Hewlett Packard Enterprise

HP-UX 特集コラム バックナンバー

― 2006 年 ―

- J2EE パフォーマンス・チューニングの"虎の巻"（2006 年 1 月）
- Solaris と HP-UX の間のカベを取り除く「SHPK」（2006 年 2 月）
- ジム・ヘイズが語る HP-UX の歴史と未来（2006 年 3 月）
- Virtualization Manager は「サーバ仮想化のポータル」をめざす（2006 年 4 月）
- 新しい Integrity サーバの魅力を探る（2006 年 5 月）
- 大きく変貌する HP-UX のシステム管理（2006 年 6 月）
- 「アイデンティティ管理」のための Red Hat Directory Server（2006 年 7 月）
- DB クラスタ構築でハマらないための「HA Simple Cluster リファレンス・アーキテクチャ　v.1.0」（2006 年 8 月）
- HA クラスタの達人に聞く「Serviceguard が選ばれる理由」（2006 年 9 月）
- 暗号化セキュリティで重要情報を保護（2006 年 10 月）
- Oracle/Solaris 移行の現場ノウハウ（2006 年 12 月）

その他の年のバックナンバーは下記からご参照ください。

2009 年 | **2008 年** | **2007 年** | **2006 年** | **2005 年** | **2004 年**

―目次―

J2EE パフォーマンス・チューニングの"虎の巻"（2006 年 1 月）

2005 年 11 月に開催された「JavaOne Tokyo 2005」で行われた米国 HPE エンタープライズ Java ラボのエンジニア、ジョセフ・コア氏の講演内容をもとに、J2EE パフォーマンス・チューニングのコツについて紹介します。

Solaris と HP-UX の間のカベを取り除く「SHPK」（2006 年 2 月）

SHPK の概要とその利用方法などについて紹介します。

ジム・ヘイズが語る HP-UX の歴史と未来（2006 年 3 月）

HPE のバイス・プレジデントであるジム・ヘイズ氏が初期の HP-UX 開発のいきさつや、現在まで HP-UX がどのような進化や成長を遂げてきたかを語ります。

Virtualization Manager は「サーバ仮想化のポータル」をめざす（2006 年 4 月）

2006 年 4 月にリリースされた「Integrity Essentials Virtualization Manager」と「Integrity Essentials Capacity Advisor」を取り上げ、その特徴や機能などについて紹介します。

新しい Integrity サーバの魅力を探る（2006 年 5 月）

本特集では、sx2000 チップセットの開発を指揮した HPE のハウエル・フェルゼンタール氏へのインタビューをもとに、新しい Integrity サーバの魅力について紹介します。

大きく変貌する HP-UX のシステム管理（2006 年 6 月）

本特集では、Web ベースの新ツール SMH の概要を紹介するとともに、SMH を中核とする HP-UX のシステム管理全般について解説します。

「アイデンティティ管理」のための Red Hat Directory Server（2006 年 7 月）

本特集では、HP-UX 版 RHDS を取り上げ、同ディレクトリ・サーバを中心とするアイデンティティ管理の構築手法について紹介します。

DB クラスタ構築でハマらないための「HA Simple Cluster リファレンス・アーキテクチャ　v.1.0」（2006 年 8 月）

本特集では、Simple Cluster を取り上げ、その概要や高速フェイルオーバーを実現するメカニズムについて紹介します。

HA クラスタの達人に聞く「Serviceguard が選ばれる理由」（2006 年 9 月）

本特集では、HA クラスタ・ソリューションとしての Serviceguard の確固たる優位性がどのような経緯で培われてきたのかを明らかにします。

暗号化セキュリティで重要情報を保護（2006 年 10 月）

本特集では、セキュリティ保護の"最後の砦"として用いられる暗号化ボリューム＆ファイルシステム「EVFS(Encrypted Volume & Filesystem)」について紹介します。

Oracle/Solaris 移行の現場ノウハウ（2006 年 12 月）

本特集では、Solaris ベースの Oracle サーバを HP-UX＋Integrity サーバへ移行する実例を通じて、「現場の移行ノウハウ」を紹介します。

J2EE パフォーマンス・チューニングの“虎の巻”

J2EE パフォーマンス・チューニングの“虎の巻”・前編

2006 年 1 月　テクニカルライター 吉川和巳

J2EE アプリケーションのパフォーマンス・チューニングでは、まずトランザクションのレスポンス時間の内訳をドリルダウンすることが肝要だ。米国 HPE エンタープライズ Java ラボのエンジニア、ジョセフ・コア氏は、「こうしたレスポンス時間の内訳を知ることが、システム全体の SLO(サービスレベル目標)を設定する基礎となる」と説明する。では、そうしたトランザクションの“解剖”はどのような手法で実施すればよいのだろうか。2005 年 11 月に開催された「JavaOne Tokyo 2005」におけるコア氏のセッション内容をもとに、“Java パフォーマンス・チューニングのプロフェッショナル”が明かすテクニックを紹介する。

J2EE のトランザクションを解剖する

J2EE アプリケーションのパフォーマンス・チューニングにおいて、コア氏がまず強調するポイントは、トランザクションの「レスポンス時間の内訳」をドリルダウンする作業の必要性だ。図 1 は、J2EE アプリケーションにおけるユーザ・レスポンス時間がどのような処理に費やされるか示した図である。

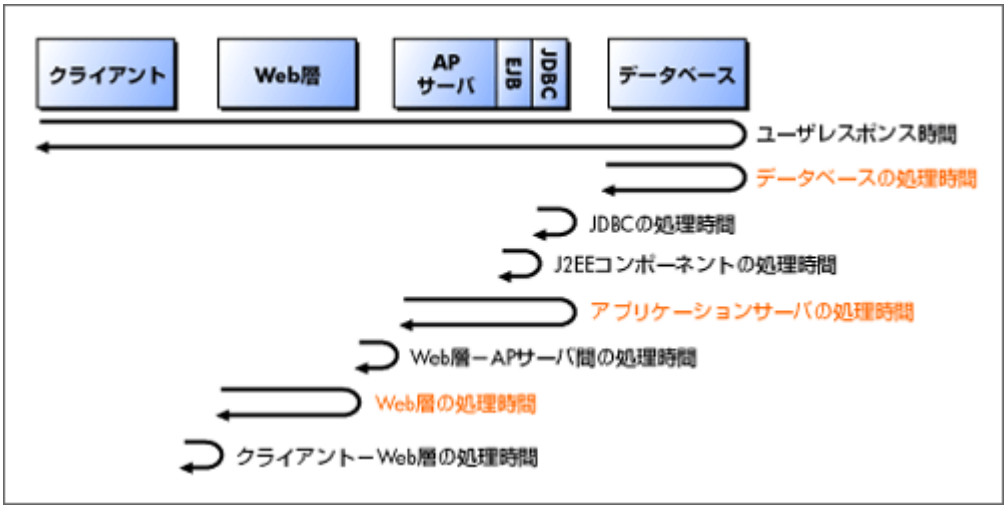


図 1：J2EE アプリケーションのレスポンス時間の内訳

ユーザから見たレスポンス時間は、「Web 層」、「アプリケーション・サーバ」、「データベース」の各階層における処理時間を集約したものだ。よって、まずはこの 3 つの階層のそれぞれで、どの程度の時間が経過しているかを把握する。さらに必要に応じて、階層間の呼び出しに要する時間をはじめ、J2EE コンポーネントや JDBC ドライバ単体での処理時間などを計測する。コア氏は「こうしたレスポンス時間の内訳を知ることが、システム全体の SLO(サービスレベル目標)を設定する基礎となる」と説明する。

どのような内容のトランザクションを実行しているのか理解しておくことも、不可欠なステップである。例えば高度に複雑なトランザクションや、長時間を要する大規模な検索など、それぞれがどのくらいの頻度で実行されるかを明らかにする。こうした理解を通じて、個々のアプリケーションに最適なアーキテクチャが次第にあぶり出されてくる。「例えば大規模な検索については、Fast Lane Reader パターンの導入も検討すべきだろう」とコア氏は述べる。ここで言う Fast Lane Reader パターンとは、リー

ドオンリーのデータベース・アクセスについては EJB や O/R マッピングをバイパスし、JDBC を利用して大量のレコードを効率よく取得するデザイン・パターンのことである。

OpenView Transaction Analyzer を使う

こういったトランザクションの"解剖"のためのツールとして、コア氏は HPE が提供するアプリケーション管理ソフトウェア「OpenView Transaction Analyzer(OVTA)」を利用している。OVTA は、J2EE アプリケーションにおけるトランザクションのトレースを実行し、個々の J2EE コンポーネントに至る詳細レベルのパフォーマンスを GUI 表示できるツールである。

OVTA では、まず個々の HTTP リクエストに対するレスポンス時間の一覧が図 2 のように表示される。

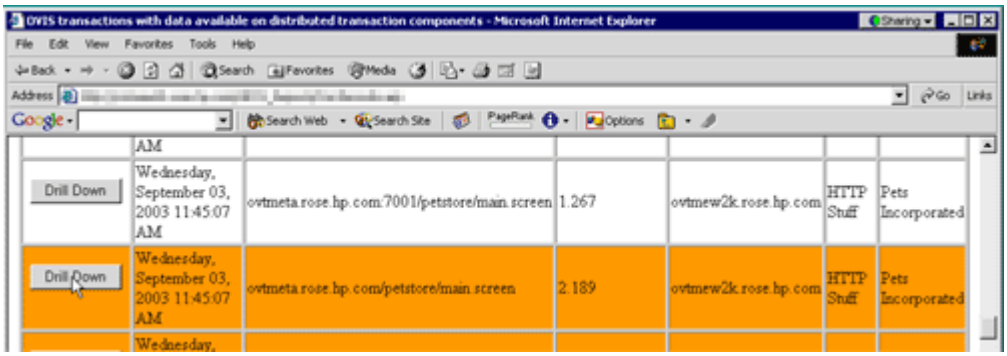


図 2：OVTA による HTTP レスポンス時間の一覧表示

オレンジ色で表示されている HTTP リクエストは、レスポンス時間があらかじめ設定した SLO を超過しているものを表している。そこで各行の「Drill Down」ボタンをクリックすることで、SLO 超過の原因を探るためのトランザクションのドリルダウンが可能になる(図 3)。

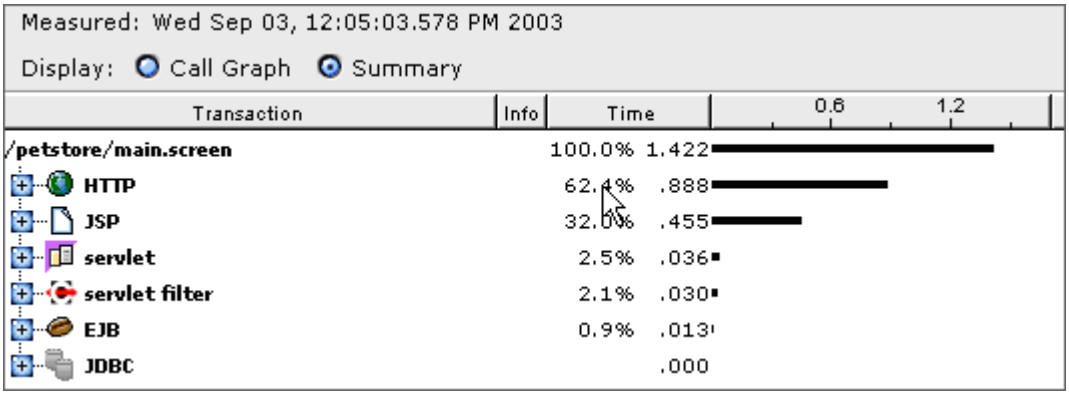


図 3：OVTA によるトランザクションのトレース

図 3 の例では、「/petstore/main.screen」というパスのページを表示するまでに要したレスポンス時間について、J2EE アプリケーションを構成する各階層(HTTP、JSP、サーブレット、EJB、JDBC)がどの程度の割合を占めているかグラフ表示されている。ここから、コード・レベルのコールグラフを分析することが可能だ(図 4)。

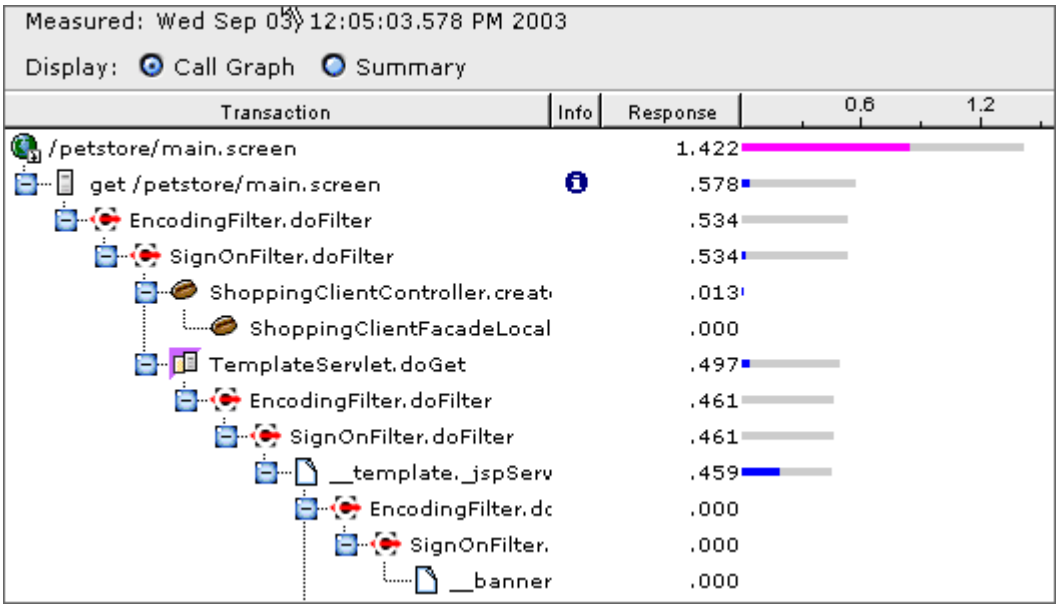


図 4：OVTA によるコールグラフの追跡

さらには、OVTA 製品に備わるツール Java Diagnostics を用いることで、より下位レベルのメソッド単位やスレッド単位の処理時間をはじめ、ガーベジ・コレクションの実行状況やオブジェクト割り当て状況までを追跡表示できる。コア氏は、「OVTA を利用することで、アプリケーションのどの部分で、どのようなタイミングで問題が発生しているかすばやく突き止められ、対処方法も速やかに明らかにできる」と、その有用性を評価している。

パフォーマンスの足を引っ張るそのほかの要因としては、レガシーシステム連携時の CICS(トランザクション処理ミドルウェア)の処理遅延や、ネットワークの遅延、そして HTTP トラフィックの過多などがある。実際にコア氏が経験したある事例では、アプリケーションのレスポンスが思ったように向上しないという現象が発生していた。そこで同氏は OVTA を用いて個々のトランザクションの処理時間をトレースし、その原因が CICS 側にあると特定できたという。

後半では、トランザクション・レベルからシステム・レベルへと視点を変えて、コア氏が明かす Java パフォーマンス・チューニング技法を紹介する。

J2EE アプリケーション設計と実装のコツ

コア氏は、J2EE アプリケーションのスケーラビリティを左右する主要な要素として、「アプリケーションに対する負荷の大きさ」「スレッド数」「キャッシュサイズやプールサイズ」を挙げる。さらに同氏は、「メッセージングを用いる場合はメッセージ処理にかかる遅延時間、そして XML を用いる場合は扱う XML 文書の大きさにそれぞれ注目する。例えば非常に大きな XML 文書を処理する必要があるかどうか、といった部分がポイント」と説明する。

一方、アプリケーションのアーキテクチャについても考慮が必要だ。例えば大規模な J2EE アプリケーションではクラスタ構成をとるのが定石だが、その場合の重要なポイントとして「ローカリティを意識すること」とコア氏は強調する。ここで言うローカリティとは、クラスタにおけるデータの局所性を意味する。「セッション・アフィニティ(ユーザ・セッションを特定ノードに固定する技法)を利用してノード間でのセッション情報の共有を減らし、オーバーヘッドを最小限に抑えることも考慮すべき」という。

また、スケーラビリティが要求される J2EE アプリケーション開発では、その開発工程のひとつのステップとして「パフォーマンス検証」を組み入れることをコア氏は推奨している。具体的には、ユーザ・アクセスをシミュレートするテスト環境を作成し、以下に示す 4 種類のパフォーマンステストを実行する方法を提唱している。

- 負荷テスト――予想される通常負荷レベルを再現する
- ストレステスト――高負荷時に発生する不具合を検証する
- ソークテスト――予想以上の高負荷を再現する
- 長時間テスト――長時間の負荷を再現する

さらに、こうしたアプリケーション全体のテストに加えて、1 ユーザ単位のリソース消費の計測が重要な意味を持つ。「ユーザごとにどの程度のリソースを消費するか実測しておくことで、システムのトレンド解析の基礎データが得られる。アプリケーションが今後どのようにふるまうか、長期間にわたって予測可能になる」とコア氏は説明する。

また昨今では、サービス指向アーキテクチャ(SOA)の隆盛にともない、アプリケーション機能をサービス単位に分割する手法も広く用いられつつある。この SOA にひそむ落とし穴について、コア氏は次のように指摘する。「SOA を導入することで、企業が有するさまざまな情報を抽象化し、個々のモジュールに分割して提供可能になる。しかしこれにはリスクも伴う。そのひとつは、パフォーマンスのオーバーヘッドが増加する懸念。もうひとつは、サービス開発のアプローチがサイロ型(垂直分割型)になりがちな点だ。」(コア氏)そこで SOA 導入では、サイロ型のアプローチをできるだけ避け、企業のビジネス全体を見通した設計や実装を実施できるかがポイントとなる。

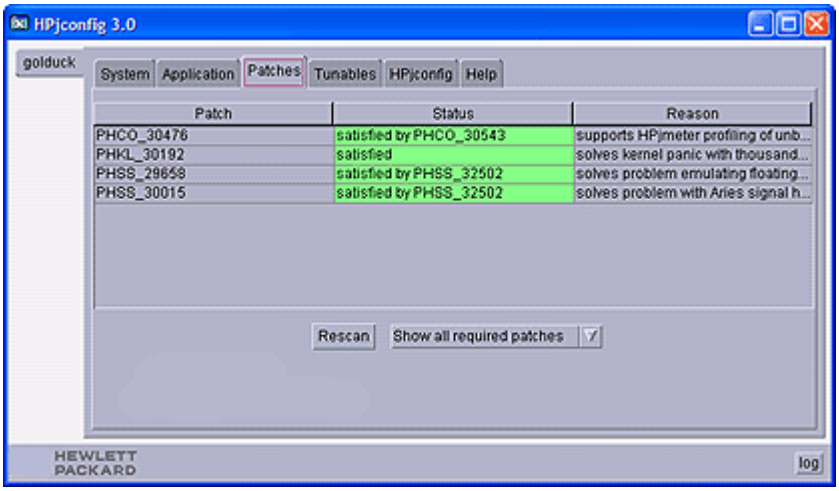
システム全体に着目する

ここまで紹介した手法は、いずれも J2EE アプリケーション内部のふるまいや設計にフォーカスしたものである。これに対し、パフォーマンス・チューニングの要となるポイントとしてコア氏が強調するのは、「システム全体に着目すること」だ。具体的には、以下の各要素についてパフォーマンスの計測と分析を実施していく。

- システム全体
- ネットワーク
- サーバ・ソフトウェア(OS、プロセス)
- サーバ・ハードウェア
- 個々のプロセス(アプリケーション・サーバ、JVM)
- 個々のスレッド

「まずはシステムを適切にセットアップできているか確認する必要がある。またシステムの構成を継続的に監査することも不可欠だ。例えば HPE が提供するツール HPjconfig を使えば、JVM の運用に適したパッチやカーネル・パラメータ設定が OS に適用されているか自動的にチェックすることができる」(コア氏)。

Patches



Kernel Parameters

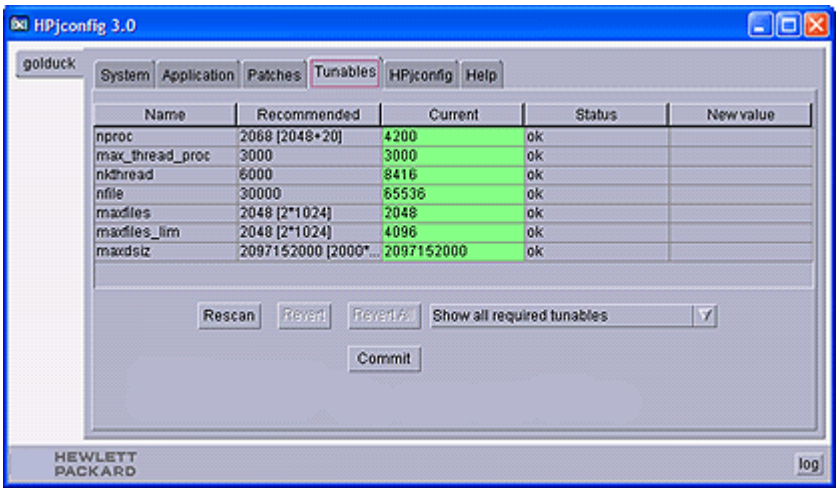


図 5：HPjconfig による診断結果

図 5 は、HPjconfig による OS 構成の診断結果の例である。ここでは、診断対象の HP-UX サーバに追加で適用すべきパッチのリスト、そして設定すべきカーネル・パラメータの値が表示されている。とりわけカーネル・パラメータは、しばしば J2EE アプリケーションのパフォーマンスの足を引っ張る要因となりうる。「ある事例では、アプリケーションの負荷上昇にともない動作の不具合が頻発した。そこでトレースを実行したところ、カーネル・パラメータ設定の変更が原因であると判明した」(コア氏)。

つづく後編では、HP-UX のパフォーマンス計測ツール Glance を活用したチューニング・テクニックを紹介する。

J2EE パフォーマンス・チューニングの“虎の巻”・後編

2006 年 1 月　テクニカルライター 吉川和巳

システム・レベルの Java パフォーマンス計測では、HP-UX の管理ツール Glance が重要な役割を担う。とはいえ、Glance は Java に特化したものではないため、その計測データが何を意味するのかは、自ら各自判断しなくてはならない。例えばコア氏は、「Glance を使えば、Java アプリケーションがオープンしている個々のソケットの送受信バイト数も知ることが可能だ」と説

明する。このようにトランザクション・レベルからシステム・レベルまでを網羅し、いくつかの視点から IT システムの性能を解析する“急がば回れ”の方法論を実践していけば、原因不明のパフォーマンス低下に足元をすくわれることもないはずだ。

