2003－EVA－7　(5)
2003／8／6

論理分割の自己管理機能による Linux Web アプリケーションの CPU 資源最適化機能の評価

加倉井 宏一*　荻田 光一郎*

メインフレームを利用した Linux Web サーバーの稼動において、論理分割機能の自己最適化機能の評価を行う。プロセッサー論理分割の環境下で、この自己管理機能は、それぞれの論理区画で稼働する Web トランザクションのサービス目標および重要度と CPU 資源の競合に応じて、論理区画に割当てられている CPU 資源の割合を動的に変更する。
テストとその評価を通じ、自己管理機能によって従来の固定的な資源配分では得られなかったトランザクションの重要度に応じた良好な応答時間を得ることが確認できた。

The evaluation of a self-optimizing function: Adjusting system resource of Linux web applications dynamically

Hirokazu Kakurai*　Kohichiro Ogita*

This paper describes that a self-optimizing function adjusting processor resources of logical partitions which are used for distributions of Linux web servers. The test of the function achieve that a dynamic resource re-assignment makes the highest important web transactions taking CPU resources effectively.

１．はじめに

現在、メインフレームの論理分割機能は企業における複数のサーバーを 1 つの筐体で柔軟に稼動させる技術として広く普及している。そして、TCO の観点から複数のサーバーOS を１つの商用大型機上に統合していく企業が年々増加している。さらに、統合対象は単一の OS に限らず、既存ホストの OS に加えて部門サーバーで使用される Linux 等、ますます多くのサーバーOS が統合対象となっている。こうした状況では、これまでのような固定的な CPU 資源の分割では充分にお客様の要件を満たせなくなってきている。業務アプリケーションの持つ応答時間やスループット等、時々刻々変化する目標と重要度に応じて連続して効率的かつ自動的に CPU 資源の分割を最適化したいというニーズは非常に大きい。

また、システム・パラメーターではなく、業務の持つ目標や重要度でパフォーマンス管理する機能は自律型コンピューターとして、ますます注目を浴びている。本研究での自己管理機能はこの自律型コンピューターの重要な一機能として挙げられる[1]。

２．論理分割の自己管理機能

メインフレームの持つ論理分割機能は、CPU 資源を柔軟に分割して複数の OS を 1 つの筐体で稼動させることができる。しかし、プロセッサー筐体の CPU 資源を柔軟に分割できる論理分割機能であるが、従来、CPU 資源分割の割合は固定的であり、区画の活動化時にウェイト値と呼ばれる値の割合で CPU 資源の割当量が決定される。指定の方法によっては、他の論理区画に処理が無ければ、ウェイト値に関わらずプロセッサー筐体にある CPU 資源を使い切ることができる。しかし、全論理区画が 100%の CPU 資源を使用する場合、ウェ

*日本アイ・ビー・エム・システムズ・エンジニアリング(株)
*IBM Japan Systems Engineering Co., Ltd.

イト値の値は正しく守られる。そのため、計画の段階で各論理区画の処理量からウェイト値を算出するのが重要となるが、一般に Web トランザクションのように前もって十分にトランザクション量を見積もることができない処理の場合ウェイト値の算出が非常に困難となる。また、統合される OS の数が増えれば増えるほど、ウェイト値の組み合わせが多くなり算出は複雑になる。

このようなウェイト値の算出が難しいワークロードや環境のため、分割区画内でのシステム資源の自己管理機能がある。基本的な考え方として、この自己管理機能は、ワークロードの稼動するところにシステム資源を動的移動する機能を実現しているといえる。この動的変更は、論理区画で稼働するワークロードの目標と重要度、そして CPU 資源の充足度から判断して行われる[2]。

ワークロードの目標と重要度を元にした資源の動的変更には図 1 にあるような処理フローが用いられている。

予めワークロードに対して、ビジネス上の重要度と目標値を設定しておく。そして、稼動中は管理プログラムが重要度の高い順に目標を満たしていないワークロードを探索する。次に、探索したワークロードの目標を満たしていない主要な資源の要因を発見して、資源調整の候補とする。最後に変更後の結果を予測して良好であれば、資源の調整を行う。このフローが一定の時間間隔で起動され実行される。

