



キャンドル コンピュータ レポート

Vol.18 2004年12月号

<http://www-6.ibm.com/jp/software/zseries/newsletter/ccr2/>

日本アイ・ビー・エム株式会社

ソフトウェア事業キャンドル事業部

目次

Feature

DB2 の Workload Manager: 最適な目標の設定 - 第 2 部	3
アプリケーション環境	3
DDF と設定目標	4
DB2 サブシステム アドレス スペースの目標	5
WLM の将来	5
結論	6

テクノロジーに関するヒント

埋め込まれたシチュエーションのオーバーヘッドを伴わない、時間によるしきい値の制御方法	7
埋め込まれシチュエーションの一般的な使用法	7
埋め込まれたシチュエーションの別の使用方法	8
埋め込まれたシチュエーションの欠点	8
時間別のしきい値に対する推奨アプローチ	8
エージェントおよび CMS のオーバーヘッドを低減するポリシー作成手順	9
結論	10

DB2 の Workload Manager:最適な目標の設定 - 第 2 部

Ed Woods

IBM/システムエンジニア

現在の大規模システム環境において、ワークロード マネージャー (WLM) は、非常に多様なアプリケーションおよびリソースを、一貫性をもって、タイムリーに管理するための重要なメカニズムになっている。洗練され時間節約に役立つ z/OS® のコンポーネントである WLM は、MVS 上で実行しているワークロードに優先順位を付け、そのワークロードを利用可能なリソースと組み合わせる。

WLM がどのようにうまく機能するかは、多くの場合、ユーザーがどれだけ現実的で達成可能な目標を設定できるかに関わっている。第 1 部では、DB2® と連携した WLM を紹介し、WLM の重要用語を取り上げ、目標、ワークロード、エンクレーブのほか、スレッドおよびアプリケーションのオプションについて詳細に説明した。

この第 2 部では引き続き、アプリケーション環境、DDF および DB2 サブシステム アドレス スペースの目標、および WLM の将来について説明する。

アプリケーション環境

ストアード プロシジャーとユーザー定義関数 (UDF) は、WLM が定義し提供するインフラストラクチャを使用する。また、WLM アプリケーション環境は、WLM 管理のストアード プロシジャー アドレス スペースを定義し、管理する。アプリケーション環境は、WLM サービス ポリシーの他のコンポーネントを定義するために使用されるのと同じ WLM ISPF ダイアログで定義される。

アプリケーション環境は、ストアード プロシジャーが実行される環境のカスタマイズに対応している。この環境は、JCL、アドレス スペースの数、およびそのアドレス スペースの特性を制御する。

アプリケーションが、SQL CALL を発行してストアード プロシジャーを呼び出すと、DB2 は要求を WLM のキューに入れる。WLM は、ストアード プロシジャー要求キューを管理し、WLM 内で定義された適切なアプリケーション環境のアドレス スペースへそれらを送信する。

続いて WLM は、プロシジャーが実行されるタスク制御ブロック (TCB) を探す。通常、各アドレス スペースには複数の TCB が定義される。そして、ストアード プロシジャー アドレス スペースは、アプリケーション モジュールを DASD からメモリにロードして(常駐していなかった場合)、実行を開始する。

WLM は、複数の DB2 ストアド プロシジャー アドレス スペースの管理に対応している。複数の JCL プロシジャーをサポートしており、1 つの JCL プロシジャーを複数回実行することもできる。ストアド プロシジャーと UDF が呼び出されると、WLM はワークロードの要求を分析し、状況に応じてストアド プロシジャー アドレス スペースを追加して開始する。

WLM では、z/OS に定義されたサービス クラスとアプリケーション環境の組み合わせごとに個別のキューを管理する。ワークロードを異なるサーバー アドレス スペースに分配するには、別々のアプリケーション環境を定義する必要がある。各アプリケーション環境内には、サービス クラスごとに個別のキューがある。

このアプローチの利点は、ストアド プロシジャー処理能力を大幅に拡張できるということにある。このアプローチでは、ストアド プロシジャーの優先順位設定と制御を、かなり細かく行うこともできる。つまり、必要に応じて、一部のストアド プロシジャーと UDF の優先順位を高くしたり、反対に他のストアド プロシジャーの優先度を低くして実行したりできるのである。

