Part 2-Technical trend
第2部―技術動向

急速に進展するサーバの仮想化 ネットワークの普及と高度化で 新たな段階に突入

サーバを仮想化し、複数台のサーバをまとめて仮想的な巨大なコンピューティング・リソース・プールを実現すれば、管理が容易になり、かつリソースの有効活用が可能になる。グリッド技術でも使われているこの考え方は、現在のサーバ仮想化の根本をなすアイデアである。現在サーバの仮想化技術が注目されているのは、高速なネットワークやインターコネクトの普及、サーバの低価格化、プロセッサの性能向上が困難になりつつあることなど、さまざまな要因を背景にしている。ここでは、技術面での現状を概観しておこう。

サーバの仮想化の基本概念

現在、ベンダー各社が精力的に取り組んでいるのがサーバの仮想化である。Part1でも紹介したとおり、コンピュータの発展の歴史は仮想化の進展の歴史と見ることができる。しかし、従来の仮想化技術では、サーバを中心として、サーバに組み込まれる各種のコンポーネントを仮想化する、という形で行なわれてきた。具体的には、SMPによるプロセッサの仮想化や、ストレージの仮想化などだ。これによって、サーバを単に「サーバ」という1つの論理的な実体と考え、取り扱うことが可能になり、実際にサーバを構成している個々のコンポーネントのレベルで詳細を把握する必要はなくなった。しかし、実はこのレベルの仮想化は、その洗練度には差があるものの、かつてメインフレームで実現されていた仮想化とほぼ同様の成果を、オープンシステム、つまりUNIX系OSやWindowsなどのPC用OSの上でも実現しようとした試みだといえる。

メインフレームの時代は、単一の巨大サーバであるメインフレームの上でさまざまなタスクを実行していたため、問題の構成は遙かに単純だった。つまり、1つのサーバと複数のタスクの組み合わせ問題だったわけだ。しかし、オープンシステムの時代に入り、巨大で高価なメインフレームは、複数の安価なサーバマシンに分割された。一方、タスクは相変わらず複数存在するため、複数のサーバと複数のタスクの組み合わせ問題を解く必要が生じたのである。

タスクごとに専用のサーバを割り当て、問題を「(1台のサーバに1つのタスク)×複数セット」というレベルに単純化できれば対応は容易だし、実際にそうした運用体制を取る例は少なからずある。しかし、この構成では、サーバの処理能力とタスク処理のための負荷にアンバランスが生じ、ほとんどの場合でサーバの処理能力が余り、使われないまま死蔵される結果になる。つまり、サーバの利用効率が極めて低くなってしまうのである。

現在オープンシステムを中心に進展しつつある仮想化実現の取り組みは、結局のところ複数のサーバ上で複数のタスクを処理する際の効率を高め、かつ管理負担を軽減したい、という要求から生まれている。これはつまり、メインフレームを複数の安価で小型のサーバに分割する、という動きを完成に導き、複数台のサーバからなるシステムを巨大なメインフレームと同等の存在としたうえでメインフレームを置き換えよう、という発想のゴール地点なのである。

ネットワークで接続された複数台のサーバが、全体で1つの巨大サーバ(コンピュータ・プール)に見えるようにするためには、いくつかの技術要素が必要となる。具体的には、各サーバの負荷状況を的確に把握するモニタ機能や、タスクの負荷を予測しつつ適切なサーバに投入するためのタスク・スケジューリング機能や全体の管理機能などだ。さらに、コンピュータ・プールにサーバを追加したり、逆に取り外したりするための機能も必要になるだろう。これらは通常、プロビジョニングと呼ばれている。現在、サーバの仮想化が急速に進展しつつあるのは、複数台のサーバ間でタスクを分散するスケジューリングやモニタ、そしてプロビジョニングの技術が完成しつつあることが背景にある。

仮想化のさまざまなレベル

さて、ここでいったん話を戻し、サーバの仮想化の実現手法をレイヤ別に整理してみよう。

Part1でも述べたとおり、コンピュータの基本的なハードウェア・コンポーネントは、OSによって仮想化されている。さらに、OSの違いによるコンピュータのインタフェースの違いは、現実にはミドルウェアによって仮想化されているといえ