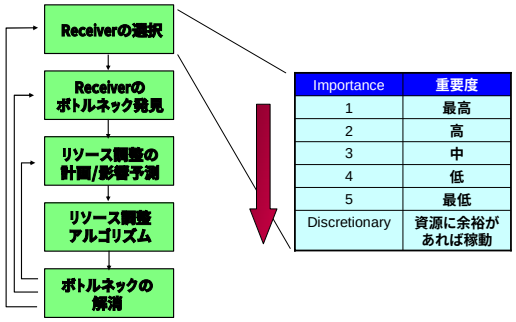

図 1 資源の調整

３．メインフレーム Linux の利用

一方、メインフレーム Linux は PC 上の Linux をメインフレーム･アーキテクチャーにポーティングしたもので、メインフレーム上で稼動する Linux である。

Linux をメインフレームで稼動させることで、次のようなメリットが発生する[3]。

- 40 年以上の長期にわたって培われてきたメインフレーム･ハードウェアの信頼性に関する部分を享受できる。
- メインフレームの論理分割機能に代表される仮想化技術を利用して、新規 Linux の容易な追加やサーバー統合による設備の有効利用などの拡張性が享受できる。
- メインフレーム関連製品を利用した集中管理による運用管理の容易性を享受できる。
- 論理区画間の仮想 LAN 等の利用により、既存の基幹アプリケーションと新しい Linux アプリケーションとの高速かつ柔軟な接続で高い連携性に関する部分を享受できる。

Linux 論理区画も自己管理機能の対象となっており、業務の重要度や目標によって CPU 資源の自己調整を行うことができる。

４．検証システム環境

検証システムの環境について解説する。

４．１． サーバー構成

検証システムのサーバーとして 3000MIPS 相当のメインフレームの論理区画を利用した。16CP を搭載している筐体を利用し、自己管理機能の対象となる 2 つの OS 区画と、管理用の OS の区画を 1 つ、さらに自己管理機能利用の際に必須となるデータ共用機能の区画を 1 つ準備した。

自己管理機能の対象となる 2 つの論理区画で稼働するオペレーティングシステムは Linux で、管理用の OS にはメインフレーム用 OS を構成した。

Linux には SuSE Linux Enterprise Server 7 を使用した。カーネルのレベルは 2.4.7 である。

管理用 OS は IBM 製メインフレーム OS の z/OS V1.4 である。

2 つの Linux 区画のウェイト値は検証ケースによって変更するが、合計を 400 として指定する。自己管理機能で稼動する場合は、最小は 1、最大は 399 までの値が割振ることができるよう指定した。管理用 OS 区画のウェイト値は最小と最大共に 600 を指定することで、この値は常に固定になる。

論理 CP の数は Linux 区画で 2 個ずつ、管理用区画では 6 個を定義した。この検証テスト以外で利用される CP はそのほかの区画に固定的に利用されて、検証テストに影響がないようにした。

初期設定値およびシステム構成は図 2 の通りである。

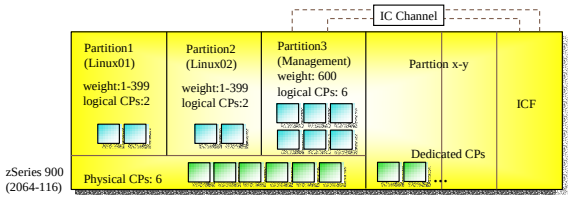

図 2 システム環境

2 つの Linux には Apache web サーバーを稼動させた。以降、Linux01 と Linux02 という名前を付け説明する。

４．２． トランザクション生成ツール

クライアント側のトランザクション生成ツールとして Web Performance Tools を利用した[4]。

Linux01 へのトランザクションは端末 2 台をシミュレートし、合計で 300 トランザクションを生成する。一方、Linux02 へのトランザクションは端末 5 台シミュレートし、合計で 1000 トランザクションを生成する。

トランザクションは Linux01 と Linux02 共に共通で、CGI を利用したファイル検索のトランザクションである。

端末台数および総トランザクション数に相違を持たせたのは、恒常的に Web のトランザクションがある状態で重要度の高いトランザクションが短期間に発生することを想定したからである。実業務での想定として、例えば Web 予約システムにて新製品の発売初日に急激に新製品向けのトランザクションが伸びることを考えてのことである。