DDF と設定目標

WLM サービス クラス定義では、DB2 DDF ワークの分類と優先順位設定を行うことができる。これを実現するには、WLM ISPF ダイアログで、サブシステム タイプ DDF に対し、ワークをサービス クラスに割り当てる。スレッド作成やセキュリティ チェックなど、理想的には迅速性が要求される作業を、制御する内部タスクが存在するので、DDF アドレス スペースには常に高い優先順位を付ける必要がある。DDF アドレス スペースは、DBM1 などの他の DB2 サブシステム アドレス スペースと同じサービス クラスに割り当てることができる。

DDF アドレス スペース実行優先順位よりも低い、DDF エンクレーブ SRB の MVS 実行優先順位を選択する。これにより、DDF 分散スレッド SRB の前に、DDF TCB サービス タスクがディスパッチされる。エンクレーブのサービス クラス定義の作成プロセスを単純化するには、サービス クラス定義パネル内のワイルドカード機能を使用できる。

最初は単純に、必要に応じて拡張していく。サービス クラスが多すぎると、混乱や非現実的な目標を招くことがある。WLM の一般的な推奨事項として、サービス クラスの数(およびさらに重要なのは期間の数)を比較的小さな管理可能な数に留めておくことが挙げられる。目標が多すぎると、WLM は優先順位の低い目標までは達成できなくなる。DB2 Connect™および WebSphere®アプリケーションから DB2 に渡されるアカウントやその他の情報を活用する。たとえば、アプリケーションの実行名を渡して、サービス クラス定義パネルで使用する。

DDF ワークロードによっては、速度目標とレスポンス タイム目標の両方を要求することがある。こ

の場合、DDF アドレス スペース自体の速度目標に対し、DDF ワークロード項目のレスポンス タイム目標という図式になる。DDF ワークロードは、依然として、サブシステム タイプ DDF ワークロード ルールを使用して優先順位が付けられるが、この優先順位は、実際には DDF アドレス スペースとは異なる。

可能な場合は、DDF ワークのレスポンス タイム目標を使用する。レスポンスタイム目標では、時間の経過とともにシステムが変化していても、継続的な修正や再検討はそれほど必要にならない。常にアクティブなスレッドの場合は、速度目標をお勧めする。長期実行アプリケーションは、レスポンス タイム目標には適さない。なぜなら、WLM はパフォーマンスをサンプリングするために、指定された数の定期的な完了を必要とするからだ。

速度目標は、オペレーティング システム、ハードウェア、またはシステム上で実行している他のアプリケーションでの変更の影響を受けやすいので、定期的に検討する必要がある。速度目標を使用する場合は、現実的な目標にする。速度目標が高すぎて非現実的だと、チューニング戦略が無駄に終わってしまうことがある。

非常に単純化したサービス クラス定義を考える。たとえば、タイプ DDF と単なる DB2 サブシステム名の定義である。これは、DB2 ワークロードに優先順位を付ける WLM の機能をほとんど利用しない。本来他よりも重要であるワークロードが必然的に存在する。予想以上に長く実行されるが、ほとんどリソースを使用しない分散ワークロードに注意する。これは、ワークロードが最適なサービス クラス (SYSOTHER など) よりも低くなっている状態である。

DB2 サブシステム アドレス スペースの目標

常時実行または長期間実行されるワークには、速度目標を使用する。DB2 サブシステム アドレス スペース (DBM1、MSTR、IRLM、および DDF) は、速度目標で効果があるワークロードの好例である。CICS や IMS 領域など、DB2 を使用した重要なサブシステム アドレス スペースには、高い速度目標を使用する(レスポンス タイム目標を使用していない場合)。VTAM および DB2 IRLM アドレス スペースなどのタスクには、デフォルトの SYSSTC サービス クラスを使用する。

前のセクションでの説明に留意してもらいたい。速度目標は、一見すると設定が簡単なように思われるが、実際には継続的な再検討が必要になる。速度目標は、システム変更の影響を受けやすく、オペレーティング環境の発展に伴って検証し直す必要がある。つまり、WLM が効率的に目的を達成できるように、現実的な目標を定めることが重要である。

WLM の将来

WLM は絶えず、オペレーティング システムにおける役割を強化、拡張している。新機能には、Integrated Resource Director (IRD) 動的アプリケーション環境 (WLM を設定せずに動的に作成され

たアプリケーション環境サーバー アドレス スペース) アドレス スペースの数を指定する機能 (1つや無制限以外) NUMTCB オプションへの拡張機能、WLM が NUMTCB 設定を動的に管理する機能がある。

