2005－DSM－39（7）
2005／10／14

ブロードバンド・プラットフォームにおける異機種混在システムのモニタリング技術と広域負荷分散への応用

山根敏志, 美園和久, 木村順子, 藤本洋平

株式会社日本 IBM

概 要

現在の基幹業務システムは大規模・複雑化した異機種混在 Web システムへと発展し、その管理運用コストが増大している。そこで、本研究では、エージェント技術を活用した異機種混在システムにおける統一的かつ効率的モニタリング技術を提案する。さらにこのモニタリング技術を複数ドメイン間の自律的広域負荷分散へ応用した結果について報告する。

A monitoring technique for web systems with heterogeneous components and its application to wide load balancing

Toshiyuki Yamane, Kazuhisa Misono, Junko Kimura, Yohhei Fujimoto

IBM Japan, Ltd.

Abstract

The present core coporate systems have evolved into large complex Web systems consisting of a lot of heterogeneous components and therefore the operational and management cost of them are increasing. The present paper shows unified and effective monitoring techniques of such complicated Web systems based on agent technology. Furthermore, this monitoring techniques are applied to load balancing over multiple domains.

1　はじめに

現在のブロードバンドプラット・フォームに基づく基幹業務システムにおけるメインフレームから分散システムへの急速な移行により、低コストで多機能なシステムの構築が容易になった。分散系システムは企業の基幹業務にも浸透していることから、メインフレームと同等の信頼性・高可用性が要求されるが、その反面システムを構成する要素の数は加速度的に増加しているため、それらのモニタリングは一段と複雑になってきている。

モニタリングの複雑さを増大させる要因の一つとして、通常、分散系システムは異なるオペレーティング・システム を持つサーバーで構成されることが挙げられる。たとえば Linux、Unix、Windows などの OS 毎に採取可能なモニタリング情報（ CPU、メモリーなどの使用状況）は、その種類、出力フォーマット、取得方法に違いがあり、従来型の方法ではこれらのデータ取得にはプラットフォームごとに別々の仕組みを作成する必要がある。また、正確なステータスを把握しながらも、モニタリングを行っているサーバーの主機能に影響を及ぼさないようにするには、モニタリングのタイミングの調整が必要である。

これらの課題を解決するために本論文では、

まずOSやミドルウェアの違いによって生じるモニタリング情報の違いを吸収し、統一的にデータを取得するためのインターフェース定義を行う。続いて、インターフェイスに基づいてモニタリングを行うためのエージェント [2] を導入し、そのアーキテクチャーおよび効率的なモニタリング手法を提案する。さらに、本論文で提案するモニタリング手法を異機種混合の複数ドメインに跨る広域負荷分散へ応用した例について報告する。

2 エージェント技術による異機種混合システムの効率的モニタリング

2.1 異機種混合システムに対する統一的モニタリング情報

システムのモニタリングを行う際、どのような情報をモニタリングするのかは目的によってまちまちであるが、一般的にマシンの性能を決定する基礎情報である、CPU 情報、メモリ情報、ディスク I/O 情報、ネットワーク情報のモニタリングは必須である。しかし、異なる OS から構成される分散系システムをモニタリングする際の課題点として、各種 OS やサーバー用途に応じてリソース状況や動作状況のデータ取得方法、出力結果や取得できるパフォーマンス項目に違いがあることが挙げられる。例えば、同じ CPU の情報を取得するにしても Windows と Unix ではその方法が全く異なり、また同じ Unix 系システムの中でも発行するコマンドは異なる場合がある。さらに、サーバーの種類によってもモニタリングする項目は大きく異なる。その結果、各プラットフォームに応じてモニタリング用のモジュールをカスタマイズせねばならず、従来型の手法ではその他モジュールとの結合が密になってしまい プラットフォーム間でのポータビリティを損なう原因となっていた。この問題の根本的解決にはデータ取得方法・出力結果の違いを吸収するためのレイヤーを作り、統一的にモニタリング情報を収集する機能を実現する必要がある。