５．検証のシナリオ

トランザクション発生の条件は各シナリオでほぼ同様として、システムの環境設定によってどのようにトランザクションの振る舞いが変化するか検証する。トランザクション発生の条件は、先ず Linux02 向けのトランザクション処理を開始して、約 3 分後に Linux01 向けのトランザクションを開始する。

５．１． シナリオ 1

Linux01、Linux02 共に論理区画のウェイト値を 200 ずつの固定とした。既存のメインフレーム環境で Linux を追加する時、自己管理機能を使用しない場合の最も一般的な設定であるといえる。

５．２． シナリオ 2

Linux01 と Linux02 にトランザクション目標を設定した。Linux01 には重要度=1、実行速度=50 を指定した。Linux02 には重要度=5、実行速度=20 を指定した。論理区画の自己管理機能を使用し、重要トランザクションの稼動する Linux01 論理区画に重要度が高く、スループットの高い指定を行っている。

５．３． シナリオ 3

Linux01 の論理区画ウェイト値を 350、Linux02 の論理区画ウェイト値を 50 とした。Linux01 に多く資源配分が行われるように固定的なウェイト値を設定した。自己管理機能を使用しない場合で、予め重要度の高い論理区画にウェイト値の多くを割振ることを想定している。

５．４． シナリオ 4

Linux01 と Linux02 に目標を設定した。Linux01 には重要度=1、実行速度=80 を指定した。Linux02 には重要度=5、実行速度=20 を指定した。Linux01 に関して、シナリオ 2 よりも高いスループットを目標とした。

５．５． システム設定のまとめ

各シナリオでのシステム設定をまとめると次の通り。

表 1 システム設定サマリー

シナリオ	Linux	論理区画ウェイト値	目標	
			重要度	実行速度
1	1	200(固定)	n/a	
	2	200(固定)		
2	1	200(初期値)	1	50
	2	200(初期値)	5	20
3	1	350(固定)	n/a	
	2	50(固定)		
4	1	200(初期値)	1	80
	2	200(初期値)	5	20

６．検証結果

検証の結果を次にまとめる。各シナリオにおいて、1 分経過毎のトランザクション数、平均応答時間、論理区画ウェイト値、平均 CPU 使用率を取得しグラフにまとめた。

６．１． シナリオ 1

Linux01 と Linux02 は同じウェイト値を指定してあるので、CPU 資源の割振り量が等しくなり、1 分間に同数のトランザクションを実行することができる。しかし、トランザクション発生端末数が異なるので Linux01 の応答時間の方が短い。その Linux01 の平均応答時間は 2133 ミリ秒であった。