Glance を活用しよう

システム・レベルのパフォーマンス計測では、HP-UX の管理ツール Glance が重要な役割を担う。Glance を用いることで、CPU やメモリ、ネットワーク、I/O などのパフォーマンス情報をシステム・レベルで収集し、GUI やコマンドラインで表示できる。また、各プロセスが消費するリソースやオープンファイル数、メモリ領域サイズ、システムコール数、スレッドリストなどを表示することで、プロセス単位でのパフォーマンス分析が可能だ。

ただし、Glance はあくまで OS レベルのパフォーマンスデータの取得のみを目的としたツールであり、Java に特化したものではない。よって、Glance が示す各種のデータが何を意味するのかは、利用者が各自判断しなくてはならない。例えば図 1 を見てみよう。

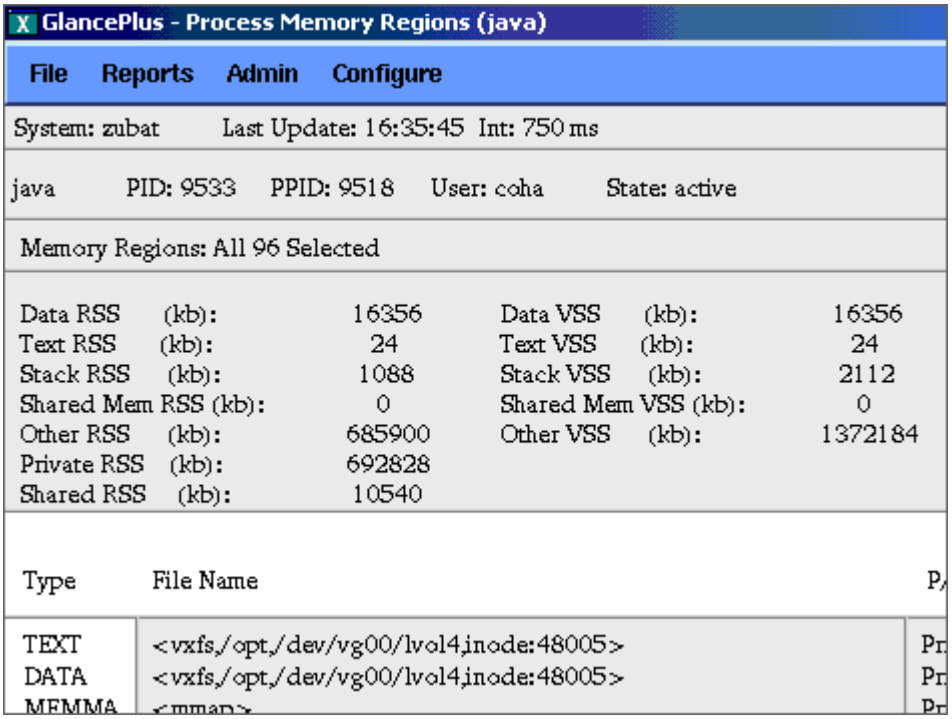


図 1：GlancePlus による JVM プロセスのメモリ消費表示

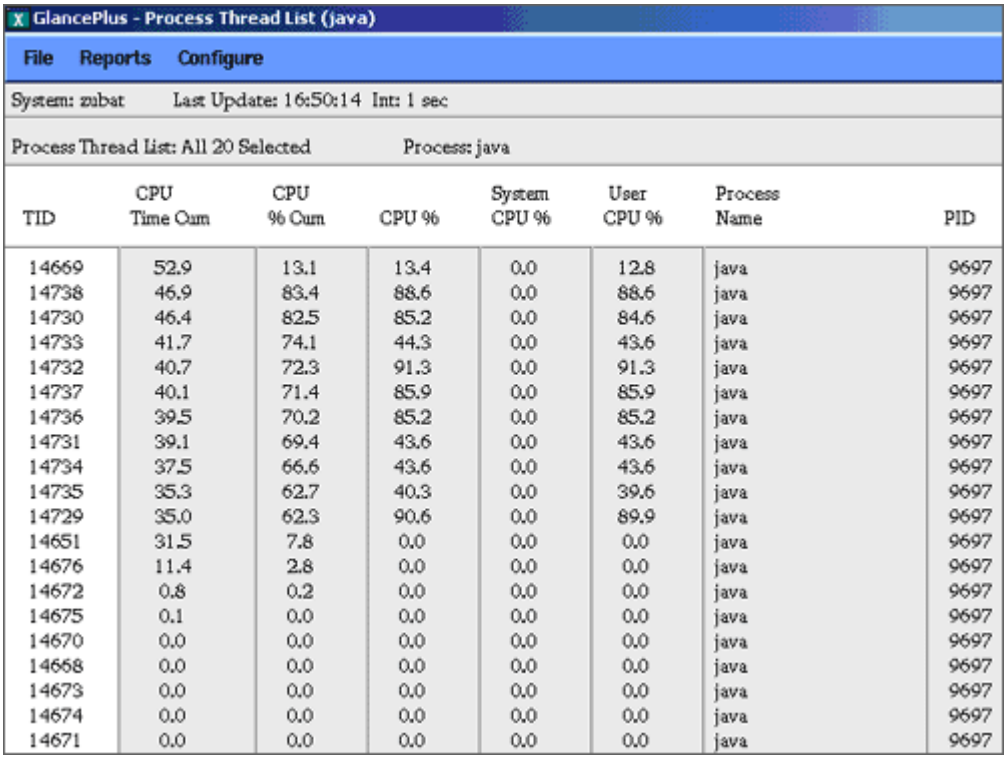
図 1 は、Glance(GUI モード)を用いて JVM プロセスが消費するメモリ領域サイズを表示した例である。ここで、左半分に表示された「RSS」で終わる各行は物理メモリの消費量(Resident Set Size)を示し、右半分の「VSS」の各行は仮想メモリの消費サイズ(Virtual Set Size)を示す。JVM がアクティブに利用しているメモリ量を見るには、RSS の各項目に注目すればよい。以下の表は、JVM プロセスに対して Glance を実行した場合、各項目がどのような意味を表すかを示したものである。この表が示すように、Java アプリケーションが消費するメモリのほとんどは Other RSS のサイズに反映される。

表：JVM プロセスに対する Glance 表示の内容

項目名	内容
Data RSS	JVM 本体が使用するデータ領域
Text RSS	JVM 本体のバイナリコードサイズ
Stack RSS	JVM 本体のメインスレッド用スタック
Other RSS	Java ヒープや共有ライブラリ、Java スレッドのスタックなど

OS スレッドと Java スレッドの対応付け

また Glance では、JVM プロセス内部で動作するすべてのスレッドの一覧を表示できる。


GlancePlus - Process Thread List (java)

File Reports Configure

System: zubat Last Update: 16:50:14 Int: 1 sec

Process Thread List: All 20 Selected Process: java

TID	CPU Time Cum	CPU % Cum	CPU %	System CPU %	User CPU %	Process Name	PID
14669	52.9	13.1	13.4	0.0	12.8	java	9697
14738	46.9	83.4	88.6	0.0	88.6	java	9697
14730	46.4	82.5	85.2	0.0	84.6	java	9697
14733	41.7	74.1	44.3	0.0	43.6	java	9697
14732	40.7	72.3	91.3	0.0	91.3	java	9697
14737	40.1	71.4	85.9	0.0	85.9	java	9697
14736	39.5	70.2	85.2	0.0	85.2	java	9697
14731	39.1	69.4	43.6	0.0	43.6	java	9697
14734	37.5	66.6	43.6	0.0	43.6	java	9697
14735	35.3	62.7	40.3	0.0	39.6	java	9697
14729	35.0	62.3	90.6	0.0	89.9	java	9697
14651	31.5	7.8	0.0	0.0	0.0	java	9697
14676	11.4	2.8	0.0	0.0	0.0	java	9697
14672	0.8	0.2	0.0	0.0	0.0	java	9697
14675	0.1	0.0	0.0	0.0	0.0	java	9697
14670	0.0	0.0	0.0	0.0	0.0	java	9697
14668	0.0	0.0	0.0	0.0	0.0	java	9697
14673	0.0	0.0	0.0	0.0	0.0	java	9697
14674	0.0	0.0	0.0	0.0	0.0	java	9697
14671	0.0	0.0	0.0	0.0	0.0	java	9697

図 2：GlancePlus による JVM プロセスのスレッドリスト表示

ここで表示されるのは OS のスレッド ID であり、Java スレッドの ID がそのまま表示されるわけではない。そこで以下のようなコマンドを実行し、OS スレッドと Java スレッドの対応付けを明らかにする必要がある。

```
# kill -SIGQUIT ＜JVM のプロセス ID＞
```

このコマンドにより、JVM プロセスに対し SIGQUIT シグナルが送信される。HP-UX 対応の JVM「JVM」では、このシグナルを受けて、以下のようなスタックトレースを表示する機能を備えている。

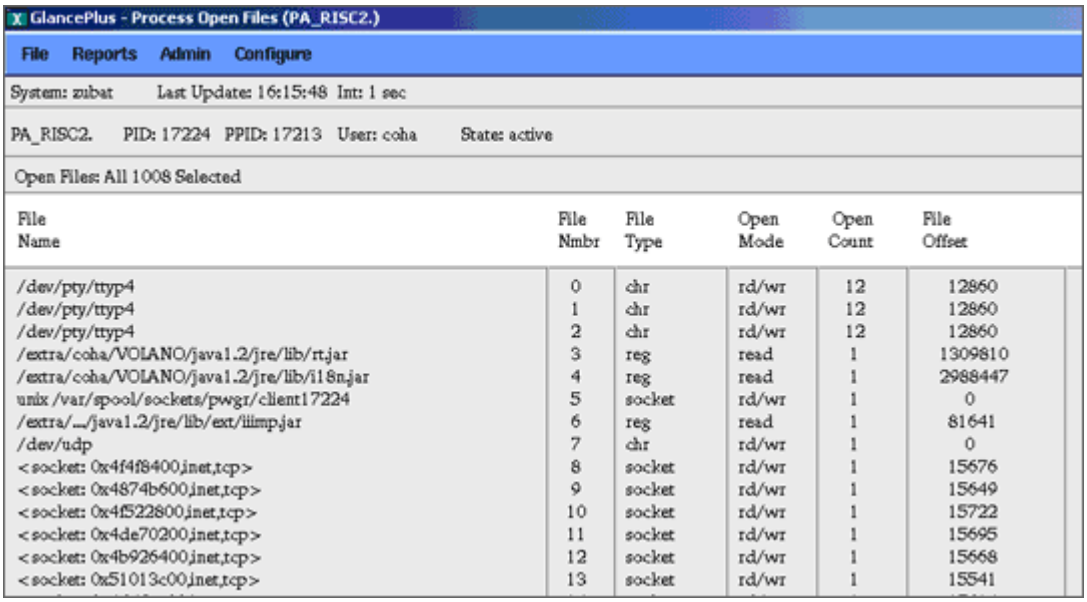
```
"Worker Thread 17" prio=9 tid=0x1310b70 nid=41 lwp_id=14165 suspended[0x1194d000..0x11948478]
at fields.FieldPropertiesLibraryLoader.forClass(FieldPropertiesLibraryLoader.java:67)
```

```
- waiting to lock <0x3ca45848> (a java.lang.Object)
at fields.FieldsServiceImpl.getFpl(FieldsServiceImpl.java:75)
at fields.FieldsServiceImpl.getFpl(FieldsServiceImpl.java:64)
＜以下略＞
```

このスタックトレースにおいて、1 行目の「Worker Thread 17」という部分が Java スレッド名、そして「lwp_id=14165」という部分が HP-UX のスレッド ID を表している。これを見ることで、Glance が表示する各スレッドがどの Java スレッドに対応するのかが分かる。

オープンファイルの調査

Glance を利用することで、JVM プロセスがオープンしているファイルの一覧を表示することも可能だ。


GlancePlus - Process Open Files (PA_RISC2.)

File Reports Admin Configure

System: zubat Last Update: 16:15:48 Int: 1 sec

PA_RISC2. PID: 17224 PPID: 17213 User: coha State: active

Open Files: All 1008 Selected

File Name	File Nmbr	File Type	Open Mode	Open Count	File Offset
/dev/pty/ttyp4	0	chr	rd/wr	12	12860
/dev/pty/ttyp4	1	chr	rd/wr	12	12860
/dev/pty/ttyp4	2	chr	rd/wr	12	12860
/extra/coha/VOLANO/java1.2/jre/lib/rt.jar	3	reg	read	1	1309810
/extra/coha/VOLANO/java1.2/jre/lib/i18n.jar	4	reg	read	1	2988447
unix /var/spool/sockets/pwgr/client17224	5	socket	rd/wr	1	0
/extra/.../java1.2/jre/lib/ext/iiimp.jar	6	reg	read	1	81641
/dev/udp	7	chr	rd/wr	1	0
<socket: 0x4f4f8400,inet,tcp>	8	socket	rd/wr	1	15676
<socket: 0x4874b600,inet,tcp>	9	socket	rd/wr	1	15649
<socket: 0x4f522800,inet,tcp>	10	socket	rd/wr	1	15722
<socket: 0x4de70200,inet,tcp>	11	socket	rd/wr	1	15695
<socket: 0x4b926400,inet,tcp>	12	socket	rd/wr	1	15668
<socket: 0x51013c00,inet,tcp>	13	socket	rd/wr	1	15541

図 3：GlancePlus による JVM プロセスのオープンファイルの表示

図 3 の例では、画面上部の「Open Files」において、JVM プロセスが合計 1008 個のファイルをオープン中であることがわかる。例えば Java アプリケーションのさまざまな負荷状況におけるオープンファイル数を計測しておけば、カーネル・パラメータ「オープン可能な最大ファイル数」に最適な値を導けるはずだ。またオープンファイルの一覧は、ネットワーク I/O の挙動を調べる手段としても使える。「Glance のオープンファイル一覧を見れば、Java アプリケーションがオープンしている TCP ソケット(File Type＝socket)の一覧を確認できる。ファイル・オフセット(File Offset)の値の変化を見れば、個々のソケットの送受信バイト数も知ることが可能だ」(コア氏)。

HPjmeter による Java プロファイリング

コア氏は、J2EE アプリケーションのスケーラビリティを左右する主要な要素として、「アプリケーションに Java プロファイリングの目的は、もっともリソースを多く消費しているメソッドやスレッドを見つけ出すこと。具体的には、メソッド呼び出し回数や CPU 時間、オブジェクト生成数に着目する。さらにコア氏は、「Java アプリケーションのスケーラビリティを左右する要素として、スレッド間のロック競合の頻度もおさえておくべきポイントだ」と述べる。

HPE では、こうした Java プロファイリングを支援する GUI ツール「HPjmeter」を無償で提供している。HPjmeter は、Java プロファイリングの標準インタフェースである JVMPI に準拠しており、ベンダー各社の JVM の-Xrunhprof オプションによるプロファイリング結果を読み込み、上述したポイントについてグラフィカルに解析結果を表示できる。

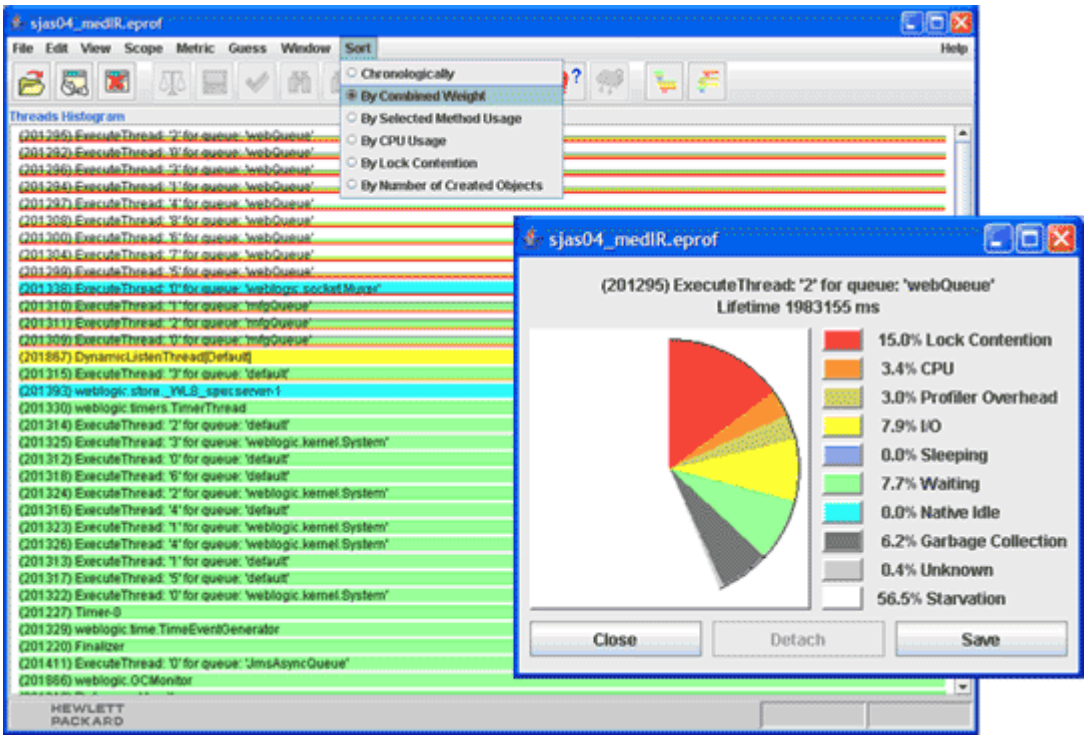


図 4：HPjmeter による Java プロファイリング例

図 4 は、HPjmeter による Java プロファイリング結果の表示例である。この例では Java アプリケーションのスレッド一覧が表示されており、個々のスレッドについて実行時間の内訳を確認できる(手前の円グラフ)。円グラフを見れば、「webQueue」と呼ばれるメッセージ・キューのスレッドでは、処理時間の 15%がロック競合に費やされていることが分かる。

さらに HPjmeter を使えば、オブジェクト・リテンションの発生を防ぐこともできる。オブジェクト・リテンションとは、プログラマーの意図せぬところでオブジェクト間の参照が残留し、不要なオブジェクトが解放されないままメモリを浪費する現象である。一種のメモリ・リークととらえることもできるだろう。HPjmeter は、「大量のオブジェクトへの参照を保持するデータ構造をヒューリスティック(発見的)に探し出し、オブジェクト・リテンションと疑わしき部分をアドバイスしてくれる」(コア氏)。

JVM の-Xeprof オプション

周知のとおり、Java プロファイリングは JVM にとっては負担のかかる処理である。-Xrunhprof オプションを指定すると Java アプリケーションの動作スピードは格段に低下するため、開発環境でしか実施できないのが難点だ。

これに対し JVM では、標準の-Xrunhprof に加えて、-Xeprof オプションによる拡張プロファイリングをサポートしている。この拡張プロファイリング機能はオーバーヘッドがきわめて低く、プロダクション環境に近い負荷状況における Java アプリケーションの挙動を正確につかめるのが特徴だ。またプロファイリングのオン／オフを Java アプリケーションの動作中に切り替えできる。例えば、以下のようなオプション指定が可能だ。

```
-Xeprof:time_on=sigusr1,time_slice=sigusr2
```

ここで、「time_on」および「time_slice」では、プロファイリングの開始と終了を指示するためのシグナルとして「sigusr1」と「sigusr2」を設定している。よって、Java アプリケーションの実行中にこれらのシグナルを送信すれば、例えば実際にトラブルが発生しているプロダクション環境において、リアルタイムにプロファイリングを実施することもできるだろう。

以下の表は、Java パフォーマンス・チューニングで活用できるツールや JVM オプションについて、コア氏がまとめた一覧である。

表：Java パフォーマンス・チューニング技法一覧

作業内容	使用するツールや JVM オプション
アプリケーションのボトルネックの解析	Glance(システム全体、プロセス単位、スレッド単位の CPU 利用率を計測) HPjmeter(スレッドヒストグラム、メソッドコールグラフ、スレッド詳細を解析) -Xeprof オプション(JVM 拡張プロファイリング)／-Xrunhprof オプション(プロファイリング) SIGQUIT シグナル(スタックトレースを解析) gdb(スレッド単位のデバッグ) Prospect および Caliper(HP-UX 対応システム・プロファイラ) OpenView Java Diagnostics
Java ヒープのリアルタイム・モニタリング	VisualGC(サンが提供する GC 可視化ツール) OpenView Java Diagnostics
Java メモリ管理やガーベジ・コレクションの詳細解析	HPjtune -Xverbosegc オプション -Xloggc オプション
Java ヒープのリアルタイム・モニタリング	VisualGC(サンが提供する GC 可視化ツール) OpenView Java Diagnostics
オブジェクト・リテンションの原因究明	-XX:+HeapDump オプション(ヒープダンプ取得) HPjmeter(ヒープダンプ解析) -Xrunhprof オプション OpenView Java Diagnostics(メモリ・リーク解析) gdb(ヒープ内容解析)
例外発生場所の解析	HPjmeter(例外発生解析) -Xeprof オプション OpenView Java Diagnostics(例外発生解析)

以上、今回は JavaOne Tokyo 2005 におけるジョセフ・コア氏のセッション内容をもとに、J2EE パフォーマンス・チューニングの各種技法を紹介した。ここで見てきたとおり、Java アプリケーションの性能向上のコツは、「複数のツールを利用して多角的なチューニング・アプローチを採用する」と要約できるだろう。ひとつのアプローチだけで済ませてしまわず、トランザクション・レベルからシステム・レベルまで、いくつかの視点から IT システムの性能を解析する。こうした手間を惜しまない“急がば回れ”の方法論を実践していけば、原因不明のパフォーマンス低下に足元をすくわれることもないはずだ。

Solaris と HP-UX の間のカベを取り除く「SHPK」

Solaris と HP-UX の間のカベを取り除く「SHPK」・前編

2006 年 2 月　テクニカルライター 吉川和巳

既存の Solaris ベースのソースコードは、当然のことながら、そのままでは HP-UX 上ではコンパイルできない。なぜなら両 OS の間にはライブラリの API やヘッダファイル、スレッドやシグナル、コンパイラオプション、make ツールなどの違いがあるからだ。とはいえ、数 10 万行や数 100 万行といった大規模アプリケーションに対してこうした違いを手直しするには膨大な工数が発生する。この問題を解消するのが「Solaris to HP-UX Porting Kit(SHPK)」である。SHPK では、両 OS 間の違いを吸収する環境を提供することで、Solaris の API やヘッダファイル、makefile などをそのまま HP-UX 上でも利用可能になる。