DB2 では、WLM が提供するサービスの一部を利用している。たとえば、DB2 for z/OS V8 では、NUMTCB を動的に変更する機能を利用しているが、アプリケーション環境の動的な作成は利用していない。WebSphere など、WLM を使用する他のサブシステムの中には、その他の機能を利用するものもある。

結論

WLM は z/OS オペレーティング システムの戦略上、重要なコンポーネントであると言える。大量かつ多様なワークロードを管理できる WLM の能力と柔軟性は、z/OS プラットフォームにとって差別化要因となっている。柔軟性と拡張性を目的とした WLM に対する拡張機能により、必然的に、パフォーマンス管理がより簡単で動的になっている。ユーザーが指定したルールに基づいて、より多くのワークが「バックグラウンド」で行われるため、DB2 だけでなく z/OS 上で実行されている他のワークロードでも、その重要性を実感できるだろう。

関連情報

IBM DB2 情報管理ソリューション

<http://www-6.ibm.com/jp/software/data/>

IBM Tivoli ソフトウェア ソリューション

<http://www-6.ibm.com/jp/software/tivoli/>

DB2 は、International Business Machines Corporation.の登録商標です。この文書に記載されたその他の製品名、語句などは各社の商標です。

埋め込まれたシチュエーションのオーバーヘッドを伴わない、時間によるしきい値の制御方法

Don Zeunert

シニア ソフトウェア エンジニア/IBM

OMEGAMON[®]の多くの顧客は、従属ワークロード時間の問題のために、埋め込まれたシチュエーションを使用している。この方法は、過度のシチュエーション評価と不要なオーバーヘッドを招くことがある。どのシチュエーションが他のシチュエーションを埋め込んでいるかによって、オーバーヘッドの程度に大きな違いが生じる。ここでは、埋め込まれたシチュエーションを使用する一般的な方法を検討した後、管理とリソースの負担が低い別の方法をいくつか取り上げていく。

埋め込まれシチュエーションの一般的な使用法

顧客にとって、膠着状態のシステムを検出する必要が生じることがある。通常この検出は、低いリソース消費または低いボリューム状態を検出することによって行う。この場合のしきい値は、アクティビティ レベルがノーマルからピークの間でのみ有効である。したがって、顧客は月曜日から金曜日にのみ、時には午前8時から午後6時に限り有効なシチュエーションを作成できるのだ。

これを行うには、製品に備えられたWeekdayシチュエーションを、平日にのみ有効なシチュエーションすべてに埋め込むという方法がある。Weekdayシチュエーションを埋め込んだ20のCICSシチュエーションがあり、1分間隔で評価する場合、通常Weekdayは1分ごとに21回評価されることになる。曜日は頻繁に変わらないので、これは過剰によるオーバーヘッドのように思われる。それでは、なぜこのようなことになったのであろうか。

『Administrator's Guide Candle Management Workstation[®] (Candle Management Workstation[®]管理者ガイド)』の第5章「Setting up Situations to Monitor Your Environment (環境を監視するシチュエーションの設定)」には、「Creating a New Situation by Embedding an Existing Situation (既存のシチュエーションの埋め込みによる新しいシチュエーションの作成)」に関する節がある。ここでは、CICS_TransRate_Lowなどの新しい(親)シチュエーションのプロパティにより、Weekdaysのような埋め込まれたシチュエーションのプロパティが上書きされると説明している。つまり、埋め込み先(親)のシチュエーションには、埋め込み元(子)のシチュエーションから要求されたデータのエージェントでのサンプリングが含まれ、親シチュエーションのサンプリング間隔で実行するのだ。上記の20回に及ぶ評価は、これで説明がつく。あるサイトでは、この設定のために、Weekdayシチュエーションが1分に55回実行されていたのである。

では、21番目のサンプルはどこから採られているのだろうか。これは、Weekdaysシチュエーションを自動開始したときに採られている。このサンプルは、シチュエーション自身のアラートを発生させるつ

もりがない限り不要である。他のシチュエーションに埋め込まれたシチュエーションの場合、通常このようなアラートは不要である。

埋め込まれたシチュエーションの別の使用方法

どのシチュエーションがどのシチュエーションを埋め込むかは、パフォーマンスにとって非常に重要になる。すべての製品固有のシチュエーションを単一の時間シチュエーションに埋め込んでコード化すれば、それぞれのシチュエーション（親と子）は1度しか実行されない。これは、埋め込まれたどのシチュエーションも自動開始しない限り該当する。