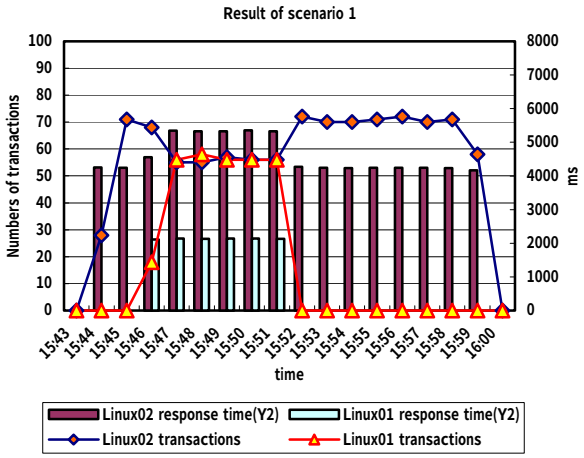

図 3 シナリオ 1 の TRX 数、応答時間

ウェイト値が同値であるので競合が発生した時間帯、ほぼ同じ CPU 使用率となっていることがわかる。

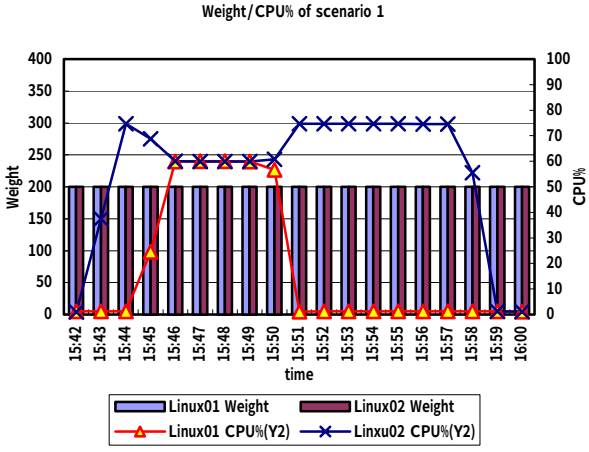

図 4 シナリオ 1 のウェイト値、CPU 使用率

6.2. シナリオ 2

Linux01 には高く、Linux02 には低く目標を設定しているため、Linux01 のトランザクションが開始した直後から徐々に Linux01 に CPU 資源がわたり、応答時間の短縮とトランザクション数の増加が見られるようになった。目標に従って制御プログラムが徐々にウェイト値を変更していったのがわかる。

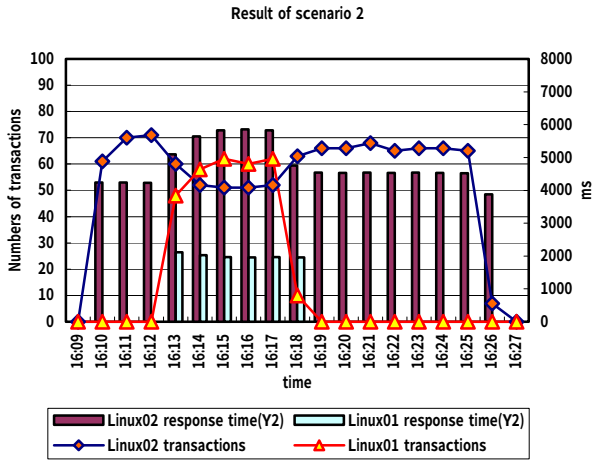

図 5 シナリオ 2 の TRX 数、応答時間

ウェイト値の調整によって、Linux01 と Linux02 の競合発生の時間帯、Linux01 に多く CPU 資源が渡った事がわかる。

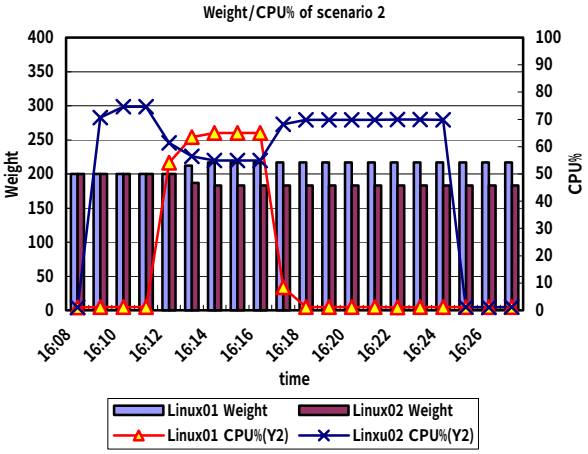

図 6 シナリオ 2 のウェイト値、CPU 使用率

6.3. シナリオ 3

Linux01 に固定的に大きくウェイト値を与えているため、Linux01 のトランザクションは大幅に短い応答時間で処理が行われている。しかし、Linux02 のトランザクションは恒常的に CPU 不足が発生して、応答時間が悪化している。

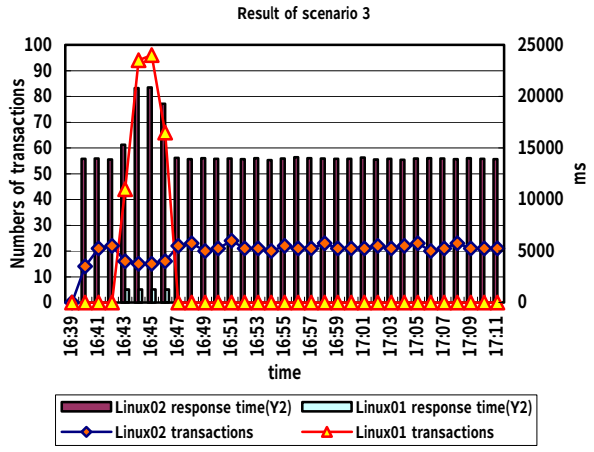

図 7 シナリオ 3 の TRX 数、応答時間(縮尺が異なる)