Solaris と HP-UX の間にあるカベ

既存の Solaris ベースのソースコードは、当然のことながら、そのままでは HP-UX 上ではコンパイルすることはできない。なぜなら、Solaris と HP-UX の間にはおもに以下のような点で違いがあるからだ。

- ライブラリの API
- ヘッダファイルの構成、定義
- スレッドやシグナルの機能、動作
- コンパイラの動作、オプション
- make ツールの機能

そこで、Solaris アプリケーションを HP-UX 上に移行するには、これらの違いをひとつひとつ修正していくポーティング作業が不可欠となる。とはいえ、数 10 万行や数 100 万行といった大規模アプリケーションでは膨大な工数が発生するうえ、そのコストを事前に見積もることさえ難しい。

この問題を解消すべく HPE が昨年 9 月にリリースしたのが、「Solaris to HP-UX Porting Kit(SHPK)」である。SHPK は、Solaris 対応の C/C++ソースコードの HP-UX への移行を支援するツールである。その最大の特徴は、上述したような両 OS 間の違いを吸収する環境を提供すること。つまり SHPK を使えば、Solaris の API やヘッダファイル、makefile などをそのまま HP-UX 上でも利用可能になる。これにより、ポーティング作業に必要な労力を大幅に軽減できる仕組みだ。

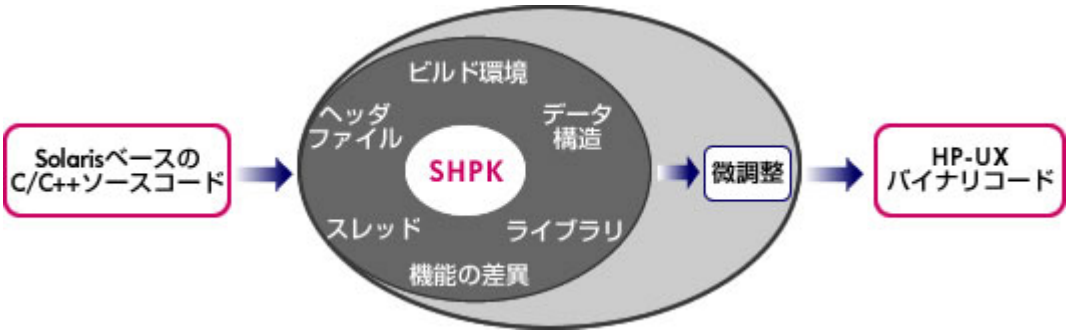


図 1：SHPK による Solaris から HP-UX への移行

今回は、この SHPK を活用することで Solaris から HP-UX へのポーティング作業がどの程度まで簡素化されるか、その実力を検証したい。

Solaris から HP-UX への移行手順

まずは、Solaris から HP-UX へのアプリケーション移行に必要な個々の手順をおさえておこう。一般に、異なる OS 間でのアプリケーション移行には、以下の 3 段階の手順が必要となる。

- アセスメント
- ポーティング
- テスト

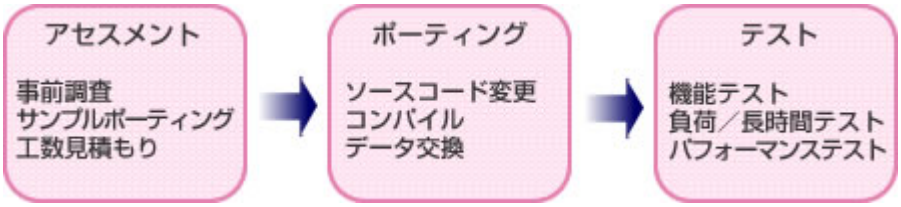


図 2：アプリケーション移行の流れ

アセスメント

アセスメントでは、ポーティング作業に先立ってさまざまな観点から事前調査を実施する。例えば、以下のようなポイントについて調べておく。

- ポーティング作業を支援するにはどのようなツールやドキュメントが利用可能か
- 互換性の無い API や型を使用しているか
- 互換性の無いシステムファイルを利用しているか
- データファイル等で互換性の無いものがあるか(エンディアンなど)
- ISV ライブラリまたはプラットフォーム依存のライブラリを使用していれば、それが移行先においても利用可能か
- 現在利用できないような古い開発環境に依存しているか
- 古い仕様のコンパイラを使っているか(C 言語の K&R、C++の ARM など)
- 移行時に 32bit のソースコードを 64bit 化する必要があるか
- マシンコードや pragma などプラットフォーム依存のものが含まれているか
- 浮動小数点演算処理の精度や処理方法の違いが問題にならないか
- 移行元でテストが定義されているか、またテスト環境構築に工数が必要か

これらのポイントについて調査したのち、実際にいくつかのモジュールを移行してみる「サンプルポーティング」を実施する。これにより、コンパイル時にツールやライブラリが正しく利用できることや、データの移行に問題がないことを確認できる。最後に、サンプルポーティングに必要とした工数をもとに、ソースコード全体を移行するのに必要な工数を見積もる。

ポーティング

ポーティングでは、アセスメントの結果に基づいて実際に移行作業を進めていく。具体的には以下のような作業を実施する。

- ビルド環境の変更(makefile やコンパイラオプション、ツールのオプションなど)
- ソースコードの変更(API や型の違い)
- データファイルの移行

テスト

テスト作業では、移行元の環境で行われていたものと同様のテストを実施する。とりわけアプリケーション移行では、以下の点が重要なポイントとなる。

【機能テスト】

ポーティングして生成した各実行ファイルが正しく動作することを確認するためにそれぞれについて単体テストを行う。さらに、データベースとの接続やネットワークを介した他のアプリケーションとの連携などを検証する接続テストも行う。ポーティング後は、タイミングの問題やメモリ使用量といった OS の動作の違いが、ポーティングしたソフトウェアの動作に反映し、正しく動作しないことがある。この場合、問題の絞り込みやデバッグなどを行い、必要に応じてソースコード変更する必要がある。

【負荷テスト・長時間運用テスト】

これらのテストにより、メモリ等のリソース使用量やパフォーマンスが適切なものかどうかを調べる。メモリリークなどもチェック項目となる。また、カーネルパラメータなど OS の設定を変更する必要がある場合もある。

【パフォーマンステスト】

ポーティングしたソフトウェアが適切な実行速度で動作するかどうかをテストする。対象ソフトウェアを Solaris 上で動作させた場合の実行速度を比較対象とし、移行先のハードウェアスペックを考慮して適切な実行速度を求める。適切なパフォーマンスが得られない場合は、プロファイリングやトレースなどをとってボトルネックを調べ、問題点を改善する。

さて、アプリケーション移行に必要なこれらの手順の中でも、とりわけ SHPK が威力を発揮するのはポーティング作業である。ビルド環境やソースコードの変更に際して、従来のような地道な手直しの必要性を大幅に減らしている。つづいては、SHPK の基本的な仕組みや利用方法について紹介しよう。

SHPK とは

前述のとおり SHPK は、Solaris 対応の C/C++ソースコードの HP-UX への移行を支援するツールである。Solaris と HP-UX の間のさまざまな差異、例えばライブラリの API やヘッダファイル、スレッドやシグナル、コンパイラオプション、make ツールなどの違いを吸収する環境を提供する。現在のところ、SHPK はトライアル版として無償でリリースされており、入手するには日本ヒューレット・パッカードへの問い合わせが必要となる。製品としての正式サポートは提供されていないが、将来的にはサポートの提供が計画されている。

SHPK がサポートする移行元環境は、Solaris 2.5 以降である。一方、移行先環境は HP-UX 11i v2 となる。インテル® Itanium® 2 プロセッサおよび PA-RISC プロセッサの両方に対応する。プログラミング言語としては C および C++に対応しており、移行に際しては SHPK の他に aC++/ANSI C コンパイラ のバージョン 6.x が必要となる。

ひとつ留意しておくべき点は、SHPK は“Solaris のエミュレータ”ではないということだ。あくまでソースコードの移行を支援するツールであり、バイナリ互換性を提供するものではない。また、従来のポーティング作業に比較して修正すべき部分は格段に減るものの、100%無修正でコンパイルできることを保証するツールではない。よって、SHPK だけで対応できない部分については、手作業によるソースコード修正が必要となる。

SHPK の恩恵をもっとも享受できるのは、社内開発のソフトウェアや ISV ソフトウェアだろう。一般的にこうしたソフトウェアは、特定のプラットフォームをターゲットに記述されることが多く、SHPK がきわめて重宝するはずだ。一方、オープンソースソフトウェアについてはもともとポータブルなビルドツール(configure など)が整えられているものが多く、SHPK を利用するまでもないだろう。ちなみに、HPE では多数のオープンソースソフトウェアを HP-UX へ移行済みで、「Internet Express」として無償で公開している。

さて、SHPK は以下のコンポーネントから構成されている

- ソースコードスキャナ
- Solaris 互換ライブラリとヘッダファイル
- Solaris 互換ビルドツール

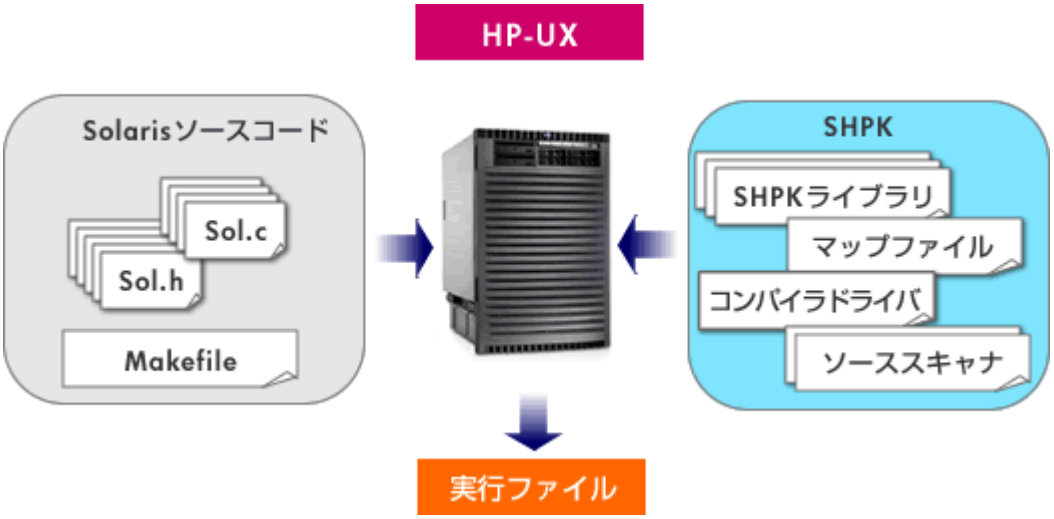


図 3：SHPK によるアプリケーション移行

以下、これらの各要素について詳しく見ていこう。

ソースコードスキャナ

SHPK には、ソースコードスキャナ「sh_src_scanner」が含まれている。このソースコードスキャナを用いて Solaris 対応ソースコードを走査することで、どの部分は SHPK によってそのまま互換性が提供されるか、そしてどの部分を手直しすべきが、ログとして以下のように出力される仕組みだ。

```
$ cat shpk_keyword_header.txt
```

File Path : /home/kamei/getmntent.c
Line No : 2
"sys/mnttab.h" will be handled by SHPK

File Path : /home/kamei/getmntent.c
Line No : 8
"struct mnttab" will be handled by SHPK

File Path : /home/kamei/getmntent.c
Line No : 16
"getmntent" will be handled by SHPK

このように、SHPK が自動的に互換性を提供する項目については、「… will be handled by SHPK」(SHPK が対応)と表示される。一方、ソースコードを手作業で修正する必要がある項目には「Manual Porting is Required for …」(手作業による修正が必要)と表示される。よって後者の内容や数を見れば、ポーティング作業にかかる工数を見積もることが可能になる。

Solaris 互換ライブラリとヘッダファイル

SHPK では、以下の Solaris 互換ライブラリ(SHPK ライブラリ)を提供している。ソースコード中で Solaris 固有の API を呼び出している部分については、コンパイル時にこのライブラリにリンクされる。

表：SHPK が提供する Solaris 互換ライブラリ

ライブラリ	機能
libwrapsc.a	システムコール関連
libwrapstream.a	ネットワーク関連
libwrapstl.a	Solaris スレッド関連

ちなみに、SHPK では 32 ビットと 64 ビットのそれぞれに対応したライブラリを提供しており、32 ビットと 64 ビットどちらの実行ファイルも作成可能だ。 一方、SHPK では Solaris 互換のヘッダファイルも多数提供している。例えばソースコード内で stdio.h をインクルードしている場合、HP-UX における標準ヘッダファイル(/usr/include/stdio.h)ではなく、SHPK が提供する Solaris 互換ヘッダファイル(/opt/shpk/include/stdio.h)がインクルードされる。このヘッダファイルにおいて Solaris と HP-UX の API や型などの違いを吸収することで、上述の SHPK ライブラリをリンクできる仕組みになっている。

Solaris 互換ビルドツール

Solaris から HP-UX への移行に際して面倒な部分のひとつが、ビルドツール make のふるまいの違いである。Solaris の make で使用する makefile の文法には、HP-UX のそれとは異なる部分がある。例えば Solaris では、makefile でのファイル名やディレクトリ名の指定において「%」などのワイルドカード指定が可能だが、HP-UX の make にはこれに相当する機能はない。こうした違いのため、makefile の内容を逐一直していかねばならず、膨大な工数が発生してしまう。

そこで SHPK では、Solaris 互換の make である「sh_make」を提供している。sh_make を用いることで、Solaris 対応の makefile をそのまま HP-UX 上で実行することができる。もっとも、明示的にリンクしているライブラリについては Solaris と HP-UX で名前が異なる場合があり、その場合には makefile を修正する必要がある。

また Solaris と HP-UX では、コンパイラのオプションにも違いがある。例えば、共有ライブラリなどの位置独立なオブジェクトコードを生成する場合、Solaris では「-G」オプションを指定するが、HP-UX では「-b」オプションを指定する必要がある。

こうした違いを吸収するために、SHPK では、コンパイルドライバ「sh_cc」および「sh_c++」を提供している。これらのコンパイルドライバでは、Solaris コンパイラ用のオプションを C/aC++コンパイラ用のオプションに変換したのち、C/aC++コンパイラを起動する。これにより、コンパイルオプションの手直しは最小限に抑えることができる。ちなみに、オプションの対応付けは SHPK のマップファイルで定義されており、必要に応じて修正が可能である。

表：SHPK コンパイラドライバのマップファイル

```
$ cat /opt/shpk/lib/sun2hp.map
                :                :
-Bstatic => -Wl,-a,archive
-G => -b
-H => -Wp,-h
-KPIC => +Z
-Kpic => +z
-PIC => +Z
                :                :
```

SHPK ではその他にも、Solaris の「ar」、「ld」、「ln」の動作を再現する「sh_ar」、「sh_ld」、「sh_ln」といった Solaris 互換ユーティリティを提供している。これらも、上述のコンパイラドライバと同様に、Solaris の各コマンドのオプション指定に対応するものだ。こうした各種ツールによって、Solaris 互換のビルド環境を HP-UX 上に再現している。

特集の後編では、SHPK によるポーティングと工数削減の実際の例を紹介する。

Solaris と HP-UX の間のカベを取り除く「SHPK」・後編

2006 年 2 月　テクニカルライター 吉川和巳

SHPK の利用はきわめて簡単だ。ソースコードスキャンによって検出される手直し部分が少なければ、ほとんど手を加えずに Solaris 対応アプリケーションのコンパイルが可能になる。SHPK の利用により、ポーティング作業にかかる日数はおよそ 1/4〜1/8 に短縮される例も報告されている。また総じて言えば、HP-UX のネーティブ・アプリケーションと何ら変わることのないパフォーマンスを引き出せる。さらに SHPK を利用して「ポーティングのプロトタイプ版」を作成しておけば、早い段階でテストを開始して問題を早期に発見できる。Solaris ベースのソフトウェアに対する投資を保護する手段として、SHPK を活用しない手はないだろう。

SHPK によるポーティングの例

では、SHPK によるポーティング作業の実例を簡単に紹介しよう。ここでは、Solaris 用の以下のサンプルプログラム getmntent.c を HP-UX に移行する手順を説明する。

```
$ cat getmntent.c
#include <stdio.h>
#include <sys/mnttab.h>

int main(int argc, char **argv)
{
  FILE *fp;
  int ret;
  struct mnttab mt;

  fp = fopen ("/etc/mnttab", "r");
  if (fp == NULL) {
    printf ("can not open /etc/mnttab\n");
    exit (1);
  }
  while (1) {
    ret = getmntent(fp, &mt);
    if (ret == -1) break;
    printf ("%s on %s\n", mt.mnt_special, mt.mnt_mountp);
  }
  exit (0);
}
```

sh_src_scanner によるソースコードスキャン

まずは、SHPK に備わるソースコードスキャナ sh_src_scanner を用いて、上述のソースコードをスキャンする。

```
$ sh_src_scanner getmntent.c
```

この結果、src_scanner_shpk.log というディレクトリが作成され、その下に shpk_keyword_header.txt というファイルが作成される。このファイルには以下のようなログが記録されている。

```
$ cat shpk_keyword_header.txt
File Path : /home/kamei/getmntent.c
Line No : 2
"sys/mnttab.h" will be handled by SHPK

File Path : /home/kamei/getmntent.c
Line No : 8
"struct mnttab" will be handled by SHPK
File Path : /home/kamei/getmntent.c
```

```
Line No : 16
"getmntent" will be handled by SHPK
```

ここでは、ソースコード上の 3 か所において HP-UX との非互換性が検出され、それぞれについて SHPK が自動的に互換性を提供していることが示されている。つまりこれらについては、ソースコードを手直しする必要はない。

sh_cc によるコンパイル

つづいてはソースコードのコンパイルである。例えば上述のソースコードでは「sys/mnttab.h」というヘッダファイルをインクルードしているが、HP-UX には同ファイルは存在しない。またソースコード内で呼び出されている関数 getmntent()は、HP-UX にも存在するものの、引数の数や戻り値の型が Solaris とは異なる。よって通常のコンパイラではエラーとなってしまう。一方、sh_cc を用いてコンパイルすることで、そうした違いを自動的に吸収し、エラーを起こさず実行ファイルを生成できる。

```
$ sh_cc -o getmntent getmntent.c
```

ここで nm コマンドを実行して生成された実行ファイルを調べると、SHPK ライブラリがリンクされていることがわかる。

```
$ nm getmntent| grep shpk_getmntent
[64]      |      67112192|      288|FUNC |GLOB |0| .text|__shpk_getmntent
```

sh_make によるビルド

ここで、上記ソースコードをビルドするための以下のような makefile があるとしよう。

```
$ cat Makefile
all:
$(CC) -o getmntent getmntent.c
```

こうした場合、HP-UX の make をそのまま使用するのではなく、SHPK の sh_make を用いてビルドを実施する。

```
$ sh_make -f Makefile
sh_cc -o getmntent getmntent.c
```

このように、makefile 内で sh_cc を明示的に指定していなくても、sh_make 側で自動的に sh_cc を起動する仕組みだ。また makefile 内部に Solaris 固有の文法が含まれていても、それらを事前に修正しておく必要はない。

ここまで見てきたとおり、SHPK の利用はきわめて簡単だ。ソースコードスキャンによって検出される手直し部分が少なければ、ほとんど手を加えずに Solaris 対応アプリケーションのコンパイルが可能になる。

SHPK がもたらす効果

では、SHPK の導入によりポーティング作業の工数はどの程度削減されるのだろうか。以下に、実際のポーティング事例における実績を示す。
表：SHPK および SLPK によるポーティング作業実績

コードサイズ	未使用時の作業量	使用時の作業量	使用ツール
12 万行	8 日	2 日	SHPK
10 万行	30 日	4 日	SLPK
80 万行	80 日(見積値)	10 日	SLPK
120 万行	N/A	25 日	SLPK
400 万行	N/A	15 日	SLPK

ここで、「SLPK」で示される実績データは、「Solaris to Linux Porting Kit」による例である。SLPK とは SHPK と同様の Solaris 互換ビルド環境を Linux 上に再現したものであり、その能力は SHPK とほぼ同等と考えてよい。この表が示すとおり、ポーティング作業にかかる日数はおよそ 1/4～1/8 に短縮されることがわかる。また 400 万行という大規模アプリケーションの移行についても、15 日間というきわめて短い時間で作業を終えている例もある。もちろん、SHPK の有効性はアプリケーションの内容によって変化するが、単純な手作業のみによるポーティングに比較して大幅な期間短縮が期待できることは間違いない。

パフォーマンス面でのインパクト

前述のとおり、SHPK では SHPK ライブラリと呼ばれる Solaris 互換ライブラリを提供することで Solaris とのソース互換性を実現している。つまり、生成される実行可能ファイルには、SHPK ライブラリがリンクされる格好となる。

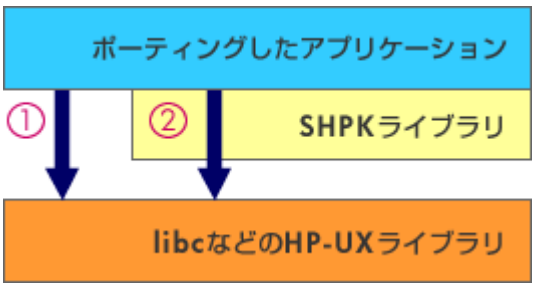


図 1：SHPK ライブラリの利用

では、この SHPK ライブラリを経由することによるパフォーマンス面でのインパクトはどの程度だろうか。
実のところ、同ライブラリでは関数名や引数を対応する名前や型にマッピングするといった単純な処理をするだけであり、大きなパフォーマンス低下は生じない。また Solaris と HP-UX 間で差異がない部分については SHPK ライブラリを通過しないので、SHPK によるオーバーヘッドは生じない。総じて言えば、HP-UX のネーティブ・アプリケーションと何ら変わることのないパフォーマンスを引き出せると考えて差し支えないだろう。

SHPK のさまざまな活用

もちろん、開発者の中には「SHPK を利用すると Solaris の API がソースコードに残ってしまい、HP-UX への完全な移行にはならない」と考える向きもあるだろう。とはいえ、そうした HP-UX への完全移行が必要なケースでも、SHPK を活用することでポーティング作業の効率化を図ることが可能だ。

早期のテストが可能

一般にアプリケーションのポーティングでは、依存関係にあるモジュールのポーティングがすべて完了するまで動作テストを開始することができない。アプリケーションが大規模な場合は、ポーティングの最終段階で動作テストが可能になるケースもある。しかし、その段階で初めて表面化するさまざまな問題は、OS 間の動作やタイミングの微妙な違いや、もともとのソースコードに起因するものが多い。そのため、ポーティングの最終段階でこれらの問題解決に予期しない工数を取られ、大幅に工数が不足してしまう原因ともなる。