そこで、まずモニタリングする対象（プロセス、ミドルウェアの種類）、およびその対象ごとのパフォーマンス指標に対しセマンティックな定義を行い、OS やミドルウェアが異なってもデータが統一的に取得できるようなデータ取得インターフェースの定義を行う。それは、システムへのリクエスト対象（プロセス・タイプ、特性）、その対象ごとのパフォーマンス指標（スループット、CPU 使用率）という階層構造をもったモニタリング情報である。定義名のネーミングは、⟨ サーバー名 ⟩.⟨ モニタリング対象 ⟩.⟨ 特性 1⟩.⟨ 特性 2⟩... というルールに基づく。たとえば、Node.CPU.Usage はノードの CPU 使用率 (%) を表し、AppServ.WebContainer.Thread.number はアプリケーション・サーバーの Web コンテナ内のアクティブスレッド数を表す、といった具合である。

2.2 モニタリング・エージェントのアーキテクチャー

異機種混在システムから必要なモニタリング情報を取得するために、それぞれの異機種サーバーからパフォーマンス・データを採取する場合は、膨大な情報量のデータ転送、解析処理をそれぞれの OS に応じて行う必要が生じる。この状況は管理系ネットワークに与えるデータ転送負荷およびモデリング情報を分析・予測するなどの自律制御のためのサーバーに対する処理負荷を必要以上に大きくしてしまう可能性がある。以上の問題を解決するために、サーバーごとにエージェントを割り当て、異機種混在のサーバーによる違いを吸収すると共に、サーバーの動作状況に応じてデータサンプリングの調整やパフォーマンス・データの標準形式へ変換などもそれぞれのサーバーに割り当てたエージェントが処理することによって、モニタリング情報収集処理の負荷軽減、効率化を実現する。モニタリング・エージェントの存在によって、プラットフォームごとに個別にシステムの動作状況モ

ニタリングする必要がなくなり、将来新しいプラットフォームが出現してもエージェントの生成、修正で対処できるようになるので、拡張性のあるモニタリング技術となる。

今回提案するアーキテクチャーは、各サーバーの ICT 資源・動作状況をモニタリングするモニタリング・エージェントと、各モニタリング・エージェント全体を統括してデータを収集・加工するドメイン・エージェントの 2 つのコンポーネントから構成される。以下で、図 1 に示すモニタリング・エージェントおよびドメイン・エージェントのアーキテクチャーの詳細に関して記述する。

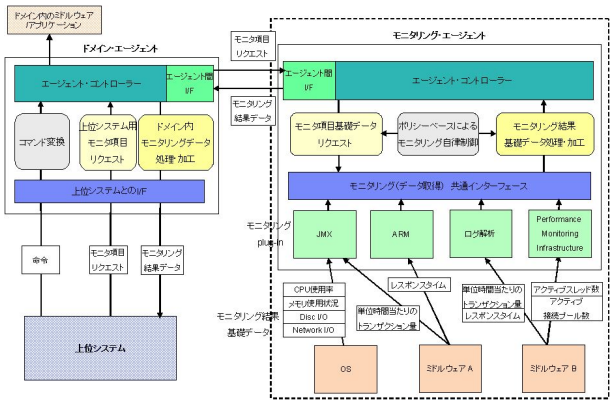


図 1：モニタリング・エージェントのアーキテクチャ

以下、各モニタリング・エージェントのコンポーネントに関して記述する。

1. モニタリング・プラグイン
   モニタリング・プラグインは、対象となる各種 OS やミドルウェアのリソース使用状況や動作状況を実際にモニタリングするモジュールである。このモジュールは Java によるアプリケーション監視のインターフェイスである JMX (Java Monitoring eXtention) や、ミドルウェア内で発生するトランザクションの応答時間を測定するための手法である ARM（Application Response time Monitoring）などが考えられる。これらの手法に対応していない OS, ミドルウェアやモニタリング項目が存在する場合は、OS やミドルウェアが独自に提供しているモニタリング用のインターフェースを用いたり、ログの出力からモニタリング情報の解析を行う方法（ログ解析）などが考えられる。