るだろう。

一方で、OS自体を仮想化しようとするアプローチもある。コンピュータ上に仮想マシンを構築し、その上でさまざまなOSを実行するものだ。VMwareがよく知られているが、仮想マシンによる実行環境という意味では、JavaもまたOSレベルでの仮想化の取り組みの一例といえるだろう。

ミドルウェアを仮想化するための取り組みは、具体的にはちょっと思いつかないのだが、アプリケーション層での仮想化手法には、またさまざまなものが並列している。一例としてあげられるのが、シトリックス・システムズのMetaFrameである。これは、Windowsアプリケーションのユーザー・インタフェースをUNIX系OSなどに転送することで、Windowsアプリケーションを非Windows環境で実行するためによく使われている。プラットフォームを問わずにアプリケーションを実行でき、アプリケーション実行に必要なシステム・リソースもリモートに配置され、ローカルでのリソース消費がないという点で、MetaFrameはアプリケーション層以下を丸ごと仮想化してネットワークに展開した例だと考えることができるだろう。

別の取り組みとしては、Webサービスをはじめとするアプリケーションのコンポーネント化の流れがある。こちらは、アプリケーション・コンポーネントをサービス化し、ネットワーク上に展開してリモートから呼び出し可能とするものだ。標準化されたインタフェースを介して呼び出される各アプリケーション・サービスは、それぞれ異なるシステム上で実行されるが、そのコードの実装や、稼働環境については呼び出し側では一切気にする必要はない。こうしたやり方も、アプ

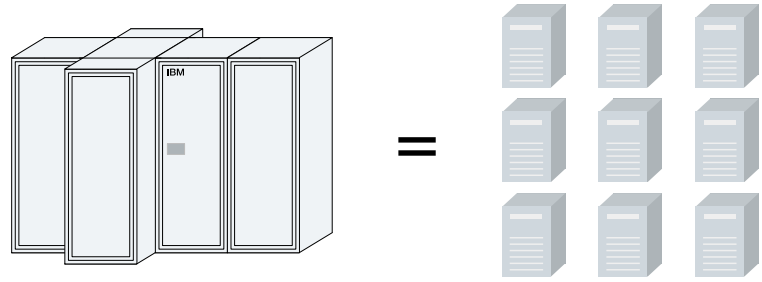

図1 サーバの仮想化は、処理能力の面だけでなく、運用管理面においてもかつてのメインフレームと同等の環境になるよう、オープンシステムによる分散環境を成熟させていく動きの最後のステップとなる。

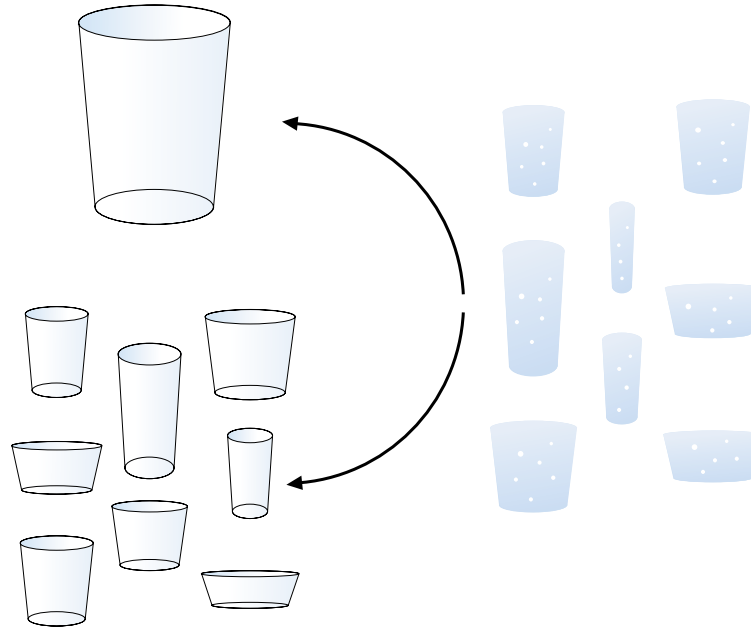

図2 巨大な1台のサーバに複数のタスクを投入するなら問題は比較的単純だが、それぞれ処理能力が異なる複数のサーバに複数のタスクを投入する場合、組み合わせ問題が発生するため、問題が一気に複雑化する。