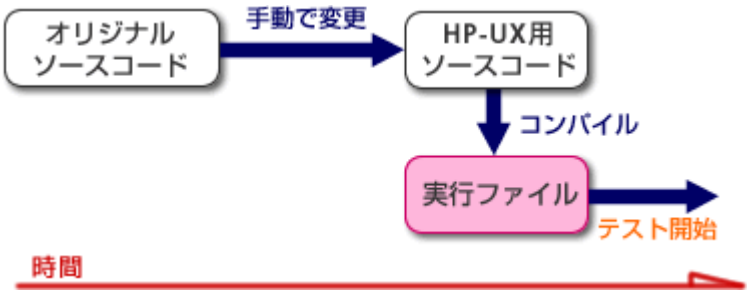


図 2：SHPK を利用しないポーティング

一方、SHPK を利用して、いわば「ポーティングのプロトタイプ版」を作成しておけば、早い段階でテストを開始して問題を早期に発見できる。例えば浮動小数点演算の精度や動作の検証なども早い段階で行うことが可能だ。また、テストで見つかった問題への対応と、SHPK を利用しない手作業によるポーティングを並行して進められる。SHPK を利用することで、このように途中の段階で動作検証を行いながら作業を段階的に進めることが可能だ。これにより、ポーティング全般の工数見積もりをより正確に行うことができるようになる。

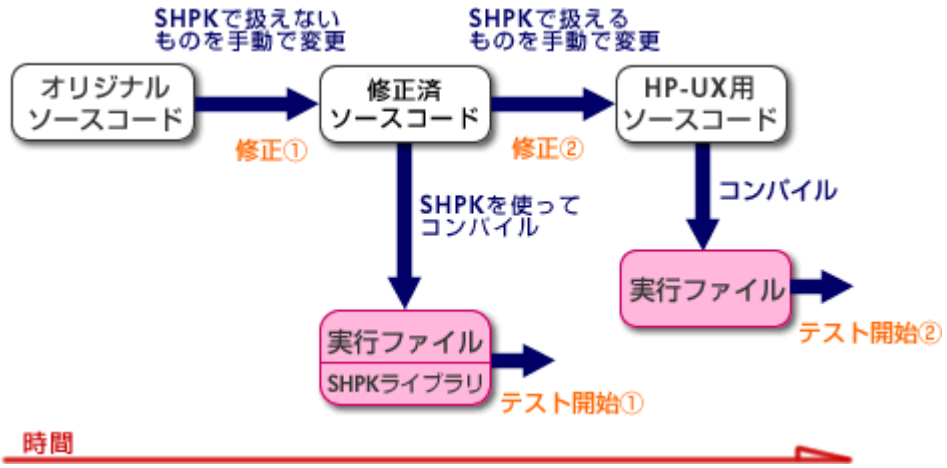


図 3：SHPK を活用したポーティング

問題の切り分けに利用

図 3 の「修正(1)」で必要となる手直しは、主にコンパイラの文法チェックでの厳しさの違いによるエラーや警告メッセージの修正となる。一方、「修正(2)」での手直しは、API やヘッダファイルの違いにより必要となる修正となる。こうした段階的なポーティングを行うことで、ポーティング途中で問題がいずれの段階で発生したものなのかを容易に判断できる。 さらに、SHPK を使ってポーティングして作った実行ファイルと、SHPK を使わずにソースコードを完全に HP-UX 用に修正して作った実行ファイルの動作を比較することにより、修正(2)による手直しの内容が適切であったどうかを検証することも可能だ。

【STKS との併用】

HP では以前より、「Software Transition Kit for Solaris(STKS) 」と呼ばれるソフトウェア移行キットを提供している。この STKS は、簡単に言えばソースコードスキャナと移行支援ドキュメントのセットである。SHPK のような Solaris 互換ライブラリやビルド環境は備えていない。

とはいえ、STKS のソースコードスキャナは SHPK よりも詳細な情報を提供するという特徴がある。とりわけ「その問題をどのように修正すれば良いか」を知るには、STKS が生成するレポートが役に立つ。よって、SHPK によるポーティング時に手作業によるソースコード修正が必要になった場合は、STKS のレポートを参考にして作業を進めるのが効率的だ。

もっとも、STKS のレポートには非常に多くの情報が含まれるため、「かえってわかりにくい」または「関係の無い情報が多すぎる」という場合もある。なぜなら、修正が必要となる可能性のあるものがすべてレポートに含まれてしまうからだ。実際には、ソースコードを変更しなくてもコンパイルが成功するものもたくさんある。そこで、まずは SHPK を使ってポーティングを済ませ、テストに入る前に STKS を実行してソースコードをレビューするという段階を踏む。これにより、コンパイルが成功していても問題解決ができていない部分を発見できるようになる。

以上、今回は SHPK による Solaris から HP-UX へのアプリケーション移行について紹介した。ここで見てきたとおり、SHPK はいわば「HP-UX 上につくられた Solaris 環境」だ。とはいえ、よくあるエミュレーターのようにパフォーマンス面でのデメリットはなく、HP-UX のネーティブ・アプリケーションとほとんど変わりない実行可能ファイルを生成できる。Solaris ベースのソフトウェアに対する投資を保護する手段として、SHPK を活用しない手はないだろう。

ジム・ヘイズが語る HP-UX の歴史と未来

ジム・ヘイズが語る HP-UX の歴史と未来・前編

2006 年 3 月　テクニカルライター 吉川和巳

HP-UX がリリースされてから 20 年が過ぎた。その 20 年間の歴史を通じて HPE が継続的に推し進めてきた技術革新によって、HP-UX は今やエンタープライズ・コンピューティング市場を代表するプラットフォームの地位を築いている。今回は、初期の HP-UX 開発のいきさつや、現在まで HP-UX がどのような進化や成長を遂げてきたか認識を新たにすべく、HPE のバイス・プレジデントであるジム・ヘイズ氏にインタビューを行った。ヘイズ氏は、HPE エンタープライズ・ストレージ＆サーバ(ESS)ビジネス・ユニットのビジネス・クリティカル・サーバ(BCS)部門において、インフラストラクチャ・ソリューション＆エンタープライズ UNIX 部門を率いる責任者である。

ジム・ヘイズ氏略歴

ジム・ヘイズ氏は、HPE エンタープライズ・ストレージ＆サーバ(ESS)ビジネス・ユニットのビジネス・クリティカル・サーバ(BCS)部門において、インフラストラクチャ・ソリューション＆エンタープライズ UNIX 部門を率いる。HP-UX オペレーティング・システムをはじめ、ネットワーキングや I/O 技術の開発責任者であり、ビジネス・クリティカル分野の顧客に向けてオペレーティング環境の統合化や品質管理、製品提供を統括している。またヘイズ氏は、管理性やビジネス継続性、仮想化、ユーティリティ・コンピューティング、コンパイラ、そして Java 各種ツールに関連するインフラ製品およびソリューションの開発を取り纏

めている。ヘイズ氏はコンピュータ・システムのビジネス分野で 23 年以上の経験を持ち、とくにソフトウェアおよびシステム・アーキテクチャを専門としている。バンダービルト大学でコンピュータ科学と数学の学士号を取得している。

HP-UX との 20 年の歩み

80 年代――HP-UX の誕生

1982 年に HPE に入社したジム・ヘイズ氏は、当時の 1000 シリーズを扱う部門であるデータ・システム部門に参加した。その後、同氏は HPE 最初の PA-RISC ベース・システムの開発を担当した Spectrum UNIX チームに異動。80 年代後半には HPE の UNIX カーネルラボにて HP-UX のリード・アーキテクトを務めた。インタビューの手始めとして、この時代における HP-UX 誕生のいきさつから話を伺った。

編集部：HP-UX が市場にリリースされてから 20 周年を迎えました。80 年代当初、HP-UX 開発チームはどのような思想や戦略に基づいて開発を進めたのでしょうか？

ヘイズ氏：HP-UX のもっとも初期の時代、HPE は技術市場向けに特化したワークステーションビジネスやサーバ・ビジネスを展開していました。当時、HP-UX 対応のワークステーション製品は、CAD/CAM や計測機器制御といった汎用の科学計算市場をターゲットとしていたのです。またそのころ、HPE のサーバ・ビジネスはリアルタイム制御や製造、地震解析、通信分野に重点を置いていました。実際に通信分野では、電話交換機網のアラームを監視するために、初期の段階から HP-UX ベースのソリューションが導入されていました。このように、HP-UX の誕生当初は技術計算分野が導入事例の大半を占めていました。

編集部：その後の HP-UX の進化にともない、その優れたスケーラビリティや可用性、信頼性といったユニークな能力はどのように培われていったのでしょうか？

ヘイズ氏：HP-UX の初期の時代からすでに、OLTP のような基幹業務に UNIX を導入したいというお客様のニーズが出始めていました。こうした業務分野における HP-UX の巨大なポテンシャルが明らかになりつつあるのと合わせて、スケーラビリティや高可用性へのさらなるフォーカスを強めていきました。これら HP-UX の主要な能力は、その後の長年にわたって進化を続けており、HPE ではその価値を高める努力を今でも続けています。

80 年代後半には、何社かの主要なお客様との密接な連携を通じて、HPE 最初の HA(高可用性)クラスタ製品である SwitchOver UX を開発しました。その後登場する HA クラスタ製品である Serviceguard と比べればごくベーシックなものでしたが、ミッションクリティカル分野での UNIX 導入を当初から進めてきた通信分野では、非常にポピュラーな製品として普及しました。HPE がミッションクリティカル分野へ参入した要因のひとつとしては、こうした通信分野でのビジネスの実績があります。この分野のお客様は、アプリケーションを UNIX 上で運用することに大変興味を持ち、高レベルのミッションクリティカルプラットフォームを求めていたのです。その後、1995 年には、新しいバージョンの HA 製品として Serviceguard をリリースしました。Serviceguard には、第一世代の HA 製品である SwitchOver UX の市場展開で得られた多数のノウハウが組み込まれました。その後 Serviceguard は市場で広く受け入れられ、今では業界をリードする UNIX HA クラスタソリューションに成長しています。

編集部：HP-UX は、他社 UNIX 製品に対してどのような点で差別化を図っていたのでしょうか？HP-UX にとってのコアコンピテンスは何だったのでしょうか？

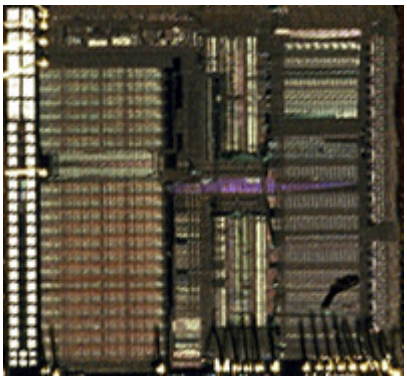
ヘイズ氏：その当時、IBM は彼らの独自 OS 環境を脅かす存在として UNIX をとらえていたため、商用 UNIX ビジネスに本格的に参入していませんでした。また SUN はおもにワークステーションに特化した企業であり、ローコストなワークステーションの開発にフォーカスしていました。一方 HPE は、グラフィクス性能の高さを武器にハイエンド市場にフォーカスしていました。

この時代の HPE および HP-UX のアドバンテージは、メインストリームである商用分野での UNIX 導入のトレンドをすばやく取り込み、OLTP や ERP アプリケーションのより積極的なサポートを UNIX 上で実施したことです。HPE は、商用分野に UNIX を展開した最初のベンダーの 1 社であり、その結果、高可用性やスケーラビリティ、コア・プラットフォームの信頼性といった商用分野で重要となるエリアで、より迅速な開発と市場投入を実現できました。とりわけ、通信分野向けに UNIX ベースの HP-UX ミッションクリティカル・ソリューションを提供していた実績が強みとなり、OLTP や ERP などの商用アプリケーションにもきわめてスムーズに対応できました。このように HPE と HP-UX は、UNIX の持つポテンシャルを引き出し、エンタープライズ・コンピューティング市場で重要な役割を演じることができました。

編集部： 1000 RTE システム(CISC ベースの独自 OS システム)から RISC ベースの UNIX システムへの移行に際して、どのような意気込みを感じていましたか？

ヘイズ氏：私は大学在学中に UNIX についてさまざまな研究を行っていたこともあり、UNIX ベース・システムの開発に携われることに非常に意気込みを感じていました。当時の 1000 システムのおもな市場であった技術分野や製造分野だけでなく、基幹業務を支えるガラス張りのコンピュータ・ルームへと UNIX が踏み込んでいくという、大きな変革の最前線に立つ機会が得られました。HPE は、それまでメインフレームや独自 OS で占められていた分野に UNIX を持ち込むことに成功したのです。

その当時の我々の課題は、HPE 社内の各コンピュータグループがそれぞれに持っていた 16 ビットアーキテクチャを、Spectrum と呼ばれるひとつの 32 ビット・アーキテクチャに統合することでした。この Spectrum が、その後の PA-RISC アーキテクチャです。全社レベルで足並みをそろえることにとても苦労しましたが、結果的にこの統合によってきわめて強力なメリットが得られました。ワークステーションから商用 UNIX サーバまで、HPE のすべてのコンピュータ製品が PA-RISC への投資を有効活用することで、長期にわたってその恩恵を享受できました。これは、HPE 社内のさまざまなグループをひとつの共有アーキテクチャへと束ねる原動力となった当時の CEO であるジョン・ヤング氏の功績と言えるでしょう。このアーキテクチャ統合によって、コンパイラや I/O サブシステムも共通化が可能になりました。また複数の市場セグメントやビジネスにわたって単一のアーキテクチャを投入することで、計り知れないメリットが得られたのです。


初代 PA-RISC

また Spectrum アーキテクチャの開発では、HPE ラボが重要な役目を担いました。さまざまな関係者をひとつ屋根の下に集め、RISC の原則に基づいて、どのようなアーキテクチャにすべきか徹底した議論を行いました。

編集部：80 年代当時の市場において、HP-UX の魅力とはどのようなものだったのでしょうか？

ヘイズ氏：HP-UX は、確たる信念を持ったとても強力なマーケティングチームと開発チームが生み出した製品でした。その信念とは、メインフレームやベンダー独自 OS が占拠してきた商用分野に HP-UX を位置付かせることです。HP-UX と HPE が商用 UNIX 市場における真のリーダーとしての地位を築く上で、このチームはとても効果的な働きをしました。

90 年代初め、ヘイズ氏は HPE を退社してネクストコンピュータに入社する。同社では NeXTSTEP オペレーティング・システム開発グループのシニア・アーキテクトとして、次世代ソフトウェア開発に携わった。

90 年代初め――ネクストとエコシステムズから学んだこと

編集部：90 年代にはネクストコンピュータに入社し、NeXTSTEP OS のアーキテクトとして開発に従事されました。NeXTSTEP の魅力とはどのようなものだったのでしょうか？

ヘイズ氏：ネクストコンピュータは、世界トップクラスのオブジェクト指向開発環境である NeXTSTEP を構築したパイオニア的存在でした。NeXTSTEP は、カーネギーメロン大学が開発した Mach OS と、Objective-C 言語をベースとしたオペレーティング・システムです。そのすべてを統合したきわめて強力な開発環境によって、オブジェクト指向プログラミングのパワーを実証したのです。もし Windows の登場以前にネクストのソフトウェア技術が x86 プロセッサーに移植されていたとしたら、コンピューティングの世界は今とはまったく異なるものとなっていたでしょう。NeXTSTEP はその後アップルコンピュータに買収され、現在の Mac OS X の基礎となった技術のひとつとなりました。

そして 90 年代前半、ヘイズ氏はネクストコンピュータを離れ、エコシステムズの共同設立者となった。同社は分散管理システム構築の先駆けであり、UNIX システムやネットワーク、リレーショナルデータベース、セキュリティなどのプロアクティブ管理をいち早く実現していた。

編集部：その後あなたは、エコシステムの設立に参加しました。エコシステムのビジョンとはどのようなものだったのでしょうか？
また、HPE から離れることでどのようなことを学ばれたでしょうか？

ヘイズ氏：その当時、ネットワークとデータベース、OS を統合管理し、さまざまなイベントや情報を関連づけるための製品が存在しませんでした。エコシステムズのプラットフォームは、これらすべての情報をひとつの管理コンソールに統合し、情報どうしの関係や関連づけを理解しやすく表示するというもので、その当時は斬新で非常にパワフルなツールでした。

より重要な点は、HPE から離れることでさまざまな事柄を学ぶことができたということです。複数のベンダーを相手にする小規模な ISV パートナーにとっては何が課題なのかを理解できました。エコシステムズでは、HP-UX をはじめ、AIX、Solaris、Dynix、SCO UNIX、そしてアムダールのメインフレーム版 UNIX に至るまで、多数の UNIX 製品をサポートしていました。そのおかげで、UNIX OS のさまざまなバージョンやリリース、パッチレベルを管理することの複雑さ、そして多数の異なる UNIX OS 環境に対応したソフトウェアをリリースするために ISV がなすべきことを習得できました。ISV が何を求めているか知ることは、私にとっては目が覚める思いでした。それ以来、HPE に戻った後でも、ISV の視点をより意識するようになりました。

私が学んだもうひとつの点は、エンド・トゥ・エンドのバリューチェーン全体の重要性、つまりパズルの個々のピースをいかにして組み上げていくかということです。エコシステムズは小さな会社だったので、製品開発からデモ実演、販売、サポートまで、たくさんの役割をそれぞれの社員が演じる必要がありました。私自身も、製品ライフサイクルのすべてのフェーズに関わっていまし

た。その経験から、ビジネスのバリューチェーンを構成する多彩な要素、つまり成功する製品を生み出すために欠かせないすべての事柄について、理解を深めることができました。

そして私が学んだもっとも価値のあることは、HPE という企業と、そこで働く人々のすばらしさです。ネクストコンピュータとエコシステムズに在籍している間、私は HPE を外側から見る立場にありました。そのため、企業としての HPE の優れている点、とりわけ HPE が抱える人材の優秀さについて、以前より正しく評価できるようになりました。私が HPE に戻った理由は、まさしくそこに集まる人々にあったのです。私にとって HPE を離れていた時間はとても有意義で、ものの見方を広げるのに役立ちましたが、同時に組織としての HPE の強さも実感させられました。

その後ヘイズ氏は再び HPE に戻り、インテル®Itanium®プロセッサのシステム・アーキテクチャ開発そしてシングル・システム高可用性プログラムをリードすることとなる。後半では、HPE に復帰したヘイズ氏の足跡をたどる。

90 年代後半――Itanium のチャレンジ

その後ヘイズ氏は HPE に戻り、ジェネラルシステムズ部門(GSY)に籍を置く。90 年代半ばにはインテル®Itanium®プロセッサーのシステム・アーキテクチャ開発の共同リーダーを務め、またその後には GSY のシングル・システム高可用性プログラムを率いることとなった。

編集部：あなたは 90 年代半ばには HPE に戻り、インテルとの Itanium の共同開発チームに参加しました。HPE とインテルが Itanium 開発を決定したおもな理由は何だったのでしょうか？　また、インテルや Itanium を選んだことで、HPE はどのようなアドバンテージが得られるのでしょうか？

ヘイズ氏：我々は当時、RISC アーキテクチャの成熟とともに、その後の 20 年にわたってお客様のニーズに応えられる新しいアーキテクチャが必要であると感じていました。既存のどのアーキテクチャにも、いくつかの限界が生じていたのです。新しいアーキテクチャへと移行することで、RISC や CISC といった従来のアーキテクチャの枠組みにとどまるよりも、プログラムに備わる並列性をより引き出すことができると考えました。HPE とインテルが共同で開発を行うことで、インテルのトップクラスの半導体製造技術と、ハイパフォーマンス・コンピューティングやシステム RAS に関する HPE のノウハウを持ち寄ることができます。

HPE にとって Itanium を選ぶメリットは、半導体製造技術のリーダーであるインテルというすばらしいパートナーを得たこと、そして複数 OS をサポートする業界標準プロセッサーを手にしたことです。インテルのトップクラスの製造技術と Itanium プロセッサーの生産量の多さは、最終的には HPE とそのお客様にとってコスト構造を引き下げる効果をもたらします。つまり HPE や OEM パートナー各社は、業界標準のローコストなサプライ品を用いることによるベネフィットを享受できるのです。

編集部：開発チームではあなたはどのような役目を担っていたのでしょうか？

ヘイズ氏：私は製品開発チームの一員として、HPE ラボとともに「WideWord」と呼ばれるプロジェクトに参加していました。WideWord プロジェクトでは、マイクロプロセッサーの命令レベルの並列性を向上させることが焦点となりました。例えばコードの流れに含まれる並列性をより効果的に引き出すためには、チップ上の専用ハードウェアよりもコンパイラが重要な役目を担います。この WideWord プロジェクトにより開発されたアーキテクチャは、複数の実行ユニットを可能にしたアーキテクチャや大規模なレジスタ・ファイル、そしてプリフェッチヒントや投機実行によるメモリ・レイテンシーの制御といったユニークな特徴を数多く備えていました。いずれも、より多くの処理を並行して実行する上で効果を発揮します。

我々はこのアーキテクチャをインテルに持ち込み、共同開発の可能性について検討しました。インテルもまた同時期に新しいアーキテクチャ拡張の作業を進めていたのです。そこで HPE とインテルそれぞれのチームが合流し、Itanium と呼ばれるアーキテクチャの開発が始まりました。そこでの私の役割は、Itanium のシステム・アーキテクチャについて、HPE のエンジニアたちを主導することです。HPE のシステムアーキテクチャ・グループでは、おもに Itanium の特権レベル命令セットや仮想記憶システム、割り込み、トラップ、そして例外モデルといった部分の開発を担当しました。

現在の Intel® Itanium® 2 プロセッサー

編集部：Itanium のような新しいアーキテクチャの設計ではどのような点が課題となりましたか？

ヘイズ氏：もっとも大きな課題は、新しいアーキテクチャを市場に投入してからひとつの大きな流れを形成するまでに時間を要したことです。Itanium のアーキテクチャは大変野心的で、そのポテンシャルをフルに引き出せるように実装技術の成熟度を高めるには時間がかかりました。