2. モニタリング（データ取得）共通インターフェース
   このインターフェースは先に述べた OS やミドルウェアごとのデータ取得手法に依存した作りにならざるを得ない各モニタリング・プラグインの違いを吸収して、上層のコンポーネントからのリクエストにはその違いを見せないようにするためのものである。ここで定義したインターフェースを用いて各種データが統一的に取得できるようにすることで、取得データやモニタリング・プラグインの手法、実装の追加・修正を容易にすると同時に、それらが変更されても上部コンポーネントにはその変更による影響を与えないように実現できる。

3. モニター項目基礎データリクエスト
   モニタリング共通インターフェースを用いて実際にモニタリングデータの取得を行う部分である。どのデータをどれくらいのサンプリング間隔で取得するかという指令をモニタリング自律制御部分から受け取り、それに基づいて動作する。

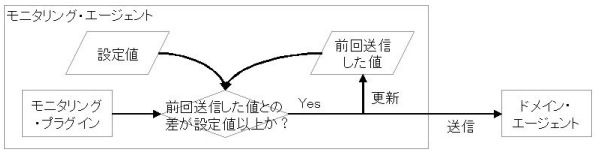
4. ドメイン・エージェント
   ドメイン・エージェントは各モニタリング・エージェントからデータを収集、集計・加工し、ドメイン全体のリソースの状況を表すデータを保持する。さらに、上位システムにデータを送信したり、上位システムからの命令を受けてドメイン全体への指令を実行する。

2.3 ポリシー・ベースによるモニタリング自律制御

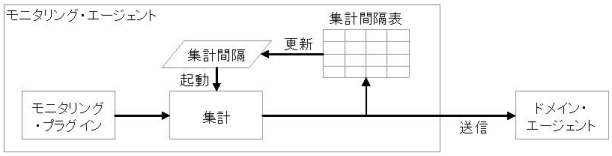
異機種混在システムの構成要素が増加するにつれて、モニタリングすべき項目や数も増加している。一般的には項目が多くかつ頻度高くモニタリングするほどシステムの状態を高精度に把握できるが、従来のモニタリングでは単一的なサンプリング間隔でのデータのモニタリングと閾値判断のみの考慮が主流であるため、膨大な情報量のデータ転送のために、サーバーに対する処理負荷を必要以上に大きくしてしまう可能性がある。そこで、取得されたデータを判断してモニタリングの間隔を最適化したり、状況に応じてどのデータを送信すべきかを選択するようなインテリジェントな機能をモニタリング・エージェントに付与することを考える。これによりデータ採取の負荷をできるだけ低減し、モニタリング機能自体がオーバーヘッドにならないことを保証できる。今回は次の 2 点からモニタリングの自律制御を行った。

ノード間データ送信間隔の調整：サーバーの状態を正確にモニタリングするには集計間隔を短くすればよい。しかし、集計間隔を短くすることは送信回数の増加を招き、ネットワーク負荷を増加させる。そこで、監視対象サーバーでの状態がドメイン・エージェントに伝わる正確さや迅速性にある一定の許容幅を持たせて、ドメイン・エージェントの把握している状態から実際の状態が大きく離れていなければ、情報の更新を行わなくてもある程度の正確さは保持できることになる。このような観点から図 2 上に示すような構成を考える。これは取得したデータを、履歴として持っている前データと比較し、差異が設定した許容範囲内であれば送信行わないというものである。このようにすれば、ネットワーク転送やログ出力等のデータ総量を抑制することができ、システムに対するモニタリングのオーバーヘッドを押さえることができる。

データ・モニタリング間隔の調整: モニタリングの目的がサービス・レベルの維持である場合、最も重要なのはサービス・レベルを満たせなくなったことを、できるだけ迅速にドメイン・エージェントが認識することである。そのような観点からモニタリングの効率化を考慮すると、サービス・レベルを満たしている状況では、サービス・レベル違反が発生した場合をなるべく早く検知するために測定・集計間隔を短く設定するのが有効である。また、一旦サービス・レベルを満たせない状況が発生してからは、測定・集計間隔を比較的長めに設定することでシステムにかかる余計な負荷を減らすという方法が有効である。以上のような観点から、図 2 下に示す構成のように、集計間隔表から次に集計を行うタイミングを決定する以下の構成を考えると、サービス・レベルの遵守状況に応じた自律的なモニタリングが可能となる。