リケーション層での仮想化の実現手法の1つと考えてよい。必ずしもサーバ全体を仮想化しなくても、必要に応じて各レイヤで実現されている仮想化手法をうまく組み合わせることで、管理コストの削減やハードウェア・リソースの有効活用など、さまざまなメリットを引き出すことが可能になる。

グリッド技術の進歩

現在のサーバ仮想化実現手法の主役となりつつあるのが、グリッド技術である。グリッドは、複数のコンピュータをまとめて仮想的なコンピュータ・プールを構築する技術で、最初に述べたサーバの仮想化方法に直接対応する技術であるといえる。

グリッド技術は当初、科学技術計算の分野で実装が始まった。膨大な計算量を要する科学技術計算のためには、従来はその用途に特化したハードウェア設計を持つベクトル型スーパーコンピュータが使われてきた。しかし、あまりに特殊なハードウェアであるベクトル型スーパーコンピュータは極めて高価なリソースでもある。かつて、ベクトル型スーパーコンピュータの輸入を巡って日米間で通商摩擦が勃発したほどだ。

科学技術計算を実行したい組織のうち、ベクトル型スーパーコンピュータを必要なだけ導入できたところはそう多くはない。そこで、不足するコンピューティング・リソースを低コストで調達するために考えられたのが、安価な通常のプロセッサ（スカラ型プロセッサ）を搭載した一般的なPCやサーバを多数集めてまとめ上げ、総計算量を高める手法である。これが、グリッドの始まりである。

当初のグリッドは、科学技術計算に特化して設計されていた。というのも、集められた複数のコンピュータ（ノード）に処理を分割していくためには、そもそも最初に投入されるタスクが多数の膨大な処理単位に分割可能になっていないとうまくいかないためだ。そのため、初期のグリッドで実行可能なアプリケーションを作成するには、分散実行をあらかじめ考慮したプログラミングを行なう必要があった。現在では、マルチスレッド・プログラミングは一般的な技法となりつつあり、ライブラリやツールキットなどの支援環境も充実してきている。しかし、初期のグリッドでは、グリッドシステムごとに異なる手法でアプリケーションを記述しなければならないなど困難も多かった。さらに、科学技術計算では膨大なデータに対して同じ操作を延々行なっていく処理が多いため、多数のノードで手分けして処理していくことが比較的容易だったことも、まず科学技術計算でグリッドが実用化された理由としてあげられる。

このように、かつては「科学技術計算専用システム」として特殊化した用途に利用されていたグリッドがこれほどまでに注目を集めるようになってきたのは、「多数のコンピュータを仮想的に1台の巨大コンピュータであるかのように見せる」というグリッドの特徴が、分散システムをメインフレームと同等にしたいという要望と綺麗に合致したためだ。技術面でも、グリッドの利用が拡大するに従って標準化の動きが生まれ、特定の科学技術計算に対応するだけでなく、さまざまな用途に汎用的に利用可能となる方向性が見えてきた。さらに、Javaのようなマルチスレッド・プログラミングをサポートし、ハードウェア・プラットフォームを問わず実行できるプログラミング言語が普及したことも、異機種混在で構成されるグリッドを実用化するのに大きな役割を果たしている。

図3 コンピュータを論理的なレイヤ構造に分割して考える場合、仮想化はそれぞれのレイヤごとに異なる技術を用いて行なわれている。必ずしも、複数台のサーバを巨大コンピュータ・プールにまとめ上げる技術だけがサーバの仮想化なのではない。