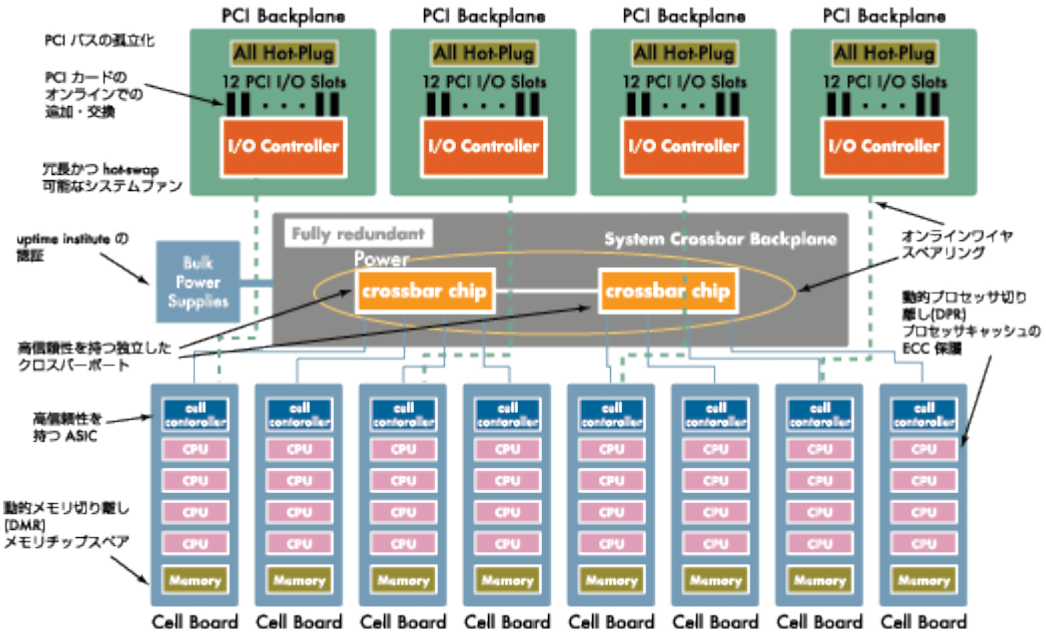
Itanium の EPIC アーキテクチャには数多くのメリットがあります。例えば、コンパイラの強化を今後も続けていくことで、Itanium コアの設計上の特徴をさらにうまく活用できるようになります。また Itanium プロセッサーは、EPIC アーキテクチャの特性を生かすことで、世代ごとにその性能も上昇しています。Itanium のアーキテクチャとプロセッサ実装は、今後何年にもわたって大きなポテンシャルを維持できると考えています。

編集部：その後、あなたはシングル・システム HA アーキテクトに任命されました。あなたの開発チームのミッションはどのようなものでしたか？

ヘイズ氏：我々のミッションは、HPE のシステムの信頼性をさらに向上させ、計画ダウンタイムと計画外ダウンタイムの双方を減らすにはどのような方策が必要か明らかにすることでした。すでに HPE は優れたシステムを持っていましたが、ハードウェアとソフトウェア両方の側面で、よりよいものを追求したかったのです。そこで我々は、統計分析や計測、実験などのテクニックを駆使した、厳格な高可用性解析を導入しました。これらのテクニックにより、システムダウンタイムの本当の要因を突き止めることができました。

まずは、ハードウェアとソフトウェアにおけるダウンタイムの主要な要素を洗い出しました。続いて、その後のハードウェア設計や OS 設計でそれらに対処するためにはどのような方策が必要か明らかにしました。例えば HPE では、メモリサブシステムの耐障害性の改善や、HP-UX 全体の信頼性向上のために、さまざまな対策を講じています。このように、より体系化された手順やモデル、ツール、方法論の投入によって、ハードウェアとソフトウェアのトータルな可用性を劇的に改善するための厳格な規範が形づくられてきました。

この当時に始めたこれらの作業が、Itanium や Superdome、そして HP-UX 11i に至る何年もの製品開発の土台となってきました。この作業は現在でも継続しており、システムの可用性向上に貢献しています。



Superdome の耐障害性機能

インタビュー前半でのヘイズ氏の説明にあるとおり、HP-UX は、UNIX の歴史のごく初期段階からエンタープライズ・コンピューティング市場のリーダーとして強力な基盤を形成していたのが理解できる。HP-UX は、通信分野のようなミッションクリティカル環境を当初からターゲットとしていたのである。そのため、スケーラビリティやパフォーマンス、RAS といったコアコンピテンスへの注力が可能となった。

ヘイズ氏は、ネクストコンピュータとエコシステムズで得た経験とともに HPE に戻り、Itanium アーキテクチャ開発とその後の高可用性プログラムにおいて中心的役割を担うこととなった。その当時の同氏の業績が、エンタープライズ・コンピューティング市場においてゆるぎない継続性を持った HPE の戦略の基幹を支えていると言えるだろう。

インタビュー後半では、仮想化やユーティリティ・コンピューティング、アダプティブ・エンタープライズなど、HP-UX の将来のビジョンについてヘイズ氏に話を伺う。

ジム・ヘイズが語る HP-UX の歴史と未来・後編

2006 年 3 月　テクニカルライター 吉川和巳

2000 年以降、HP-UX は、インターネットの急激な成長や Linux の台頭といったコンピューティングの世界の劇的な変化にも柔軟に適応してきた。それだけでなく、仮想化やユーティリティ・コンピューティングといったユニークな特徴を生かして、市場における役割を一段と広げつつある。インタビューの後半では、HP-UX の現在そして未来についてヘイズ氏に話を聞いた。パーティショニングやインテリジェントポリシーエンジンといった HP-UX の新しいテクノロジーが、HPE のアダプティブ・エンタープライズのビジョンをいかにして実現していくのか、同氏による解説をレポートする。

HP-UX の現在と未来

2000 年以降――次世代のコンピューティングへ

2000 年代の初めに Linux が市場に普及し始めると、HP-UX はしばしばその比較の対象とされた。しかし HP-UX は、エンタープライズ・コンピューティング分野におけるマーケット・リーダーであり続け、現在もその市場シェアを広げつつある。HP-UX がなぜこうした市場での成功を収めることができたのか。ヘイズ氏はその理由について説明する。

編集部：Linux と比較して、HP-UX はおもにどうのような点に強みがあるのでしょうか？また Linux や Windows と比較して、HP-UX はどのような位置づけにあると言えますか？

ヘイズ氏：Linux は当初はデスクトップ PC や 2-way サーバでの利用に特化していました。UNIX のような洗練された高可用性や信頼性、堅牢性、セキュリティ、管理性、そしてリソース管理機能に乏しく、動作する商用アプリケーションの数も限られていました。その後 Linux の地位は向上し、より多くのアプリケーションが利用可能になり、2-way を超える SMP サーバへのスケールアップも可能になりました。以前より機能面は向上しましたが、それでもなお堅牢性や高可用性、セキュリティ、リソース管理といった側面では UNIX の完成度に及びません。これらが、HP-UX の強みです。

例えばスケーラビリティについて言えば、HP-UX は多種多様なワークロードに対してさまざまな面でのスケーラビリティを実現する、バランスのとれた実装がなされており、最大で 128-way までスケールアップが可能です。Linux が UNIX と同様のスケーラビリティを達成し、競合力を見せ始めるまでには時間がかかります。これまで HPE では、HP-UX のきわめて堅牢な実装を実現すべく、OS 内部のさまざまなボトルネックを取り除くためのチューニングに何年もの時間をかけてきたのです。

また可用性については、OS コアの信頼性をはじめ、HP-UX は一段と優れた堅牢性を備えています。また(HA クラスタツールである) Serviceguard のようなソフトウェア製品についても、HP-UX 版はトータルな高可用性機能をそろえています。さらに、新たに追加される高可用性機能や仮想化機能はつねに HP-UX で最初に実装されており、より統合化されたソリューションを提供しています。

Linux や Windows は、それぞれが得意とする分野を持ち、ある部分ではそれらがオーバーラップしています。Windows はクライアント層やアプリケーション層で優れた威力を発揮する OS です。一方 Linux は、Web サーバやファイヤウォール、そしていくつかのエンタープライズ向けアプリケーションなど、エッジ用途やインフラ用途に強い OS です。そして HP-UX は、ミッションクリティカル分野で主流を占める OS であり、トータルなスケーラビリティや信頼性、高可用性、セキュリティの面で他を凌いでいます。

編集部：2000 年代の初め、次世代のコンピューティングを実現する上で、どのようなテクノロジーが求められていたのでしょうか？

ヘイズ氏：2000 年代の初めには、サーバ・コンソリデーションやサーバ仮想化が重要なテクノロジーとして認識され始めていました。複数の OS やアプリケーションを、いかにして互いに干渉させることなく 1 台のシステムにスタック(集約)していくか、検討を始めました。これは従来メインフレームが備えていた能力で、複数のアプリケーションを 1 台のメインフレームに効率的に集約可能でした。

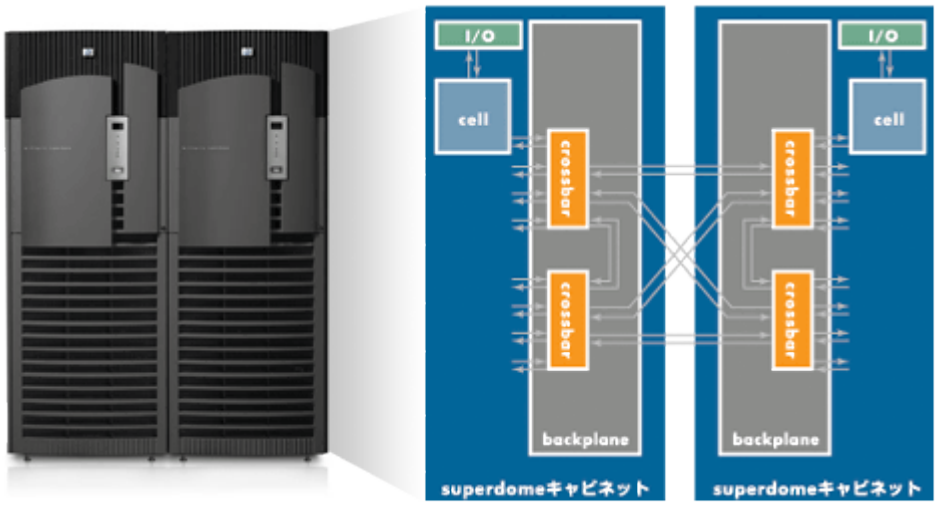
HPE では、こうしたサーバ仮想化の実現に向けて、nPars(nPartitions) や vPars(Virtual Partitions) 、WLM(Workload Manager) 、PRM(Process Resource Manager) 、psets(processor sets) といったテクノロジーの投入に注力してきまし

た。これらのテクノロジーにより複数のワークロードを互いに分離することで、1 台のサーバに多数のアプリケーションを集約したり、または nPars や vPars の利用により複数の OS を集約したりできるようになりました。

HPE では、さまざまな顧客ニーズに対応する多彩なサーバ仮想化技術をそろえた豊富なポートフォリオを提供してきました。ユーザの要望に応じて、アプリケーションの集約と OS の集約のどちらにも対応できます。例えば nPars では、電気的な分離までも実現するパーティション間の完全な隔離が可能です。そのため、いずれのコンポーネントに障害が発生しても、他の OS やアプリケーションに被害が及ぶことはありません。パーティションごとに個別にメンテナンスが可能です。IBM および SUN は、nPars に匹敵する隔離性を実現するに至っていません。vPars では、ソフトウェア・パーティショニングの技術ながらも殆どそのオーバーヘッドはなく、逸早く OS 環境間での CPU リソースの移動を実装しました。リソース最適化を求める商用環境に積極的に適用されている技術です。

2000 年から 2004 年にかけて、高度なスケーラビリティの達成がもうひとつの重要なテクノロジーとして認識されていました。システムが抱える負荷の規模が拡大するにつれ、ますます大規模なスケーラビリティが必要とされ始めていました。そうしたニーズに対応するため、64-way までスケールアップできるハイエンドサーバ Superdome をこの時期にリリースしました。こうした高レベルの SMP を HP-UX や主要なアプリケーションでサポートするための調整には時間を要しましたが、市場が必要とするタイミングに合わせて製品を投入することができました。

Superdome はとても効率的に設計されたシステムで、他ベンダーの ccNUMA アーキテクチャ製品とは異なり、メモリアクセスの遅延を低く抑えています。その後、Superdome は 128-way までサポートできるシステムにアップグレードされ、さらに ccNUMA 向けの最適化機能としてセルローカルメモリ(メモリアクセスをセル内部に制限する機能)をサポートすることで、大変多くのプロセッサ搭載した際の SMP パフォーマンスを強化しました。



superdome のシステム・アーキテクチャ

アダプティブ・エンタープライズを実現する HP-UX

現在のヘイズ氏は、仮想化技術やユーティリティ・コンピューティングに関連するインフラ製品およびソリューションの開発を担っている。そこで最後に、HPE のアダプティブ・エンタープライズのビジョンを実現していく上で取り組んだ UDC(Utility Data Center)や現在の VSE(Virtual Server Environment)による仮想化ソリューションがどのような役割を担うのか伺った。

編集部：アダプティブ・エンタープライズのビジョンとはどのようなものでしょうか？

また UDC とアダプティブ・エンタープライズはどのように関係するのでしょうか？

ヘイズ氏：アダプティブ・エンタープライズは、HPE が描くエンタープライズ IT のビジョンです。ビジネスと IT をうまくシンクロさせ、ビジネスの変化の波に乗ることをめざしています。そのために不可欠なことは、ビジネスニーズの変化に適応し、必要な場所に必要なリソースをリアルタイムに配分できる IT インフラの実現です。

UDC は、このアダプティブ・エンタープライズのビジョンを具体化した最初のソリューションのひとつであり、コンピューティングパワーをあたかも電気や水道のように提供することを目的としていました。プロセッサーパワーやストレージをオンデマンドですばやく切り出し提供でき、アプリケーションの状況に応じたリアルタイムのリソース配分や調整を目標としています。

UDC のリリースを通じて我々が学んだことは、アダプティブ・エンタープライズのビジョンをユーザが段階的に導入できる環境を整えることの重要性です。多くのユーザにとって、当初の UDC は敷居が高すぎました。そこで我々は、UDC のテクノロジーをいくつかに分解して提供することにしました。その成果は、HPE のコンサルティング＆インテグレーショングループの顧客向けソリューションで活用されています。

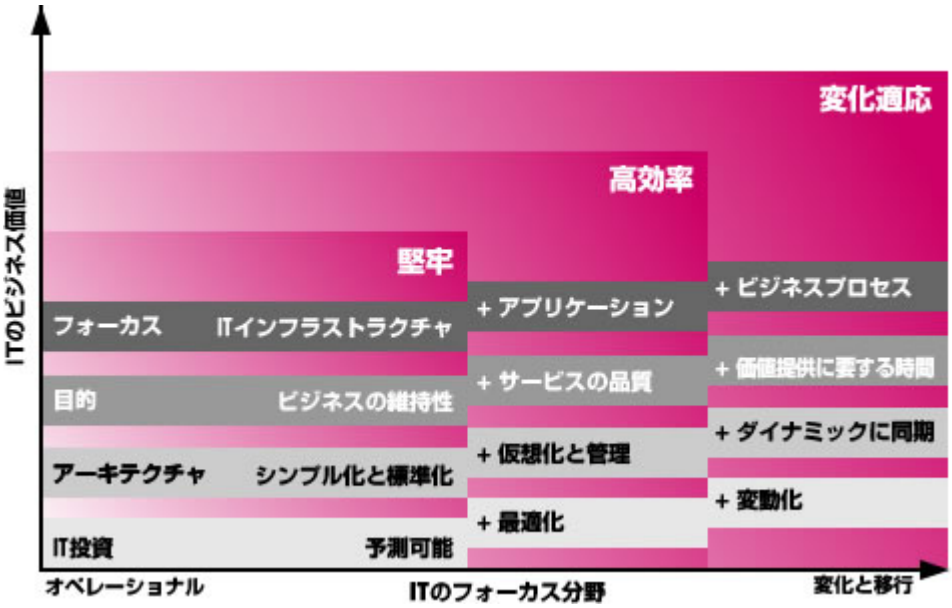
また UDC から得たフィードバックは、その後の製品開発にも反映されています。例えばこれから登場するいくつかの製品では、ユーザが簡単かつ段階的に各種機能を導入できるよう実装されています。

そして HPE では、データセンターのオートメーションや管理性の向上といったコンセプトをベースに、アダプティブ・エンタープライズのビジョンを実現する仮想化ソリューションとして、VSE（Virtual Server Environment）をリリースする。

編集部：アダプティブ・エンタープライズの実現に際して、仮想化技術はどのような役割を担うのでしょうか？また VSE(Virtual Server Environment)はどのように位置づけられるのでしょうか？

ヘイズ氏：このアダプティブ・エンタープライズの実現には、仮想化技術が重要な役割を担います。仮想化によって、リソースのプーリングや共有、そしてリソース利用率の最適化が可能になり、需要に合わせてリソースを自動的に供給できます。ワークロードの SLA(サービスレベル保証)に基づいて、需要のあるところにリソースを移動できるのです。その結果、ハードウェアコストやサポートコスト、ライセンスコストが低下し、TCO の削減が可能になります。

VSE は、アダプティブ・エンタープライズを実現するソリューションであり、Integrity サーバに対応したさまざまな仮想化技術を統合したものです。VSE では、WLM(Workload Manager) のようなワークロード管理ツールをはじめ、高可用性機能、サーバ仮想化、ユーティリティプライシングといった多彩な仮想化技術をひとつのソリューションとして統合しています。例えば HA クラスタにおいて、障害が発生したサ―バから他のサーバにワークロードを移動する局面では、移動先のサーバに十分なリソースが行きわたるようインテリジェントポリシーエンジンとなる WLM がリソース配分の再調整を行います。また、gWLM(Global Workload Manager)により仮想化された環境全体にわたってリソースを移動できるため、リソースを効果的にプールし、ビジネスニーズに基づいてさまざまなワークロードに対してそれを割り当てることが可能です。

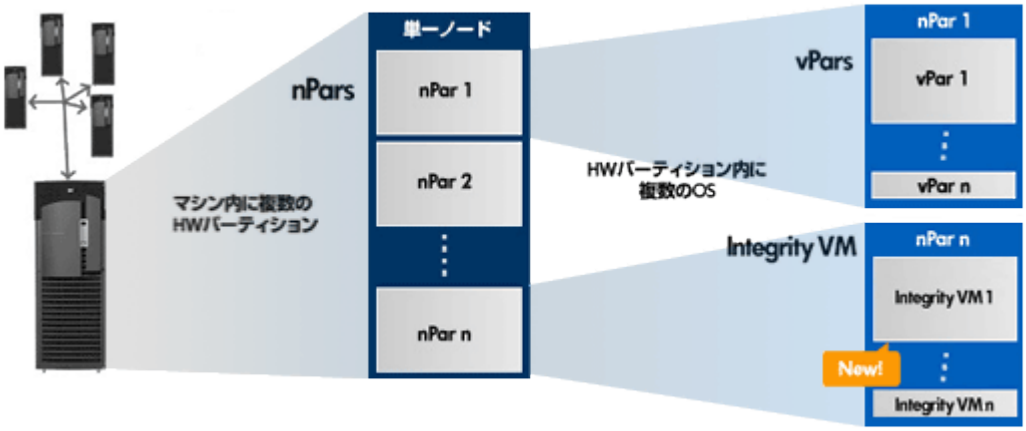


アダプティブ・エンタープライズ実現へのステップ

編集部：VSE の新製品 Integrity VM を投入した目的はどのようなものでしょうか？

ヘイズ氏：我々は以前から、既存の vPars を補完する、より粒度の細かい仮想化へのニーズを認識していました。vPars は業界初の商用 UNIX 向け仮想化ソリューションであり、競合ベンダーのさまざまな製品と比べてももっとも効率的な製品です。大半の HP-UX アプリケーションには vPars がもっとも適しています。

これに対し Integrity VM は、マルチ OS サポートや、CPU や I/O リソースのより粒度の細かな分割を必要とする用途に向けて開発されました。Integrity VM では、1 つの CPU や I/O リソースを複数に分割できるほか、マルチ OS をサポートします。ゲスト OS として HP-UX に対応した製品がすでにリリースされており、Windows および Linux に対応した製品は年内にリリースされる予定です。Integrity VM の導入により、vPars の機能に加えて、まったく新しい仮想化機能が利用可能になります。オーバーヘッドがきわめて小さく、粒度の大きな仮想化を求めるケースには、vPars が最適です。一方、HP-UX およびそのほかの OS をサポートし、粒度の細かな仮想化を求めるケースには、Integrity VM がより優れたソリューションを提供します。



HP VSE を構成する仮想化技術

Integrity VM の登場によってマルチ OS サポートや粒度の細かな仮想化にも対応可能になり、VSE はさらに幅の広い仮想化ソリューションへと進化した。一方、こうした進化によって多数の OS やアプリケーションが 1 台に集約されるようになると、ホストシステムの信頼性やスケーラビリティがきわめて重要な意味を持つようになる。ヘイズ氏は、それこそが HP-UX と Integrity サーバがアダプティブ・エンタープライズ実現のカギを握る理由であると説明する。

編集部：アダプティブ・エンタープライズにおいて、HP-UX と Integrity サーバはどのような役割を担うのですか？

ヘイズ氏：HP-UX は、アダプティブ・エンタープライズにおいて、ミッションクリティカル・ソリューションを支えるもっとも堅牢な環境を提供します。優れたスケーラビリティと信頼性、可用性を備え、仮想化機能と管理ソリューションを統合することで、メンテナンスを含む総合的なコストの削減を可能にする OS です。HPE では、この HP-UX がアダプティブ・エンタープライズの礎と考えています。

また VSE と Integrity サーバは、アダプティブ・エンタープライズのビジョンを実証する製品として、きわめて重要な役割を演じます。これらの製品は、ユーザにとってもっとも重要なミッションクリティカル・アプリケーションの受け皿となるエコシステムを形成します。マルチ OS に対応した Integrity サーバは、きわめて広範囲の OS 環境を収容できます。業界標準のサーバ環境を整え、ユーザにとってのソリューションの選択肢の幅を広げることで、非常に効果的な投資保護を実現します。

今回のインタビューを通じてヘイズ氏が語ってきたように、HP-UX の 20 年間の歴史は、実績豊富なスケーラビリティと信頼性、可用性、管理性を備えた、完成度の高い IT プラットフォームが生まれるまでの長いプロセスでもある。今後の HP-UX は、この盤石の基礎のもとに、nPars や vPars、PRM、WLM と gWLM、そして Integrity VM といったユニークな VSE のポートフォリオを生かしつつ、アダプティブ・エンタープライズの実現に大きく貢献するはずだ。またヘイズ氏は、HPE はさらにデータセンターオートメーションの分野にも投資を行い、仮想化されたデータセンターにおけるリソース配分を VSE 管理ツール上で自動化できる環境を実現していくと述べる。