ノード間データ送信間隔の調整



データモニタリング間隔の調整

図 2 ：自律・適応的モニタリング手法

3 自律的広域負荷分散への応用

この節では前節において述べたエージェント技術に基づくモニタリング技術を異機種混合のシステムからなる複数のドメイン間での広域負荷分散 [3] へ応用した例について述べる。具体的には、複数の優先順位の異なるアプリケーションが存在する状況で、最優先アプリケーションの応答時間が悪化してきたとき、低優先度のアプリケーションへのリクエストを余剰のリソースをもつ他ドメインに転送することで最優先アプリケーションの応答時間 SLA(Service Level Agreement) を維持する、というものである。

3.1 負荷分散システムの概要

実証実験では実験環境として図３のようにLinux, Windows, AIX(IBM Unix) の異機種システムが混在した２つのドメイン (ドメインAおよびドメインB) からなる Web システムを構築し、この実験環境に対してピーク性のあるWeb アプリケーションを２つのドメインで同時に実行させる。各ドメイン内のロード・バランサーは、ドメイン内の負荷分散を実行する「シングルドメイン用」と、複数ドメイン間の負荷分散を実行する「マルチドメイン用」の２段構成とする。

そして、ドメインA、ドメインBに対し、研究開発を行った各コンポーネントを図??に示されるように配置する。統括監視システムは各ドメインに１つ配置され、ドメイン内のアプリケーションに関する情報を一括管理する。統括監視システムはドメイン・エージェントから取得した自ドメインのパフォーマンス情報をRMI/IIOP (CORBA) を用いたリモート通信によってサービスとして他ドメインへ公開する役割を持つ。統括監視システムは内部に待ち行列モデル [?],[1] を保持し、スループットや応答時間などのパフォーマンス情報に基づいて他ドメインへのリクエストの負荷分散割合を計算する。モニタリング・エージェントに関しては今回の実装では、Web サーバーが出力するアクセス・ログを解析し、アプリケーションに対するHTTP リクエストの平均応答時間および各アプリケーションの一定時間あたりのリクエスト到着率/スループットを取得する方法を取った。この際のモニタリングの間隔はあらかじめ設定されたサービス・レベル・アグリーメント（SLA）によって判断され、集計・加工されたデータがSLA 違反を示した場合、モニタリングの間隔が調整される。そして、ドメイン・エージェントが自ドメイン内のパフォーマンス情報を一括収集し、集計した情報を統括監視システムに報告する。統括監視システムからの負荷分散割合の変更要求があった際、自ドメインのロードバランサーに対してドメイン間の負荷分散割合の命令を発行する。

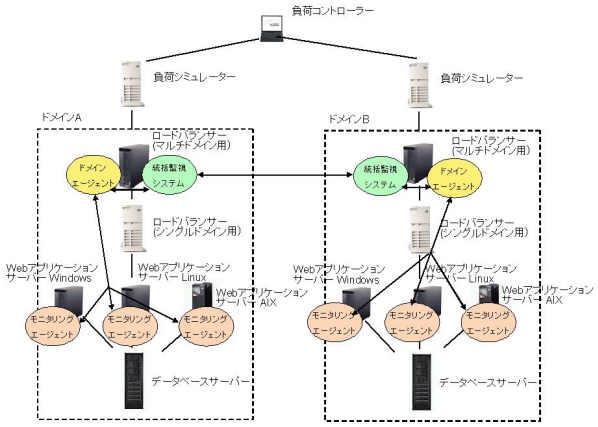


図３：自律的負荷分散システムの全体像

3.2 システム動作の概略

実証実験用に、高優先度と低優先度の次の２種類の典型的なユーザー・アプリケーションを選定した。優先度が高いアプリケーションとしてオンライン証券取引 (以下アプリ A と呼ぶ) を選定し、アプリ A の目標平均応答時間 SLA を２秒と設定した。また、優先度が低いアプリケーションとしてオンライン・ショッピング (以下アプリ B と呼ぶ) を選定した。