このアプローチの欠点は、イベントビューで発生したシチュエーションが、一般名（Weekdays）を持つ親シチュエーションになるという点だ。Weekdaysから「create another」によってシチュエーションCICS_Stallを作成し、続いて、CICS_CPU_LowやCICS_TranRate_Lowなどを、OR条件として埋め込むことができる。この場合に発生したイベントシチュエーション名が問題につながることもある。

ただし、両方が製品固有のシチュエーションであり、一方が他方よりもオーバーヘッドが高い場合、どちらのシチュエーションを親にするかを考慮する必要がある。たとえば、システムCPUまたはCICS CPUは、プロセス/タスクCPU例外よりもオーバーヘッドは低い。したがって、プロセス/タスクシチュエーションを複数のシチュエーションに埋め込んだりはしない。

埋め込まれたシチュエーションの欠点

埋め込まれたシチュエーションを使用するときは、それらのいずれも（親も子も）シチュエーションの同期には向いていない。同期によって、製品固有のシチュエーションが頻繁に実行されることになり、スケジュールされている追加データ収集からのオーバーヘッドの増加が生じてしまうのだ。

時間別のしきい値に対する推奨アプローチ

時刻または曜日別にしきい値を設けてオーバーヘッドを減らすには、別の方法がある。時間を単一の製品シチュエーションに埋め込んだり、複数の製品シチュエーションを時間シチュエーションに埋め込んだりする代わりに、時間に基づいて偽アラートを制御するポリシーを使用できる。この方法には、どの製品シチュエーションもシチュエーションの同期向きだが、それらが適用できない間、CMSシチュエーション監視ではアラートの処理（フィルタリング）が不要になるという利点があるのだ。

これによりCMSサイクルが減少する。すべてのシチュエーションを同期化することにより、シチュエーションの総数が減り、CMSおよびCMAのオーバーヘッドも減少する。もう1つの利点は、ポリシーがOMEGAMON DEインターフェースでサポートされるという点である。ただし、埋め込まれたシチュエーションは、従来のCandle Management Workstationインターフェースでしかサポートされていない。

エージェントおよびCMSのオーバーヘッドを低減するポリシー作成手順

オーバーヘッドを減らすために、しきい値を調べる期間（Weekday）に1つ、しきい値を適用しない期間（Weekend）に1つ、合計2つのシチュエーションを作成する。アクティブな時間シチュエーション（Weekday）に基づいたポリシー（Weekday_Policy）を作成する。CICS_TransRate_Lowなど、アクティブにする必要のある製品シチュエーションをすべて開始するようにWeekdayポリシーを設定する。続いて、非アクティブな期間に基づいた2番目のポリシー（Weekend_Policy）を作成し、最初のポリシーで開始したすべてのシチュエーションを停止させるように設定する。図1を参照してほしい。

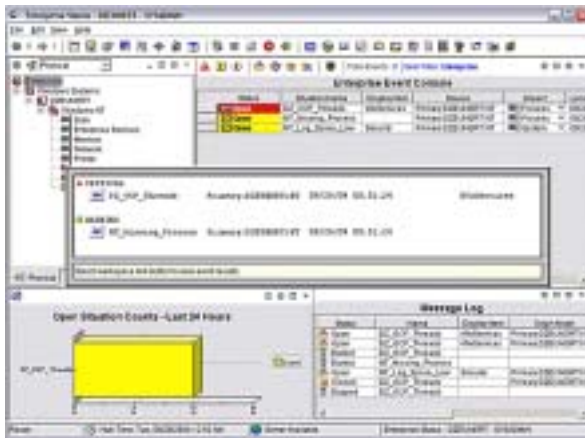


図1: 平日に実行するようにすべてのNTシチュエーションを開始しているWeekdayシチュエーション
(ポリシーにより自動開始し、エージェントに割り当てられる)

親によって自動開始するシチュエーションは、もはや埋め込まれていないので、必ずすべてのシチュエーションが自動開始するように設定する。また、自動開始しないとシチュエーション同期処理に向かなくなるので、シチュエーションを最初に開始するときにはポリシーに頼らないようにする。図2を参照してほしい。

