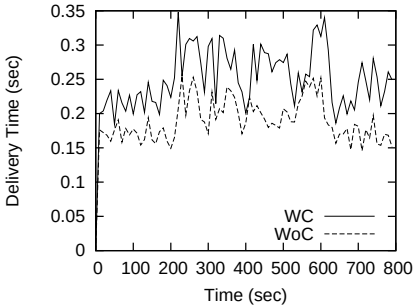

図 14 Scenario2: TransactionTime

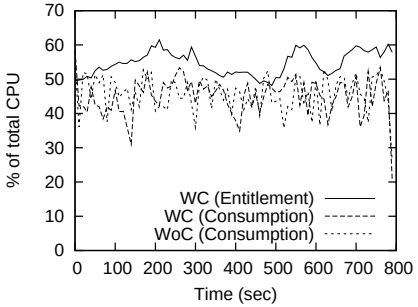

図 15 Scenario2: CPU Entitlement and Consumption on Mail Server

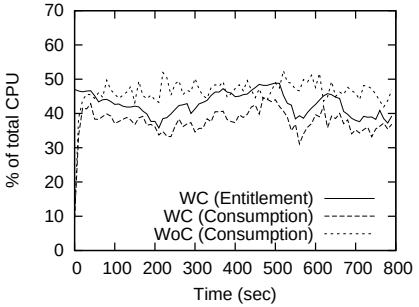

図 16 Scenario2: CPU Environments and Consumption on Java Server

ルサーバーの CPU 資源がその分減少しているのが図 15，図 16 から見て取れる．メールサーバーにおいては WC，WoC の間にほとんど違いがなく，Java サーバーでは WoC が WC に比べて一回り高い QoS を出しているため，総合的に WC，WoC を比較しても，シナリオ 2 では資源管理システムを利用しない場合の方がよい結果を出している．

5.3 計測結果：まとめ

シナリオ 1 では，資源管理システムを利用しなかった場合，メールサーバーと Java サーバーの QoS はそれぞれ $High$，Low の状態を取っており，仮想化システム全体としてみると Java サーバーに偏った運用を行っていた．一方で資源管理システムを動かした場合はメールサーバー，Java サーバーの双方が $Medium$ の状態を維持し，仮想化システムとしても平等

な資源管理を実現していた．資源の使用率は，図 11 と 図 12 で CPU 配分量（*Entitlement*）と CPU 使用量（*Consumption*）が示すように高い資源使用率を維持している．全体の平均を見ても，QoS に関しては *DeliveryTime* は 3.65 秒，*TransactionTime* は 0.25 ミリ秒と *Medium* の範囲にある．また，資源使用率ははそれぞれメールサーバー 75%，Java サーバーが 92%、と 80%に近い値を取っている．

シナリオ 2 では，全体的な性能、個々のドメインの性能においても資源管理システムを利用しない場合の方が良い結果を出している．しかし，資源管理システムを利用した場合であっても，各ドメインの QoS は *Medium* と相違ない範囲で値を動かし，仮想システム全体で見た場合であっても平等な性能を引き出している．また，資源使用率においても 図 15，図 16 から見て取れるように高い使用率を維持している．各値の平均を取ってみると，*DeliveryTime* は 6.6 秒，*TransactionTime* は 0.25 ミリ秒とほとんど *Medium* と相違ない値を持ち，資源使用率はメールサーバーが 82.8%，Java サーバーが 87.6%であり，資源管理システム自体は正しく動作していることは示された．

6. おわりに

本報告では，開発中の仮想化システムにおける動的な資源管理の仕組み，および現在までの計測結果について述べた．

アプリケーションの状態を反映させる資源管理システムにおいては，従来の資源管理が行っている，過剰な資源配分，偏ったシステム運用，を取り除いた効率的な資源管理の実現性が実証された．

しかしその一方で，QoS を一定のレベルに維持する安定性や QoS の変化に対する CPU 資源量の反応速度などまだまだ不安定な部分があり，従来の資源配分法を利用した場合に比べ効率が落ちる場面も見られた．

今後の課題として，上記の問題点の解決を通し仮想化システムにおけるさらなる効率的な資源管理の手法の実装，およびその評価を行う予定である．

参考文献

1) Pradeep Padala, Kang G. Shin, Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, Arif Merchant, Kenneth Salem, "Adaptive Control of Virtualized Resources in Utility Computing Environments", ACM SIGOPS Operating Systems Review, Volume 41, Issue 3 (June 2007).
2) Pradeep Padala, Kai-Yuan Hou, Kang G. Shin, Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, Arif Merchant, "Automated Control of Multiple Virtualized Resources", Proceedings of the 4th ACM European Conference on Computer Systems, pp. 13–26, 2009.
3) Weiming Zhao, Zhenlin Wang, "Dynamic Memory Balancing for Virtual Machines", Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments.
4) SPECmail2001, http://www.spec.org/mail2001
5) SPECjbb2005, http://www.spec.org/jbb2005
6) Jan Jantzen, "Design Of Fuzzy Controllers", Technical University of Denmark, Tech. Rep. no. 98-E-864, 1998.
7) Marcelo Godoy Simoes, "Introduction to Fuzzy Control", http://inside.mines.edu/~msimoes/
8) Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Hrris, Alex Ho, Rolf Neugebauert, Ian Pratt, Andrew Warfield, "Xen and the Art of Virtualization", Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, pp.164–177, 2003.