最後にヘイズ氏は、HP-UX ユーザに向けた次のようなメッセージでインタビューを締めくくった。「エンタープライズ・コンピューティング分野をリードする UNIX として、HPE は今後も HP-UX に対して積極的に投資を続けていきます。これからの 20 年間もお客様のニーズに応えていけることを楽しみにしています」。

Virtualization Manager は「サーバ仮想化のポータル」をめざす

Virtualization Manager は「サーバ仮想化のポータル」をめざす・前編

2006 年 4 月　テクニカルライター 吉川和巳

HP-UX のサーバ仮想化技術「VSE(Virtual Server Environment)」のユニークな点は、それが多彩な仮想化技術の集合体であること。しかし、HPE にはそうした仮想化の“先駆者”ゆえの悩みもある。VSE の多岐に渡るサーバ仮想化技術の全体を俯瞰し、自由自在に使いこなすためには、「統合管理」の手段が不可欠となる。このニーズに応えるツールが、「Integrity Essentials Virtualization Manager」だ。同ツールは、データセンター内に存在するすべての Integrity サーバから自動的に構成情報を収集し、仮想環境と物理環境の最新のマッピング状況をグラフィカルに表示し、管理できる。

サーバ仮想化の先駆者ゆえの“悩み”とは

HP-UX のサーバ仮想化技術「VSE(Virtual Server Environment)」のユニークな点は、それが多彩な仮想化技術の集合体であることだ。VSE ではハードパーティションからソフトパーティション、仮想マシン、リソースパーティション、そしてワークロード管理まで幅広い仮想化ソリューションをそろえているため、それらの中からユーザの用途に最適なものを組み合わせて導入できる。これは、ブイエムウェア社の VMware やマイクロソフト社 Virtual Server、オープンソースの Xen など、他の仮想化ソリューションを大きく引き離すアドバンテージである。

しかしながら、HPE にはそうした仮想化の“先駆者”ゆえの悩みもあった。VSE の多岐に渡るサーバ仮想化技術の全体を俯瞰し、自由自在に使いこなすためには、「統合管理」の手段が不可欠となるのだ。

例えば VSE をすでに利用しているユーザの多くは、個々のパーティションに対してどの物理リソースを割り当てているか――すなわち「仮想環境と物理環境のマッピング」――を、手作業でドキュメント化したり、セルボードや CPU、I/O カードなどのパーティション配分状況を個別のツールを用いてその都度確認することもある。しかし、こうした手段で得られた情報は、往々にして最新の構成状況とは食い違ってしまうものだ。例えば、管理スタッフがリソース割り当てを変更したものの、忙しくてドキュメントの方は更新できず、結局いまの最新状況は誰も把握できていない……といった具合である。これでは、IT インフラ管理の効率化をもたらすはずの仮想化技術が、逆に現場のオペレーションの足を引っ張りかねない。

VSE のための統合管理ツール Virtualization Manager

この問題を解決するツールが、2006 年 4 月リリースされた「Integrity Essentials Virtualization Manager 」(以降、Virtualization Manager)だ。Virtualization Manager は、データセンター内に存在するすべての Integrity サーバや HP 9000 サーバから自動的に構成情報を収集し、仮想環境と物理環境の最新のマッピング状況をグラフィカルに表示、管理できるツールだ。また同時に各サーバや仮想環境の負荷状況もグラフ表示する。

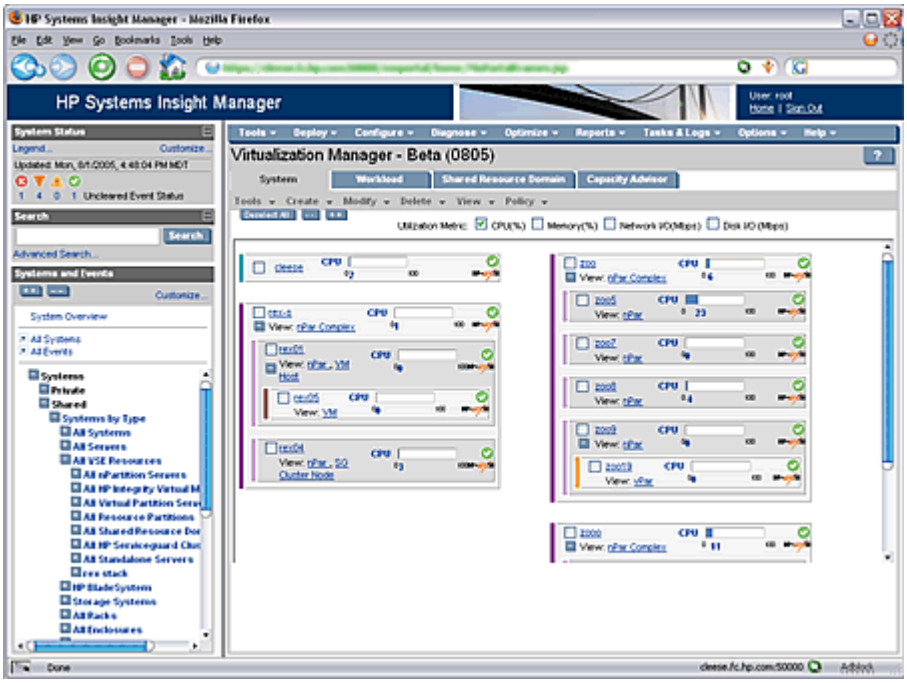


図 1：Virtualization Manager の画面表示例

図 1 は、Virtualization Manager の画面表示例である。同ツールは Systems Insight Manager(SIM) の一機能として統合されており、SIM のほかの機能やツールとスムーズに連携できるのが特徴だ。

ここで、Virtualization Manager の GUI 画面にはどのような情報が集約されているのか見てみたい。その前に、VSE を構成する各テクノロジーについて簡単におさらいしておこう。

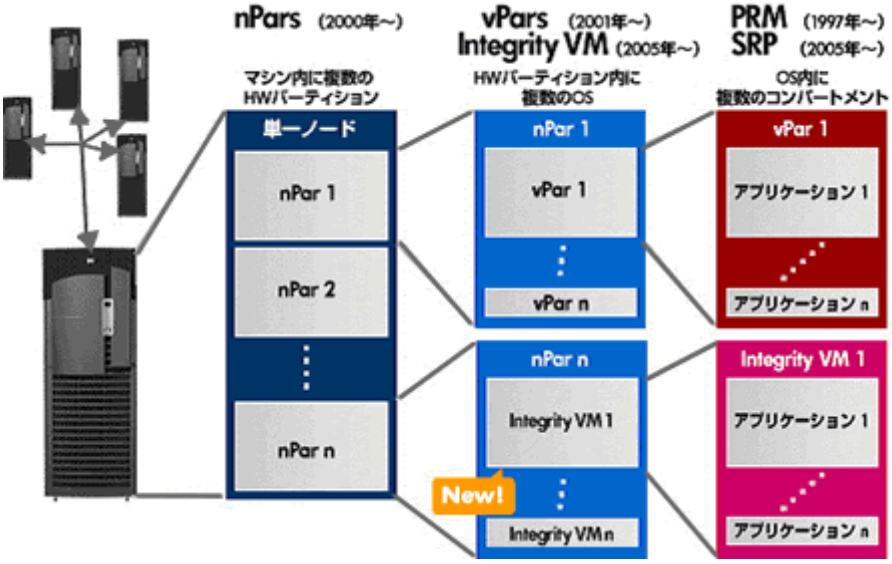


図 2： VSE を構成する仮想化技術

まず、ハードパーティションとしては nPartitions(nPars) を提供する。nPars は、サーバのハードウェア・リソース(CPU、メモリ、I/O など)を物理的にパーティション分割する技術であり、それぞれのパーティションが完全に独立したサーバとして動作可能となる。これにより、各パーティションが電気的に分離されているので OS レベルでの障害のみならず、ハードウェア障害の影響を個々のパーティションに封じ込めることができる。

もうひとつの仮想化技術である仮想パーティションとしては、Virtual Partitions(vPars) そして Integrity VM を提供する。いずれもサーバリソースをソフトウェアレベルで分割する技術であり、それぞれのパーティションや仮想マシンで独立した OS が動作する。OS レベルでの障害の影響を遮断できるほか、サーバをリブートせずにリソース配分をリアルタイムに変更できるのが特徴だ。

3 つめの仮想化技術であるリソースパーティションとしては、PRM(Process Resource Manager) および SRP(Secure Resource Partition) を提供する。これらは、1 つの OS のリソースを分割し、個々のアプリケーションに配分するテクノロジーだ。PRM では CPU やメモリ、I/O など一定のリソースを各アプリケーションに%単位で正確に配分でき、他のアプリケーションの過負荷による影響を防ぐことができる。さらに HP-UX 11i v2 の標準機能である Security Conteinment と統合した SRP では、それぞれのアプリケーションをセキュリティ的に完全に隔離しつつ、PRM の機能によりリソースの共有をすることが可能になる。

そして、これらの仮想化技術と連携するワークロード管理技術として、HP-UX Workload Manager(WLM) および Global Workload Manager(gWLM) を提供する。WLM と gWLM は、アプリケーションの負荷状況を監視しながら、個々のパーティションに対するリソース配分をリアルタイムに調節する。いわば VSE というオーケストラにおける指揮者のような存在だ。

後半では、Virtualization Manager の導入によってこれらの仮想化テクノロジーをどのように統括管理できるのか紹介しよう。

図 3 は、先ほど紹介した Virtualization Manager の画面例の中央部分を拡大した図である。

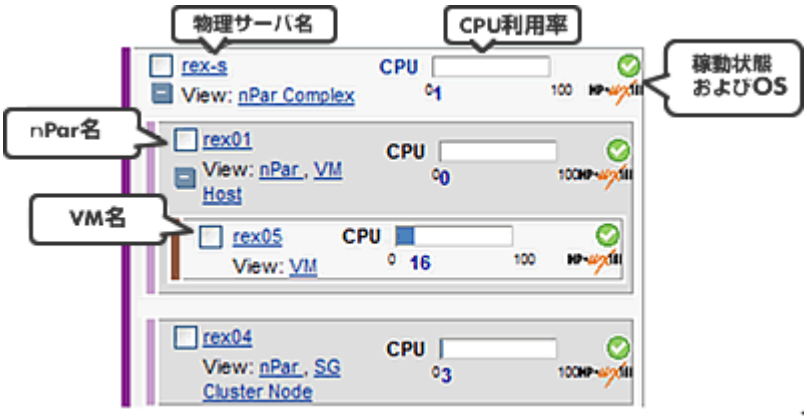


図 3 : Virtualization Manager によるシステムトポロジー表示例

ここで、もっとも外側を囲む領域全体が、1 台の Integrity サーバを表す。図 3 の例では、「rex-s」という名称のサーバであることがわかる。また画面上部には現在の CPU 利用率を示すバーグラフがあり、さらに右上には、稼働状態を示す緑のチェックマーク、そして稼働 OS を示すロゴマークが表示されている。

さて、さきほど図 2 で示したように、VSE を構成する nPars、vPars、Integrity VM といった仮想化技術は、それぞれを入れ子にして組み合わせることができる。この VSE のポートフォリオを反映して、Virtualization Manager の GUI 表示も入れ子構成になっている。例えば図 3 の例では、物理サーバ rex-s の内部には nPars「rex01」および「rex04」が存在する。また、nPars である rex01 の内部には Integrity VM「rex05」が動作している。

また先に述べたとおり、Virtualization Manager の特徴は SIM の他のツールとの連携機能である。例えば、図 3 の物理サーバ名 rex-s をクリックすれば、同サーバのハードウェア管理画面が開く。また CPU 利用率のバーグラフは、後述する「 Capacity

Advisor 」へのリンクになっている。同様に、nPars や vPars のパーティション管理画面、Serviceguard クラスタの管理画面、gWLM によるワークロード管理画面にもクリックひとつでジャンプできる仕組みになっている。

このように、Virtualization Manager は、VSE 全体の稼働状況を概観し、個々の管理ツールをローンチするための、いわば“サーバ仮想化のポータル”の役目を担っているのである。

VM Manager による Integrity VM 管理

VSE のポートフォリオの中でも、もっとも新しい仮想化技術として昨年末にリリースされたのが、Integrity VM である。Virtualization Manager では、この Integrity VM の仮想マシン(VM)を管理するためのツール「VM Manager」を無償で提供している。SIM で連携するのはもちろのこと、SMH (System Management Homepage とも連携可能だ) 。

例えば、図 3 の画面において Integrity VM のホスト名をクリックすると、以下のような VM Manager 画面が表示される。

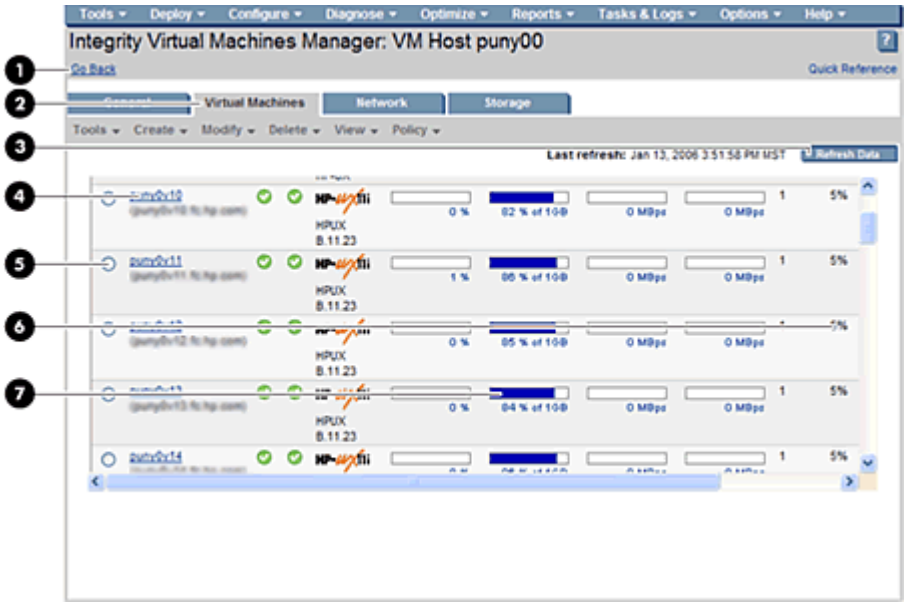


図 4 : VM Manager の画面例

1. Virtualization Manager 画面に戻るリンク
2. ホストサーバ上で動作するすべての VM を一覧表示するタブ
3. 画面内容を最新のものに更新するリンク
4. VM の詳細情報を表示するリンク
5. VM を選択し、さまざまなアクションを実行するためのラジオボタン
6. VM に割り当てられた CPU リソースを表すエンタイトルメント値
7. VM のリソース利用率

図 4 のように、VM Manager では各 VM の CPU やメモリ、ディスク、ネットワークなどのリソース使用率や、エンタイトルメント(リソース配分設定)を確認できる。

また、VM Manager はネットワークやディスクなどの I/O リソースがどのようにマッピングされているかも、GUI 表示可能だ。

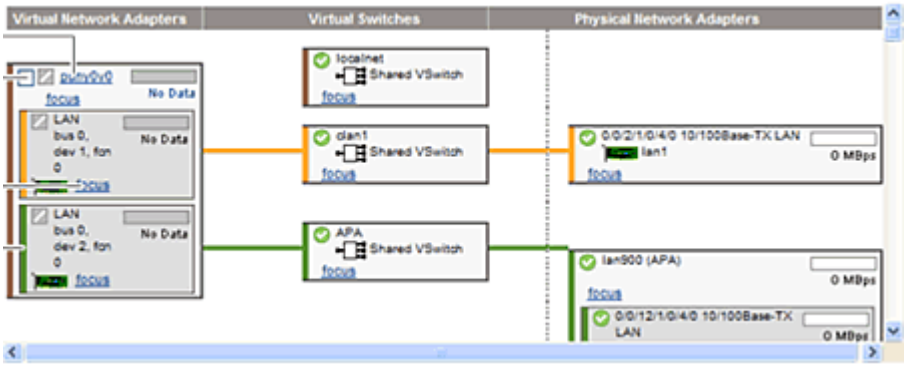


図 5：VM Manager によるネットワーク設定表示の例

図 5 の例では、画面左側の「仮想 NIC」(VM 内の仮想的なネットワークアダプタ)と、画面右側の「物理 NIC」(ホストが装備するネットワークアダプタ)が、画面中央の「仮想スイッチ」によってどのように“結線”されているかが図示されている。

このほか VM Manager では、物理ディスクと仮想ディスク間のマッピング表示や、VM の作成・起動・リセット・停止・削除といった構成作業を実施できる。

つづく 後編 では、VSE のためのもうひとつの仮想化管理ツールである「Capacity Advisor」について紹介する。

Virtualization Manager は「サーバ仮想化のポータル」をめざす・後編

2006 年 4 月　テクニカルライター 吉川和巳

Virtualization Manager に統合されたもうひとつの興味深いツールは「Integrity Essentials Capacity Advisor」である。これは、簡単に言えば「サーバ仮想化のシミュレーション」のためのツールだ。例えば、「サーバ仮想化によって CPU リソースの利用率はどの程度まで高められるか？」「パーティション間の CPU 配分を変更すると、どのような影響があるか？」「クラスタのフェイルオーバーが発生したとき負荷状況はどのように変化するか？」といった問いに、即座に答えを出すことができる。Capacity Advisor は、サーバ・コンソリデーションの際のアドバイスを得るためのツールと言えるだろう。

Capacity Advisor によるサーバ仮想化シミュレーション

Virtualization Manager に統合されたもうひとつの興味深いツールは、2006 年 4 月リリースされた「Integrity Essentials Capacity Advisor 」(以降、Capacity Advisor)である。これは、簡単に言えば「サーバ仮想化のシミュレーション」のためのツールだ。例えば以下のような管理者の疑問に対し、その場で答えを出すことができる。

- サーバ仮想化によって CPU リソースの利用率はどの程度まで高められるか？
- サーバ仮想化によって、既存のサーバ上に追加のアプリケーションを収容できるか？
- パーティション間の CPU 配分を変更すると、どのような影響があるか？
- 新しいアプリケーションはどのパーティションに配置すべきか？
- TiCAP(テンポラリー・インスタント・キャパシティ)の導入でどのようなメリットが得られるか？
- クラスタのフェイルオーバーが発生したとき負荷状況はどのように変化するか？
- gWLM の導入でどのようなメリットが得られるか？

図 1 は、Capacity Advisor にて CPU 利用率の履歴を表示した画面例である。

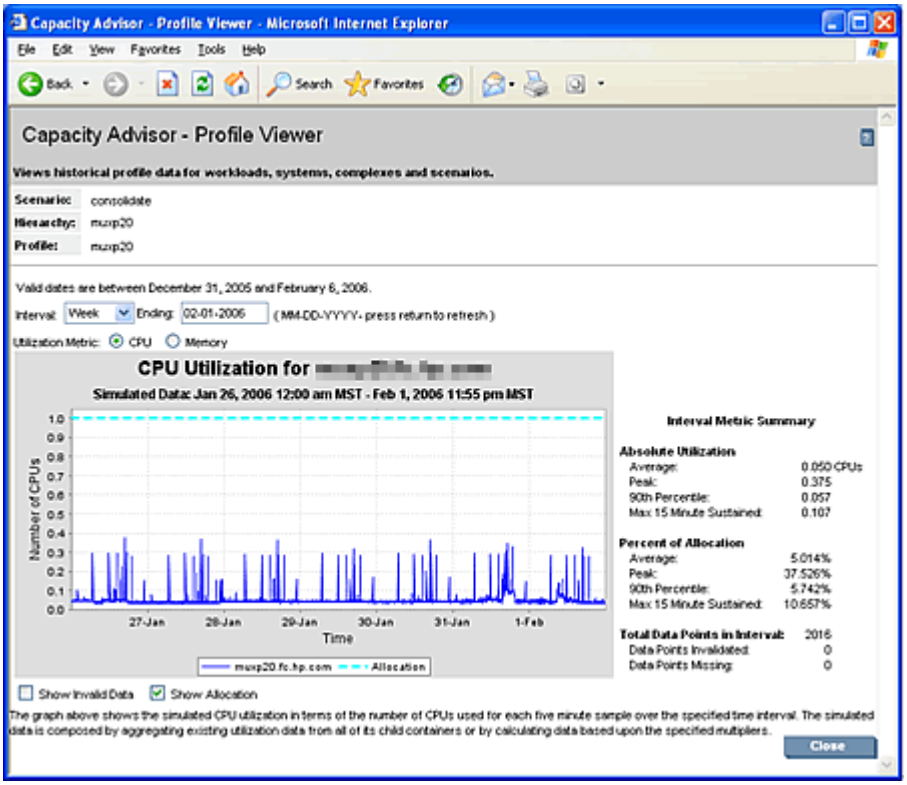


図 1：Capacity Advisor による CPU 利用率の履歴表示

Capacity Advisor で可能なこと

この Capacity Advisor によって具体的にどのような作業が可能なのか紹介しよう。

- 既存のワークロード情報のインポート - これまでのアプリケーション運用で計測されたワークロード情報のインポート
- ワークロード情報の編集 - 今後のワークロードのトレンド予測を反映させる
- 新しいワークロード情報の作成 - 新たに発生すると予想されるワークロード情報を手作業で入力する
- ワークロード移動時のシミュレーション
- VM 移動時のシミュレーション
- 新しいシステム構築時のシミュレーション(スタンドアロン構成や VM ホスト構成の作成)
- 既存システム変更時のシミュレーション(CPU の動作スピードや CPU 数、メモリ量の変更)