Linux02 のトランザクションに CPU 資源がわたらず、処理に多くの時間が費やされていることがわかる。シナリオ 3 では図 7図 8のグラフにある範囲内では 1000 件の Linux02 の処理は終了していない。

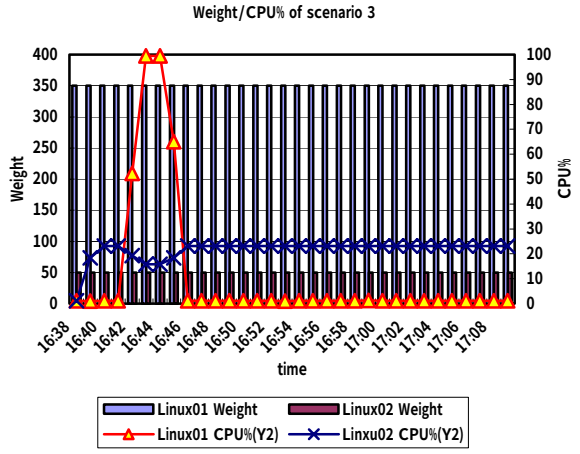

図 8 シナリオ 3 のウェイト値、CPU 使用率

6.4. シナリオ 4

シナリオ 2 と比較してさらに Linux01 の応答時間が短くなって、Linux01 の 1 分間当たりのトランザクション処理数も大きくなっている。

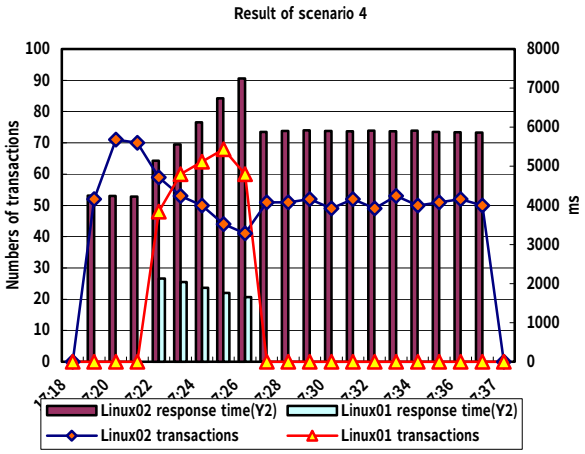

図 9 シナリオ 4 の TRX 数、応答時間

目標に応じて Linux02 の論理区画に CPU 資源が多く渡っている事がわかる。

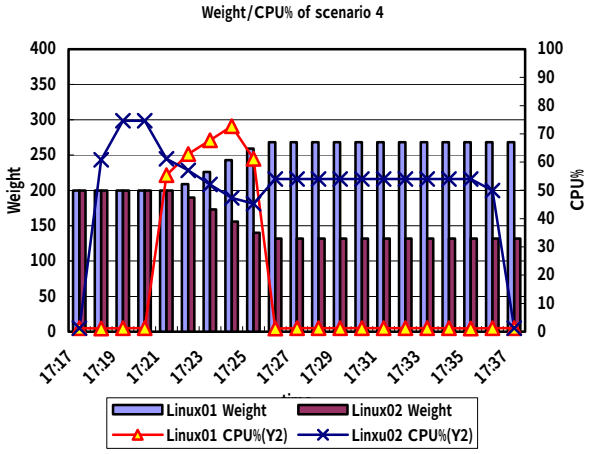

図 10 シナリオ 4 のウェイト値、CPU 使用率

6.5. 検証結果のまとめ

各シナリオでの応答時間、および最初のトランザクション始まってから終了するまでの時間を表 2 にまとめた。

表 2 結果サマリー

シナリオ	Linux	平均応答時間(ミリ秒) (Linux01=端末2台、Linux02=端末5台)	処理実行時間(秒) (Linux01=300件、Linux02=1000件)
1	1	2133	320
	2	4564	914
2	1	2002	301
	2	4777	956
3	1	1267	190
	2	14435	2848
4	1	1881	282
	2	5616	1124