図４はドメイン A における統括監視システムが取得したアプリ A、アプリ B のそれぞれのリクエスト到着率、応答時間、およびドメイン B に対する負荷転送率（負荷分散割合）の時間変化を表した図である。この実験においてはアプリケーション A のユーザー数を 10 に固定した上で、アプリケーション B のユーザー数を 0 $\longrightarrow$ 20 $\longrightarrow$ 1 と変化させた。この測定ではアプリケーション B のユーザー数が 20 の場合にアプリケーション A の平均応答時間が２秒を超え、SLA 違反を起こす。すると、統括監視システムが稼動を始め、ドメインBへ負荷分散を始める。負荷分散の割合はおよそ 65%をドメインBへ割り振るという状態で安定し、アプリケーション A の SLA は遵守される。そして、アプリケーション B のユーザー数を 1 に減少させたところ、負荷分散の割合もそれに伴って減少し、最終的に初期状態の 0%に復帰する。この間、アプリケーション A の SLA は遵守されて

いる。

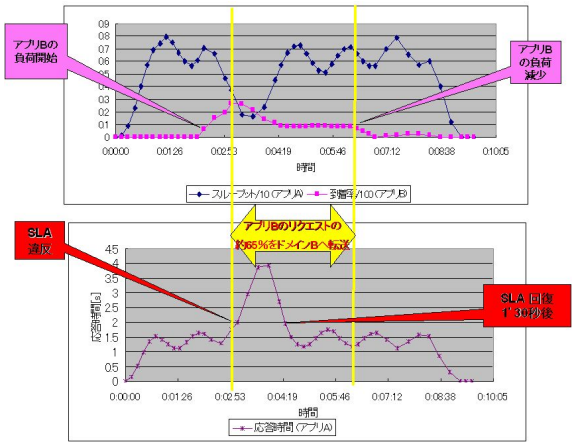


図４：全測定データの時間変移

本論文で提案したモニタリング技術のオーバーヘッドについて調べるため、モニタリング・エージェントを停止させた状態において負荷を段階的に増大させた場合の CPU 使用率とスループット（単位時間当たりのシナリオ実行数）と、モニタリング・エージェントを起動させた状態においての CPU 使用率とスループットを測定し、両者の比較を行った。エージェント起動あり・なしで CPU 使用率、スループットいずれにも大差はなく、エージェント起動による影響は CPU 使用率で +5% 以内、スループットで -1% 以内に収まっていた。従って、統括監視システムが与えるオーバーヘッドは、無視できる程度に小さいといえる。

本論文で述べたモニタリング技術によって異機種混合システムの状態が抽象度の高いジェネリックな少数のデータ (応答時間、スループットなど) に集約されたため、上位のインテリジェント・モジュール (統括監視システム) はシステムの詳細に依存しない抽象度の高いロジックで負荷分散を行うことができるのである。

4　まとめ

本論文では、Web システムの特性、それらを構成するミドルウェアの特徴に基づき、各種 OS、ミドルウェア製品に対する統一的なモニタリング項目と、それらデータ取得のための統一的インターフェースを定義した。また、従来型のモニタリング手法で課題であった異機種混在システムにおける統一的なデータ取得と効率的なモニタリング手法の確立を目的として、エージェント技術によるモニタリング用コンポーネントのアーキテクチャーを提案した。また、それに基づいてアプリケーションのサービスレベル遵守をモニタリングする実装を行い、システムに負荷を与えない効率的なモニタリング方法についての考察を行った。さらに、本手法を異機種が混在する複数ドメイン間での負荷分散へ応用し、実証実験によって負荷が増大していったときにシステムに深刻なオーバーヘッドをもたらすことなく、負荷分散が可能となることを示した。

参考文献

[1] Raj Jain, The Art of Computer Systems Performance Analysis, Techniques for Experimental Design, Measurement, Simulation and Modelling, Wiley, 1991.

[2] Tom De Wolf and Tom Holvoet, ”Towards autonomic computing, agent-based modelling, dynamical systems analysis, and decentralised control”, Workshop on Autonomic Computing Principles and Architectures (AUCOPA), 2003.

[3] 馬場始三、山口英、”DNS を用いた広域負荷分散の実装、” 研究報告 - 分散システム／インターネット運用技術　No.036, 1998.