図 2 は、Capacity Advisor 上でワークロード情報の編集を実施している画面例である。

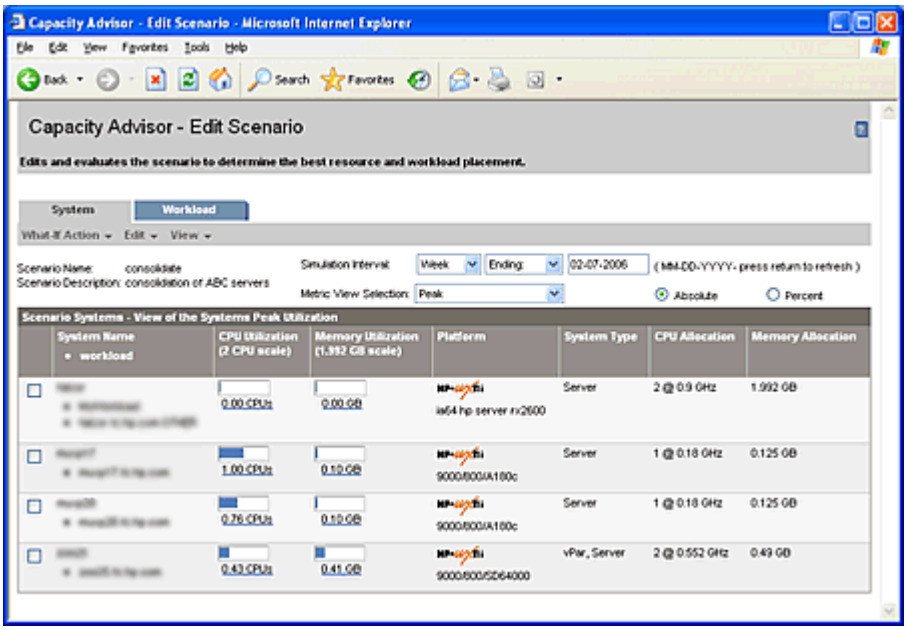


図 2：ワークロード情報の編集

では、Capacity Advisor による実際の仮想化シミュレーションの例を見てみよう。

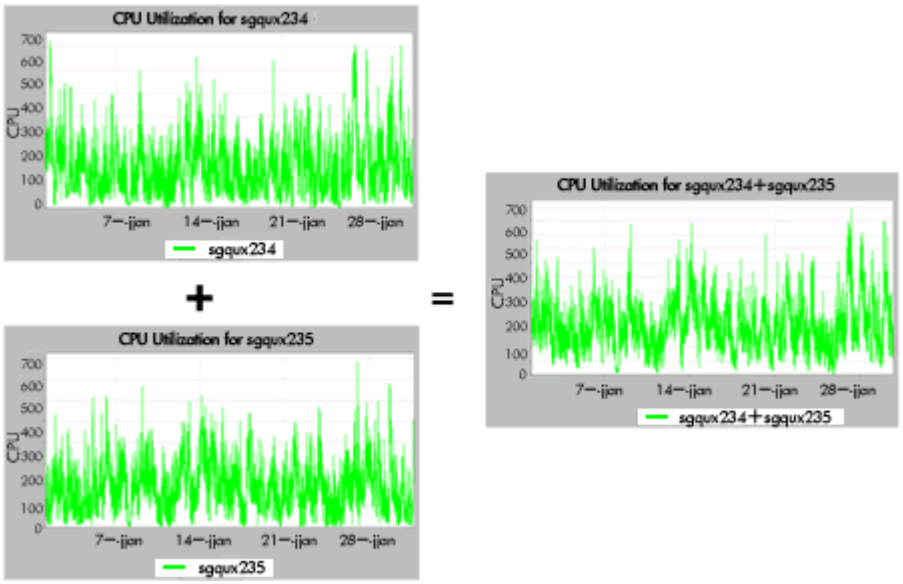


図 3：ワークロード移動時の仮想化シミュレーションの例

この例では、画面左側に示す 2 つのワークロードをひとつのマシン上にコンソリデーションした場合のシミュレーション結果が画面右側のグラフに表示されている。もともとのワークロードは、それぞれが 8 CPU 構成の独立したマシンで運用されていたもので、ピーク時負荷の数値は 700%～800% (CPU が 7～8 個分)に達している。しかし、この 2 つのワークロードをひとつに集約するシミュレーションを実施すると、画面右側のグラフが示すようにピーク時負荷は 1200% 以下に収まることがわかる。つまり、CPU 8 個＋8 個＝16 個ではなく、CPU 12 個分のサーバ・リソースで十分なのである。

このように Capacity Advisor によるシミュレーションによって、ワークロードやリソースのさまざまな組み合わせを事前に試すことが可能になる。サーバ・コンソリデーション際のアドバイスを得るためのツールと言えるだろう。
後半では、VSE を活用するためのベスト・プラクティスをいくつか紹介しよう。

VSE 活用のベストプラクティス

つづいて、こうして使いやすくなった VSE を活用するためのベスト・プラクティスをいくつか紹介する。

負荷の予測が難しいときは vPars を使う

多くのサーバで CPU の平均利用率を引き上げられない理由の多くは、負荷の予測が難しいことにある。例えばこれまで 1 時間に 100 件程度の処理を扱ってきたサービスに対し、広報活動などにより、今後は最大で 1000 件の処理が集中するとマーケティング部門が予測したとしよう。そうしたビジネス・レベルの予測は往々にして外れるとわかっていても、IT 部門としては 1000 件の処理に耐えるリソースを事前に配分せざるを得ない。こうして、いわゆるオーバー・プロビジョニング(過剰配分)が生じてしまう。

vPars は、こうした悩みの解決に最適なソリューションだ。vPars を使えばリアルタイムに CPU リソースを増強できるため、実際の負荷を処理するに十分なだけのリソースを配分しておけばよい。余ったリソースは開発環境やテスト環境、もしくは他のサービスに割り当てできる。これにより、負荷の予測が難しいビジネスであっても、オーバー・プロビジョニングの問題を回避可能だ。

サーバの代わりに vPars を購入する

ミッドレンジの Integrity サーバを必要とする 2 つのアプリケーションがあるとき、2 台のサーバを購入してそれぞれにアプリケーションを占有させるケースが一般的だ。しかし、それぞれのサーバをフル構成で導入することはめったにないだろう。2 台とも拡張性に余裕を残したまま運用されることになる。

こうしたケースでは、vPars により 2 つのアプリケーションを 1 台にコンソリデーションすることも可能だ。個々のアプリケーションにぴったりフィットする CPU パワーを配分できるほか、データセンターのハウジングコストや電源容量も 1/2 に抑えられる。なにより、Integrity サーバ 2 台分の運用コストより、1 台分＋vPars ライセンス方で運用した方が格段に安くつくのである。

負荷ピーク時のすばやい対応には gWLM を使う

vPars では CPU リソースの配分をリアルタイムに変更できるものの、そのためには管理者によるコマンド操作が必要だ。しかし管理者はサーバに常時張り付いているわけにはいかない。よって現実的には、数日〜数 10 日単位の負荷変動への対応が限度だろう。

しかし gWLM を使えば、管理者の手を借りずに、負荷状況に応じてリアルタイムに CPU リソースの割り当てを自動変更できる。したがって、負荷ピーク時にすばやい対応が必要なアプリケーションについては、gWLM の導入によって大きな効果が得られる。

gWLM を利用する際には、各 vPars のアプリケーション用途に応じて適切なポリシーを選択することが肝要となる。例えばパフォーマンスを一定に維持する必要のあるプロダクション環境については、最低限の CPU リソースを保証する設定が不可欠だ。一方、開発やテスト環境については、プロダクション環境の負荷に余裕があるときのみ CPU を借り受けるような設定にしておく。

アプリと DB 間で CPU リソースを共有する

OLTP を扱うアプリケーション・サーバと、その OLTP およびバッチ処理を扱うデータベース・サーバからなる典型的な n 階層システムでは、アプリケーション・サーバとデータベース・サーバのそれぞれを vPars で収容し、1 台の Integrity サーバの

CPU リソースを共有させる構成が推奨される。これにより、OLTP 負荷が高い日中にはアプリケーション・サーバに CPU を多く割り当て、バッチ負荷が高くなる夜間にはデータベースに CPU を配分するといったリソース共有が可能になる。とりわけ、SAP などの ERP アプリケーションではこうした使い方がマッチするはずだ。もちろん、こうした頻繁な CPU 配分変更には gWLM の利用が適している。

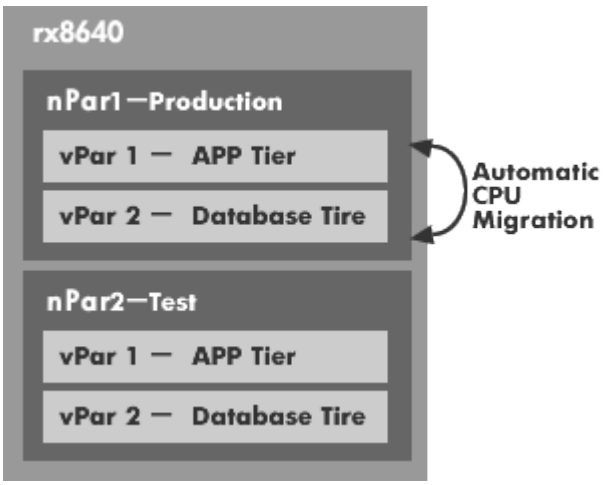


図 4：アプリと DB 間の CPU 共有

HA クラスタで iCAP を活用する

iCAP(インスタント・キャパシティ) とは、未購入の CPU をあらかじめ Integrity サーバに搭載しておくサービスである。負荷上昇やトラブル発生などで CPU 追加が必要になったときに瞬時に CPU リソースを増強できる一方で、必要になるまでは CPU のコストを支払わなくてもよいというメリットがある。

この iCAP は、Serviceguard による高可用性クラスタ(HA クラスタ)構成と組み合わせることでコスト削減に役立てることができる。図 5 は、ある顧客における iCAP の導入例である。

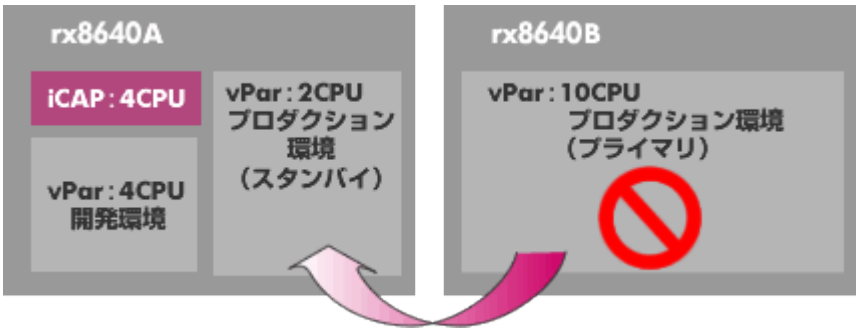


図 5：iCAP 導入例

この構成で注目していただきたいのは、2 台の rx8640 のうち「A」の方には iCAP の CPU が 4 個搭載されている点だ。ここで、「B」の vPar 上で動作するプロダクション環境がダウンすると、Serviceguard のフェイルオーバーが発生し、「A」のスタンバイ環境が起動することになる。しかし、同環境の vPar には 2 CPU しか割り当てられておらず、もともと 10 CPU 分を消費していた負荷の肩代わりは難しい。

こうしたとき、iCAP による CPU 増強が威力を発揮する。iCAP 分としてあらかじめ装着されていた 4 CPU 分を追加し、さらに開発環境の 3 CPU 分も合わせて移動することで、合計で 9 CPU 分を瞬時に確保することができる。このように、HA クラスタの構成に iCAP を導入することで、スタンバイ側の設備コストを最小限に抑えることが可能になる。

以上、今回は Virtualization Manager による VSE 統合管理について説明した。ここまで見てきたとおり、Virtualization Manager という統合管理ツールを得たことで、VSE のポートフォリオの豊富さが一段と魅力を増したことがご理解いただけるだろう。

VSE 管理ツールを始めとする主要コンポーネントは VSE Suite として提供されている。また、HP-UX の環境さえあれば、これらを無償で試すことができる。

新しい Integrity サーバの魅力を探る

新しい Integrity サーバの魅力を探る・前編

2006 年 5 月　テクニカルライター 吉川和巳

例えばクルマの性能は、エンジンの能力だけでは語れない。ギヤボックスやサスペンション、ブレーキなどの「足まわり」にも相応の強靭さが求められる。4 月発表の新しいハイエンドおよびミッドレンジの Integrity サーバの魅力は、独自開発の「sx2000 チップセット」を搭載し、今後インテル®から登場するデュアルコア Itanium®プロセッサの真価を引き出せること。次世代レベルのキャパシティを先取りして実装することで、プロセッサや I/O の交換でシステム全体の性能を高められる、「買い換えせずとも中身が進化するサーバ」なのである。生まれ変わった Integrity サーバの技術革新について、sx2000 チップセット開発担当者に話を聞いた。

2 世代先まで見越した sx2000 チップセット

例えばクルマの性能は、エンジンの能力だけでは語れない。高性能のエンジンの動力を余すことなく路面に伝え、そのポテンシャルをフルに引き出すには、ギヤボックスやサスペンション、シャーシ、そしてブレーキなどの「足まわり」にも相応の強靭さが求められる。今回発表された新しいハイエンドおよびミッドレンジの Integrity サーバの魅力、それは HPE が独自開発した sx2000 チップセット(以下 sx2000)という強靭な「足まわり」を獲得したことにある。

図 1：HPE が新開発した sx2000 チップセット

sx2000 の開発を指揮した HPE のハウエル・フェルゼンタール氏は、同チップセット開発の背景を次のように説明する。「sx2000 の開発目標は、今年後半にインテルから提供されるデュアルコア Itanium プロセッサをはじめ、進化し続けるメモリ標準規格や I/O カードの能力にふさわしい高パフォーマンスを提供すること。そのためには、メモリや I/O、クロスバーといったインフラの強化が不可欠でした」

sx2000 は、いわば「デュアルコアの真価」を引き出すべく生み出されたチップセットだ。従来のチップセット「sx1000」に比べ、プロセッサバスの帯域が 33%向上したほか、メモリ帯域は 2.1 倍、I/O 帯域は 4.4 倍、そしてセルボード間を結ぶクロスバーの帯域は 4.2 倍にそれぞれ大幅強化された。

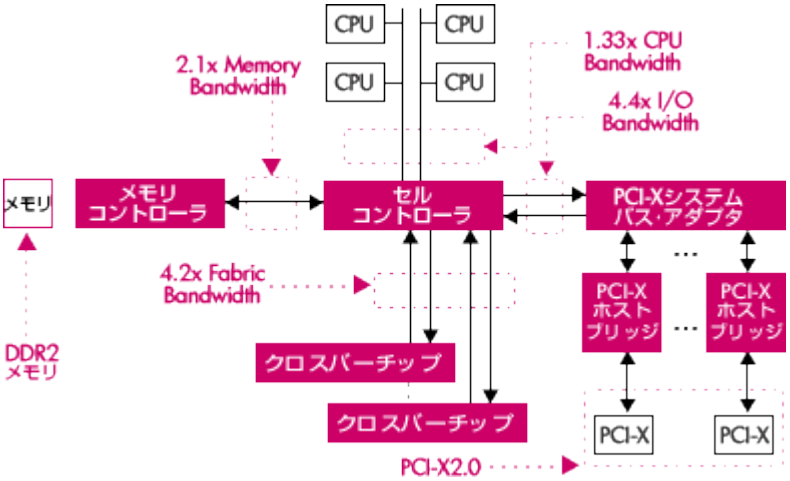


図 2： sx2000 チップセットによる帯域強化のポイント

こうした徹底的な帯域強化は、やみくもなパフォーマンス向上だけを目指したものではない。長期間にわたって Integrity サーバの「中身」を進化させ、買い換えをできるだけ不要にすること――すなわち投資保護こそ、sx2000 の最終目標と言えるだろう。

クルマとは違い、コンピュータではエンジンに相当するプロセッサだけを容易に入れ替えできる。よって次世代レベルのメモリ帯域や I/O 帯域を先取りして実装しておくことで、同じシャーシやセルボードを何年も継続して使いつつ、ビジネスの成長やプロセッサの技術革新に合わせてシステム全体のパフォーマンスを引き上げられる。日本ヒューレット・パッカードのテクニカルコンサルタント白井泰博氏は、「sx2000 は、今年後半に登場するデュアルコア インテル®Itanium®プロセッサの帯域をフルに生かせるよう最適化されています。また今後インテルが投入するマルチコア・プロセッサを含めて 2 世代に渡ってプロセッサ交換のみで対応できます」と述べる。

またプロセッサに限らず、I/O についても今後の数年間を見据えた設計が施されていると、フェルゼンタール氏は説明する。「1 枚のカードで 2 倍の帯域（2GB/s）を占有できる新しい PCI-X 2.0 バスを今回新たに採用しました。これにより 10G Ethernet や 4G FibreChannel といったこれから普及する最新の標準インタフェースをフルスピードでサポートできます。また今後は、同じシャーシ上で PCI Express バスがサポート可能になる予定です」。つまり sx2000 を搭載した新しい Integrity サーバは、いわば「買い換えせずとも中身が進化するサーバ」なのである。

サーバの買い換えを極力抑え、モジュール交換だけで性能向上を持続していくには、チップセットのみならずシャーシや電源の設計にも、あらかじめ余裕を持たせておく必要がある。実のところ sx2000 ベースへのアップグレードに際しては、従来の Integrity サーバの同じ筐体をそのまま使うことができる。つまりシャーシ全体の買い直し（いわゆるフォークリフト・アップグレード）は不要だ。この点について白井氏は「Integrity サーバの優位性のひとつは、将来を見通したシャーシ設計が当初より施されている点にあります。同じシャーシを何年も使いながら、企業の成長に合わせてシステムの中身も成長させることができます」と説明する。

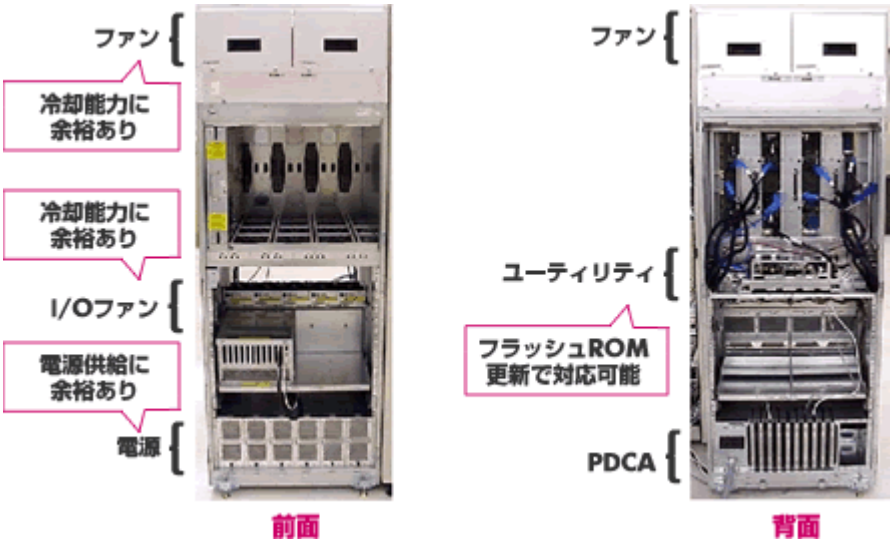


図 3： Integrity Superdome の余裕を持った筐体設計

後半では、この新しい Integrity サーバにおいて刷新された DDR2 対応のメモリシステムにクローズアップしたい。

メモリシステムを DDR2 対応に刷新

ここで、新開発された sx2000 の構成を簡単に紹介しておこう。

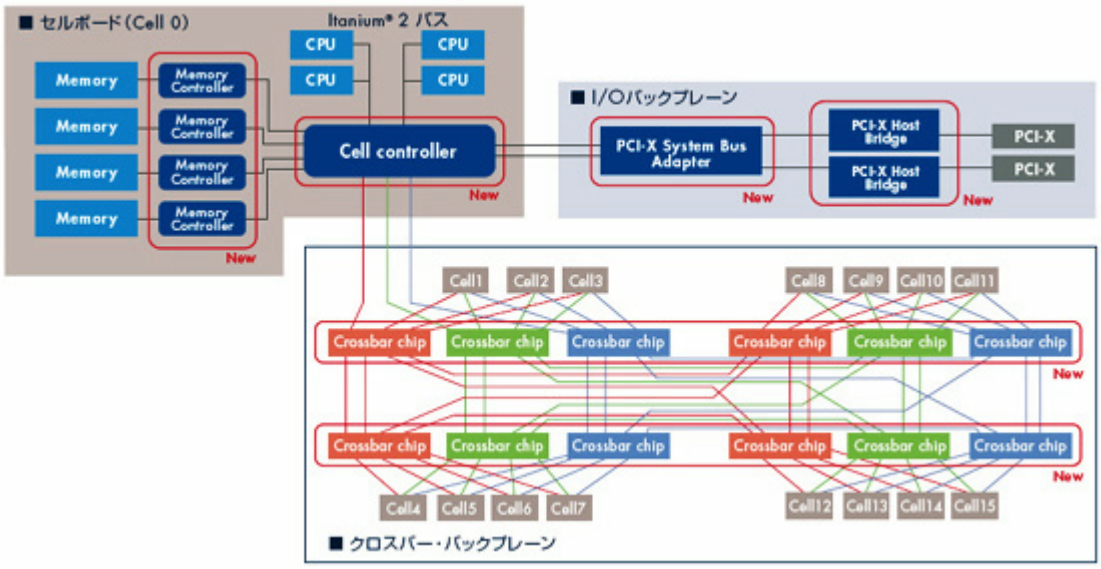


図 4： sx2000 チップセットのブロックダイアグラム

図 4 において、赤く囲まれた部分が sx2000 を構成する新しいコンポーネントである。それぞれの役割と新機能を以下に示す。

＜表： sx2000 チップセットを構成するコンポーネント＞

コンポーネント	役割	新機能
セル・コントローラ	プロセッサ、メモリ、I/O、セル間通信を統括	• プロセッサバス帯域が 1.33 倍 • メモリ帯域が 2.1 倍 • I/O 帯域が 4.4 倍 • クロスバー帯域が 4.2 倍
メモリ・コントローラ	メモリ・モジュールへのアクセスや冗長化	• DDR2 メモリのサポート • ダブルチップスペア
クロスバー・チップ	セルボード間通信	• HSS によるセルボード間接続の冗長化と帯域拡張
PCI-X システム・バス・アダプタ	セル・コントローラと I/O バックプレーン間の接続	• HSS による I/O 接続の冗長化と帯域拡張
PCI-X ホスト・ブリッジ	I/O コントローラ	• PCI-X 2.0 のサポート

これらのコンポーネントに加えて、新しい Integrity サーバでは MP（Management Processor）やシステムクロック、DCDC コンバータなどの部分で信頼性や機能の強化が施されている。ちなみに sx2000 が導入されるのは、Integrity Superdome をは

じめ、rx8640、rx7640 といったセルベースのハイエンド／ミッドレンジモデルである。今年 4 月のリリースと同時に出荷が開始されている。

DDR2 で構築された巨大メモリシステム

ハウエル・フェルゼンタール氏

sx2000 が新たに獲得した能力は、新しいプロセッサのサポートに限らない。「確かにデュアルコア・プロセッサへの対応は、sx2000 の開発を後押しした大きな要因です。しかし同時に、メモリサイズの巨大化とともにメモリシステムの信頼性をいかに維持するかという課題もありました。次の世代のメモリシステムが求められていたのです」（フェルゼンタール氏）。そこで sx2000 では、いくつかの新しいテクノロジーの投入によってメモリシステムが刷新されている。