7. 評価

シナリオ 1 を基準と考慮して分析を行うと、ワークロードへの目標を設定したシナリオ 2 では重要度の高いトランザクションに対する応答時間の

改善が見られる。目標を設定しておけば、必要に応じて重要トランザクションへの資源配分が行われる。

シナリオ 1 よりも積極的に重要論理区画へウェイト値を割振ったシナリオ 3 では Linux01 側の重要トランザクションの応答時間が大幅に良くなった。しかし、Linux02 のトランザクションで恒常的に応答時間が悪化した。固定的なウェイト値の設定の考慮の必要な点といえる。

また、シナリオ 4 にあるようにシナリオ 2 よりも積極的な目標設定した場合さらに重要トランザクションで応答時間の短縮化を図ることができた。

注目すべきなのは、ウェイト値のようなシステム・パラメーターではなく、トランザクションの持つビジネス上の重要度や目標によってシステムが自己最適化したことである。従来であれば、システム管理者がシナリオ 1 やシナリオ 3 のような設定を試行錯誤で決定しなければならないものが、エンドユーザーとのサービスレベルで一意にシステム設定が決定することができるようになった。

また、拡張性や変化への柔軟性を考えると、自己管理機能の優位性がさらに際立つ。例えば、ビジネス環境の変化によって、さらに Linux システムを追加しなければならないケースが発生した場合、固定的なウェイト値の設定ではより複雑度が増すことになり、さらにシステム管理者を悩ます事態になることが簡単に予想される。ビジネスの重要度による目標設定であれば、サーバーの数はそのままで新規にアプリケーションを増やすときであっても、アプリケーションはそのままでサーバーを増やしたいときであっても、柔軟に対応する(もしくは設定を変更しないでおく)ことが可能である。

８．今後の課題

検証によって論理分割の自己管理機能が良好に働くことが確認できたが、さらに利用者にとって、魅力的な機能になるよう今後の課題を記述する。

先ず、現在は Linux 区画全体のスループットが目標として扱われている。今後は Linux 区画で稼動する Web トランザクションの応答時間での目標管理が必要になると考える。一部のメインフレーム OS ではトランザクションレベルでの応答時間の目標設定も可能であるが、オープンな Linux 環境でのトランザクションレベルでの目標設定は今後の課題となる。

次にトランザクション連携での課題もある。現在の実システム環境において、１つの Linux システムだけでトランザクション処理が完結することはない。バックエンドの DB やアプリケーション・サーバー同士の連携が必ずあり、ユーザーから見たエンド・トゥ・エンドの応答時間による管理というのが重要となる。現在の自己管理機能の設定ではサーバー単位の目標設定である。よって、トランザクションレベルの目標設定も含めた、エンド・トゥ・エンドの目標設定が必要になるであろう。

９．おわりに

サービスレベルを設定すれば、サービスレベルの維持を自動的に図る自律型コンピューティングは今後ますます重要になってくると考えられる。本研究では、少ないサーバーで限定された環境であったが、実際のお客様の環境では、もっと多量のサーバーで、インターネットなどのオープンな環境へ接続された複雑性の著しい環境が主である。そういった環境にこそ、自律型コンピューティングの更なる発展が望まれる。よって、本研究で限定された環境ながらも自律型コンピューテュングの有効性が証明できたことの意味は大きいと考える。

１０．　参考文献

[1]荻田光一郎、オートノミック(自律型)コンピューティングの最新動向、電子情報通信学会、ディペンダブルコンピューティング研究会、2003

[2]加倉井宏一他、商用大型機での分割区画内のシステム資源の自己管理機能の評価、情報処理学会、システム評価研究報告、No.4、2002

[3] 坂爪淳力他、Linux/390 活用への第一歩、日本ガイドシェア研究論文、SP 部会、2001

[4] Web Performance Tools、http://www.alphaworks.ibm.com/tech/wptools、2003.7.3