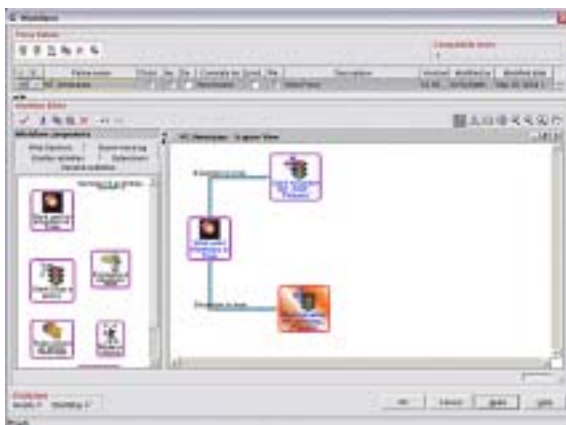


図2: 真と評価され、ポリシーにより開始されているシチュエーション

このコンセプトは、定期保守期間での偽アラートの排除にも役立つ。たとえば、すべてのCICS領域

を、毎晩午前2時から午前4時まで停止させる場合、この予定された停止に関連するアラートは回避する必要がある。

この場合、偽アラートを生成するシチュエーションすべてを停止する処理を設定したポリシー `Begin_Maintenance_Window` を作成できる。また、これに対応する、保守終了ウィンドウのポリシーも設定することができる。これらのポリシーは、通常生成されるアラートすべてを認識する代わりに、オペレータが手動で開始および停止させることができる。

これらのポリシーのいずれかを自動化する場合は、内部スケジューリング シチュエーションを作成するのではなく、自動化/スケジューリングによってトリガーされるシチュエーションを作成する。時間シチュエーションでトリガーしたり、手動で開始するポリシーの代わりに、コンソール イベントからトリガーできる。CMS RAS1 ログ イベントをトラップできるように Universal Message コンソールを設定した場合は、先月号の記事「*OMEGAMON* データと *Tivoli Event Console* との統合」を参照して、ポリシー アクションをトリガーするためのイベントのトラップ方法を確認してもらいたい。

<http://www.candle.co.jp/download/material.html#material06> (2004年12月末まで公開URL)

<http://www-6.ibm.com/jp/software/zseries/newsletter/ccr2/index.html> (2005年1月から公開URL)

したがって、CMS に対する変更コマンドを発行して、たとえば「`F CMS,ECHO MNT0800 CICS maintenance window`」などのメッセージを RAS1 ログに書き込むように、自動化を設定するだけでよいのである。この場合、このメッセージは、ポリシーの実行に使用できる Universal Message ベースのシチュエーションでのトラップに利用できる。

結論

埋め込まれたシチュエーションを使用する方法はいくつかあり、それぞれに固有の長所と欠点がある。これらの特性は、複数の製品シチュエーションを埋め込む場合に考慮する必要がある。ただし、時間ベースの監視だけが必要な場合は、ポリシーを使用して、発生するアラートと抑制するアラートを制御した方がよいだろう。

関連情報

シチュエーションの同期に関する詳細は、CCR2資料である「[Customizing OMEGAMON XE for maximum benefit and minimal overhead – Part 1: Common data collection overhead reduction tips](#)」を参照

<http://www-306.ibm.com/software/tivoli/features/ccr2/ccr2-2004-02/techtip-common.html>

IBM Tivoli ソフトウェア ソリューション

www-306.ibm.com/software/tivoli/ (英語)

<http://www-6.ibm.com/jp/software/tivoli/> (日本語)

図に関する補足

図1 (Policy_Weekdays.GIF)

このポリシーは開始後、平日になるまで、つまりWeekdayシチュエーションが真と評価されるまで待機する。真と評価されると、週末に休止している2つのシチュエーションを開始する。ポリシーは自動的にWeekdayシチュエーションを開始するが、これがエージェントへ割り当てられ、妥当な評価間隔が設定されるようにする必要がある。この場合、曜日は頻繁に変わらないので、評価間隔は10分で十分である。

図2 (Alerts_weekdays.GIF)

ワークスペースの右下のビュー(枠内)のMessageログには、Startedのステータスになっている2つのシチュエーション(NT_Missing_ProcessおよびDZ_OCP_Threads)が示されている。このステータスの変更は、ポリシーによって行われた。この同じビューには次のOPENのステータスも示されており、これは、シチュエーションが真と評価されたことを示している。このアラートは動的に開始された後、00:51まで発生しなかった。これは、最前面のアラート ウィンドウに図示されている。