プロビジョニング技術の整備

最近急速な進歩を遂げた分野の1つに、プロビジョニング技術をあげることもできるだろう。プロビジョニング

(Provisioning)は、現時点では定まった訳語がないのだが、準備を整えて提供する、といったイメージで捉えるとわかりやすいだろう。

サーバの仮想化という文脈でプロビジョニングを位置づけるのであれば、「グリッド環境に自動的にサーバを追加したり、逆にグリッドからサーバを取り除いたりする技術」と理解することも可能だろう。前にも見たとおり、現実のサーバは論理的に複数のレイヤに分割して理解することができる。ハードウェアがあり、OSやミドルウェアによってソフトウェア環境が作られた上に、目的の処理を行なうアプリケーションが稼働する。プロビジョニングの目的の1つは、用意されたハードウェアに対して必要となるソフトウェア環境を用意し、実行可能なようにすべてインストールして初期設定を行なうことにある。当然、アプリケーションによって必要な環境は異なるため、現実のサーバはOSやミドルウェア、アプリケーションの各レベルで細かなパラメータ・チューニングが行なわれていることも珍しくはない。こうした環境整備は、システムの構成を熟知したシステム管理者が経験に基づいて蓄積されたノウハウを駆使して行なっていたが、当然人力で処理するのは相応の時間が掛かるうえ、熟練した技術者が付きっきりになることで生じる高いコストも無視できない。プロビジョニング技術は、この作業を自動化することで、必要なタイミングで適切にサーバを用意することを可能にし、かつ運用管理コストを大幅に引き下げる。グリッドを構築しても、投入されるタスクの総量が増加していけば、いずれ処理能力不足に陥ることになる。このとき、サーバの追加に熟練技術者の長時間労働を要するようでは、実用的な環境とは言い難い。グリッド技術とプロビジョニング技術を組み合わせることが仮想化実現のための重要な要件と考えられているのは、この点が理由となる。

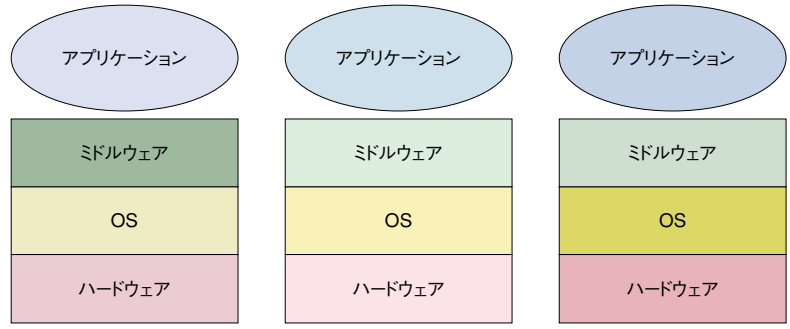

図4 アプリケーションによって、前提とするハードウェアやOS、ミドルウェアの環境が異なるため、アプリケーションを実行するためにサーバを用意するなら、当然アプリケーションごとに対応する環境構築まで行なう必要がある。この作業を自動化するのがプロビジョニングである。

サーバの仮想化の将来像

サーバの仮想化は、現在かなりの進捗を示しているが、ベンダー各社の取り組みが平行してそれぞれ独自に行なわれているため、現時点ではバラバラなソリューションが混在している状況だ。しかし、今後は他の分野での成果と同様に、サーバの仮想化に関してもある程度の標準化が実現することが期待される。可能性としては、仮想化の中核となる「仮想コンピュータ・プール制御コンソール」といった存在となるソフトウェアでまずデファクト・スタンダードが確立され、やがては物理的なサーバが最初からこのソフトウェアに対応する形で設計され、リリースされるようになるのではないだろうか。この段階に至れば、コンピューティングの中心的な存在は現在のサーバから仮想化ソフトウェアに完全に移行し、サーバは「仮想化ソフトウェアが作る環境に組み込んで利用するためのコンポーネント」と考えられるようになるかもしれない。

もちろん、そうした環境が実現するのはまだまだ時間を要することは間違いない。その間、既存のシステムから漸進的に仮想化環境への以降を実現する必要があるため、今後の新規サーバの導入に関しては慎重な検討が必要になるだろう。

特に、既存のサーバをうまく仮想化環境に組み込むためには、現在ベンダーごとに個別に提供されている仮想化技術をうまく組み合わせ、互換性に配慮しながら取捨選択していく必要があるのはまちがいないだろう。