そのひとつは、DDR2 SDRAM（以降、DDR2）のサポートだ。「メモリシステムを刷新した理由のひとつは、DDR2 のサポートです。従来の Integrity サーバに搭載された sx1000 では DDR2 に対応しておらず、通常の SDRAM しか利用できません」（フェルゼンタール氏）。

DDR2 とは、従来の Integrity サーバのメモリシステムに採用されてきた通常の SDRAM を高速化したメモリバスの標準規格である。具体的には、SDRAM ではメモリクロックが 100MHz、最大転送速度は 0.8GB/s であるのに対し、DDR2 ではメモリクロック 266MHz、最大転送速度は 4.3GB/s に達する。クロックを 2.7 倍に高速化したほか、クロックの立ち上がりと立ち下がりの両方でデータを転送する（ゆえに Double Data Rate と呼ばれる）ことで、トータルで 5 倍以上の帯域拡張を実現する。

これがメモリ・モジュール（DIMM）を 4〜8 枚程度しか装着しない PC サーバであれば、マザーボード全体でのメモリ帯域は 4.3GB/s の 1〜2 倍程度に収まり、既製のチップセットでもメモリシステムを実装できる。これに対し Integrity Superdome では、セルボード 1 枚あたり 32 枚、システム全体では 512 枚の DIMM を装着可能だ。これによりメモリサイズは 2TB（年内サポート予定）まで拡張可能で、さらに今後 2G ビット DRAM の供給開始とともに 4TB という文字通り広大無辺なメモリ空間がサポート可能になる。

メモリクロックが大幅に高速化された DDR2 をベースに、この巨大なメモリシステムをいかにして構築するかが、sx2000 開発のひとつの難関であったとフェルゼンタール氏は振り返る。「DDR2 をベースに大規模なメモリシステムを設計するのは容易ではありません。DDR2 はきわめて高速なメモリバス規格であるため、それを大規模に集積する上では信号処理や電源供給などの電子設計で苦労しました」。実際 sx2000 では、8 つのメモリ・コントローラとセル・コントローラの間を 4 本の DDR2 メモリバスで接続し、4.3GB/s×4＝合計 17.2GB/s という PC サーバとはけた違いのメモリ帯域を 1 枚のセルボード上で確保することに成功している。

このように業界標準テクノロジーをうまく活用しつつも、並大抵のサーバベンダーにはまねのできない圧倒的なスケーラビリティを実現するのは、ミッションクリティカル分野で長年の実績を積み重ねてきたの“お家芸”と言えるだろう。「DDR2 のような業界

標準のメモリ技術に対応することで、メモリ・モジュールのコストを抑えつつ、これからのメモリの高集積化による恩恵も享受できます」とフェルゼンタール氏は述べる。

さらに sx2000 のメモリシステムでは、DDR2 のサポートに加えて、「ダブルチップスペア」と呼ばれるまったく新しいエラー訂正技術が投入されている。後編では、このダブルチップスペアをはじめ、新しい Integrity サーバに備わる「自律的な障害復旧」の能力にスポットを当てる。

新しい Integrity サーバの魅力を探る・後編

2006 年 5 月　テクニカルライター 吉川和巳

新しい Integrity サーバのもうひとつの魅力は「フェイルセーフ」の能力、つまり「障害時にもシステムダウンを極力回避できる力だ。例えば新開発のエラー訂正技術「ダブルチップスペア」の投入により、合計 3 個の DRAM が故障しない限りシステムダウンは発生しないメモリシステムが実現された。またクロスバーや I/O とセルボードを結ぶ高速シリアルリンクでは、エラー発生時にもデータの再送信を行う「リンクレベルリトライ」やリンク障害時の縮退動作といった自己修復機能により、システムはあくまでも継続運用を試みる。ミッションクリティカル・サーバに集約された多彩なノウハウや知恵を紹介する。

sx2000 チップセットのフェイルセーフ能力とは

ハイエンドおよびミッドレンジの新しい Integrity サーバに加わったもうひとつの魅力、それは「フェイルセーフ」の能力である。これはすなわち、「障害時にもシステムダウンを極力回避できる力」と表現できるだろう。

新開発の sx2000 チップセット(以下、sx2000)を搭載する Superdome や rx8640 、rx7640 といった Integrity サーバの上位機種は、nPars（nPartitions） や vPars（Virtual Partitions） 、Integrity VM など HPE の仮想化技術を用いたサーバ・コンソリデーションに利用されるケースが少なくない。しかし多数のアプリケーションを集積すればするほど、その土台となるサーバのリブートは原則的に許されなくなる。つまり求められる可用性も、一般的なサーバに比べて段違いに高いものとなるのだ。そこで sx2000 では、万が一のトラブル時にもシステムダウンを極力回避し、システム運用を自律的に継続できる機能が数多く盛り込まれた。

そのひとつが、sx2000 のメモリシステムに導入された「ダブルチップスペア」と呼ばれる、画期的と言ってよいエラー訂正技術の導入である。

メモリエラーとエラー訂正技術

周知のとおり、メモリ上に記録されたデータは、メモリエラーの発生によって失われるリスクが常につきまとう。メモリエラーは、大きく分けて「ソフトエラー」と「ハードエラー」の 2 種類に分類できる。

このうちソフトエラーは、単発的に発生するエラーであり、宇宙線が DRAM 内部のメモリセルに衝突することで発生する。実際、分厚い岩盤の中ではソフトエラーは発生せず、高度 3,000m では 10 倍に増えるという。ソフトエラーが起きると通常 1 ビット～数ビットのデータが失われ、メモリ集積度とともにその発生確率も上昇する。一般的には、1GB のメモリ（非 ECC）を搭載した PC の場合、ソフトエラーによるシステムダウンの発生確率は 1 年で 7～8 割程度と言われている。

一方、ハードエラーは、DRAM そのもののハードウェア障害によって発生する。その確率はソフトエラーよりも大幅に低いものの、ゼロにすることはできない。ハードエラーが発生すると、場合によっては 1 個の DRAM 全体のデータが失われる。

これらのメモリエラーに対処するため、これまで以下のようなエラー検出・訂正技術が利用されてきた。

- パリティ付きメモリ
- ECC（Error Correction Code）メモリ
- チップスペア／チップキル

ご存じのとおりパリティ付きメモリとは、通常のデータに加えてパリティビットを DIMM 上に保持することで、1 ビットエラーの検出を行う。しかしこうしたパリティチェックではエラー訂正までは行えず、ソフトエラー発生時にはシステムダウンが免れないため、最近ではあまり用いられていない。

一方、PC サーバ製品の大半が実装する ECC（Error Correction Code）メモリは、64 ビットのデータに対して 8 ビットの冗長ビットを付加することで、1 ビットエラーの訂正と 2 ビットエラーの検出を可能にしたものである。ECC メモリを導入することでソフトエラーによるシステムダウンの確率は低くなるものの、ハードエラーの発生時には 4 ビットや 8 ビット（DRAM の 1 ワード）単位でデータが失われる可能性が高く、ECC によるエラーの訂正や検出が不可能になる。

そこで従来の Integrity サーバのすべてのモデルをはじめ、ベンダー各社のハイエンド／ミッドレンジサーバでは、チップスペアもしくはチップキルとも呼ばれるエラー訂正技術が採用されている。チップスペアでは、データを 1 ビットずつ異なる DRAM に分散して保存することで、DRAM 全体が故障しても「大量の 1 ビットエラー」として扱うことで ECC による訂正が可能になる。これはちょうど RAID 構成によるハードディスクの冗長化に似たメカニズムと言える。

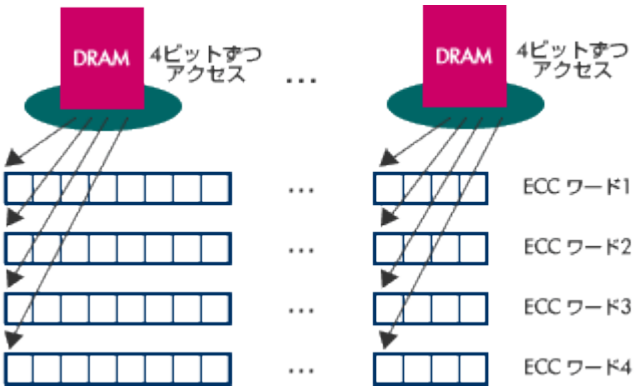


図 1：チップスペアのメカニズム

「ダブルチップスペア」のねばり強さ

ダブルチップスペアは、ECC のアルゴリズムに HPE が独自の改良を加えることで実現した、まったく新しいエラー訂正技術である。簡単に言えば、合計 3 個の DRAM が故障しない限りシステムダウンしないというメカニズムだ。以下の表は、既存のエラー訂正技術とダブルチップスペアを比較した表である。

＜表：エラー訂正技術によるエラー対処の違い＞

エラー規模	パリティ	ECC	チップスペア／チップキル	ダブルチップスペア
1 ビット	検出（ダウン）	訂正	訂正	訂正

2 ビット	検出不可	検出（ダウン）	訂正	訂正
DRAM×1 個	検出不可	検出不可	訂正（即交換）	訂正（要交換）
DRAM×2 個	検出不可	検出不可	検出（ダウン）	訂正（即交換）
DRAM×3 個	検出不可	検出不可	検出不可	検出ダウン

従来のチップスペアでは、1 個の DRAM の障害まで耐えることができる。しかしこれでも可用性の面では不十分であると白井氏は説明する。「1 つの DIMM 上で DRAM が 1 個故障しチップスペアリングの機能によって継続運用されている状況で、さらに 1 ビットエラーが発生しただけでシステムがダウンしてしまいます。クラスタ環境であればフェイルオーバーで復旧されますが、一時的にパフォーマンスの低下やフェイルオーバが完了するまで業務中断してしまうこともあり得ます。従いましてチップスペアリングの機能では DRAM が 1 個つ故障後には、すみやかにシステムを止めて DIMM を交換しておく必要がありました。」（白井氏）

これに対しダブルチップスペアに対応した sx2000 ベースの Integrity サーバでは、こうした緊急対応は不要になるとフェルゼンタール氏は説明する。「ダブルチップスペアでは、1 個の DRAM が故障しても ECC メモリと同等の信頼性を確保できるため、1 ビットエラーが発生してもシステムは継続運用できます。よって DIMM の交換は計画停止のタイミングで実施すれば十分です」

事前に計画されたシステム停止と、緊急メンテナンスによるシステム停止では、サービスや業務に対する影響はまったく異なってくる。このようにダブルチップスペアは、“もしも”の事態でも可能な限り継続運用できる、「ねばり強い」メモリシステムと言えるだろう。つづく後半では、sx2000 から新たに投入された高速シリアルリンク技術、「HSS」がもたらす高可用性に注目してみたい。

スピードと可用性を両立した高速シリアルリンク

前編で述べたとおり、sx2000 ではセルボード間を結ぶクロスバー帯域が 4.2 倍、I/O 帯域も 4.4 倍に引き上げられている。この帯域拡張を可能にした新技術が、HPE が開発した高速シリアルリンク HSS（High Speed Serial link）である。「sx2000 では、クロスバーとセルボード間、そして I/O サブシステムとセルボード間を結ぶ手段として、従来のパラレル接続より大幅に高速な HSS を新たに開発しました。11.5GB/s というきわめて高帯域のシリアルリンクをゼロから設計し、それを VLSI チップに組み込むのは容易ではなく、sx2000 開発で苦労した点のひとつでした」とフェルゼンタール氏は語る。

クロック同期式のパラレル接続によるデータリンクでは、ギガビットレベルへの高速化は困難である。なぜならこのスピードでは信号のひずみやスキュー（到達時間のずれ）が大きくなり、複数ビットのデータ信号をクロックに同期して読み取るのが難しくなるからだ。一方、高速シリアルリンクでは、SerDes（シリアライザ／デシリアライザ）と呼ばれる機構により複数ビットを 1 ビットデータの並びに変換し、さらに 8B/10B エンコーダや CDR（Clock Data Recovery）によりクロックとデータを同じ 1 本の信号に織り込む。これによりスキューやひずみの問題を解消しつつ、ギガビットレベルのデータリンクを実現できる仕組みだ。

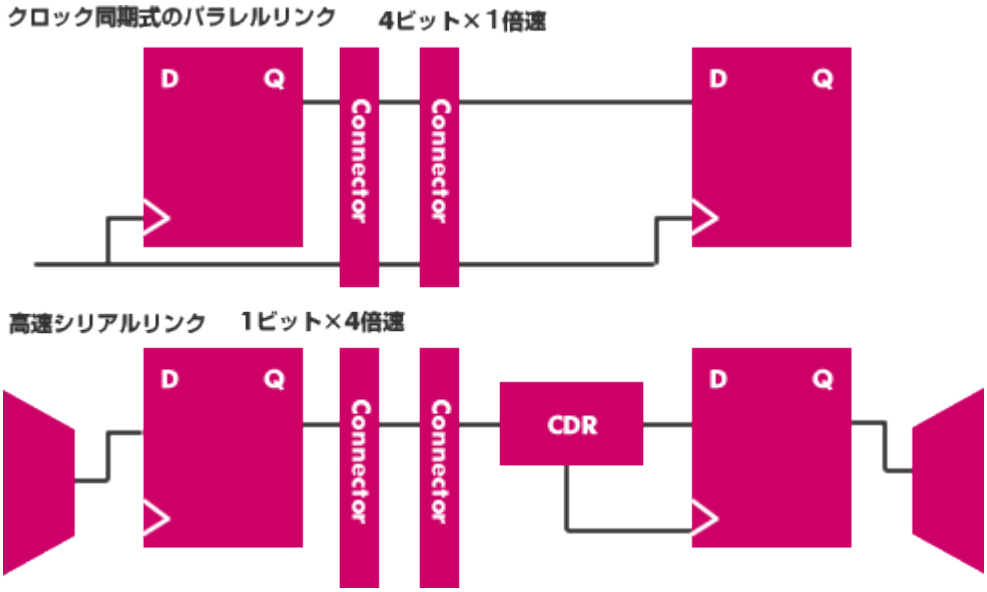


図 2：高速シリアルリンクのメカニズム

新しい Integrity サーバでは、クロスバーおよび I/O サブシステムとの接続手段として、この 11.5GB/s の HSS を用いている。さらにセルボードとクロスバー間を 3 本の HSS で負荷分散することで、セルボード 1 枚あたり 11.5GB/s×3＝合計 34.6GB/s という帯域を達成している。前編で説明したとおりセルボード内部のメモリ帯域は 17.2GB/s であり、異なるセルボードへのメモリ・アクセスでもローカル・メモリと同様のスループットが得られる設計である。「HSS の 2 重化では帯域が足りず、4 重化は物理的に実装が難しい。そこで 3 重化という設計が選択されました」（白井氏）。またクロスバーの遅延についても、従来の sx1000 に比べて 2〜3 割短縮されているという。

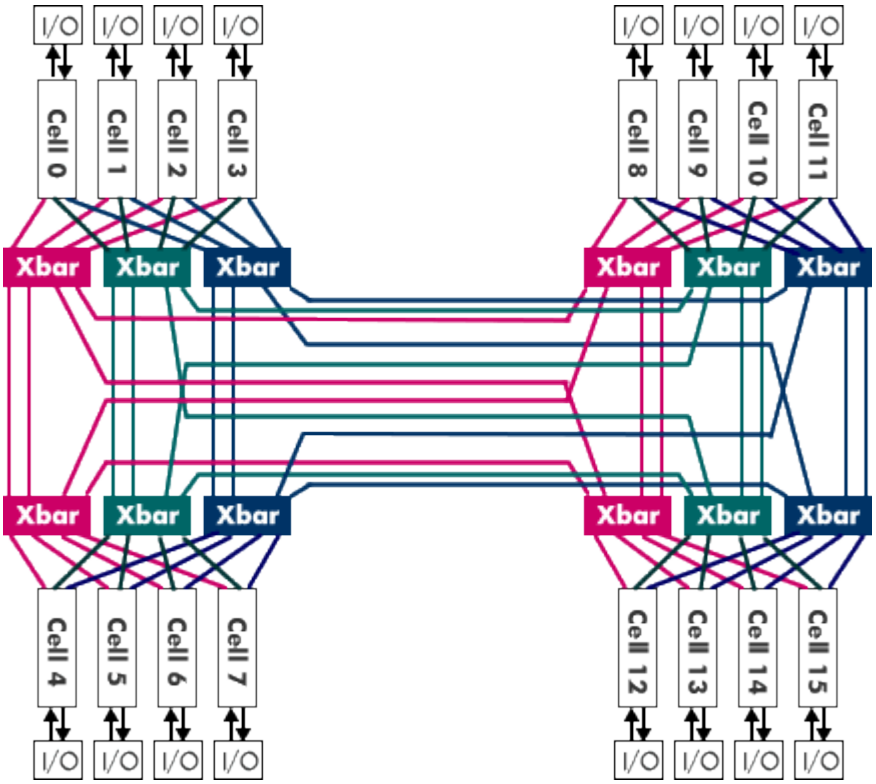


図 3：3 本の HSS によるクロスバー接続

クロスバーがもたらす可用性

さらに、この HSS によるクロスバー接続は、帯域向上と同時に可用性向上も実現しているとフェルゼンタール氏は説明する。「HSS ではリンクレベルリトライと呼ばれる自己修復機能を実装しています。データ伝送時にエラーが検出されると、そのデータが再送信され、システムダウンを防ぎます。またクロスバーに接続された 3 本の HSS のうち 1 本が完全に切断されても、システムは残る 2 本だけで継続運用する仕組みです。さらに万が一クロスバーチップに障害が起きた場合、以前であればチップを交換するまでシステムを停止する必要がありましたが、sx2000 では自動的に障害チップを迂回して継続運用できます」（フェルゼンタール氏）。こうした冗長化設計によって、Superdome ではシステム全体の SPOF（単一障害点）がほとんど排除されているという。

こうした Integrity サーバのクロスバー・メカニズムは、nPars によるパーティション分割で威力を発揮する。システム全体をひとつの大きな SMP として構成するサーバとは異なり、Integrity サーバではシステム全体のリソースをセルボード単位で区切り、複数の nPar に割り当て可能だ。「異なる nPar 間はクロスバーによって電気的に完全に隔離され、ある nPar の障害が他に影響することはありません。複数の nPar にわたる障害が発生する確率はきわめて小さく抑えられます。これは（大規模 SMP タイプの）他社のサーバでは実現されておらず、Integrity サーバの大きなアドバンテージになっています」（フェルゼンタール氏）。

以上、今回は sx2000 チップセットによって生まれ変わった新しい Integrity サーバの魅力を紹介した。長く使うための構造、継続運用のための仕組み、パーティショニング間のトラフィックを念頭としたシステム設計、これらのメリットはそのまま HP-UX 11i に受け継がれる。ミッションクリティカルーサーバとは、ノウハウや知恵の集約であることをご理解いただけたはずだ。

大きく変貌する HP-UX のシステム管理

HP-UX 管理は「SAM」から「SMH」へ

2006 年 6 月　テクニカルライター 吉川和巳

SAM (System Administration Manager)は、HP-UX の運用で必要とされる大半の管理機能を網羅しており、HP-UX 管理者にとってなくてはならないツールのひとつだ。実際、HPE の調査によると、HP-UX 利用者の 8 割が SAM を用いているという結果が得られている。いわば HP-UX の“定番ツール”である SAM だが、実は新世代のシステム管理ツールにその座を譲ろうとしている。その新ツールとは、2005 年 5 月より HP-UX に搭載され始めた、SMH（System Management Homepage）である。SMH は、ひとことでいえば「Web 版の SAM」だ。ここでは、SMH を中核とする HP-UX システム管理の今後のロードマップについて概観する。

HP-UX 管理は SAM から SMH へ

SAM（System Administration Manager、以下 SAM）は、HP-UX の運用で必要とされる大半の管理機能を網羅しており、HP-UX 管理者にとってなくてはならないツールのひとつだ。実際、HPE の調査によると、HP-UX 利用者の 8 割が SAM を用いているという結果が得られている。

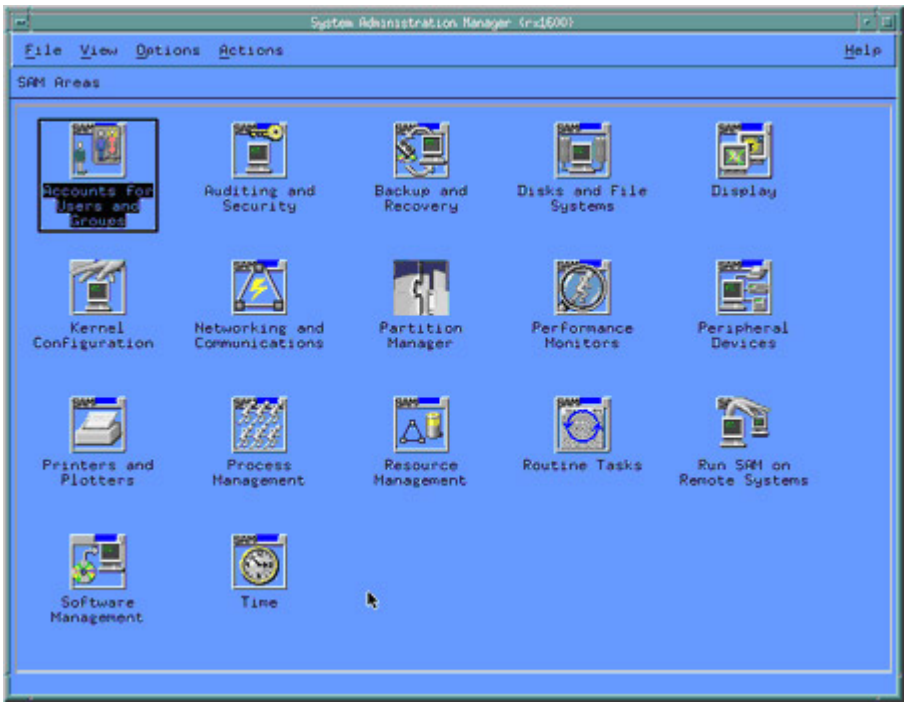


図 1：SAM の画面例

いわば HP-UX の"定番ツール"である SAM だが、実は新世代のシステム管理ツールにその座を譲ろうとしている。その新ツールとは、2005 年 5 月より HP-UX に搭載され始めた、SMH（System Management Homepage、以下 SMH）である。
SMH は、ひとことでいえば「Web 版の SAM」だ。SMH では、ハードウェア障害管理をはじめ、パフォーマンス情報の表示、システムの各種しきい値の管理、各種診断、そしてソフトウェアのバージョン管理などを SMH の Web インターフェースから実施できる。

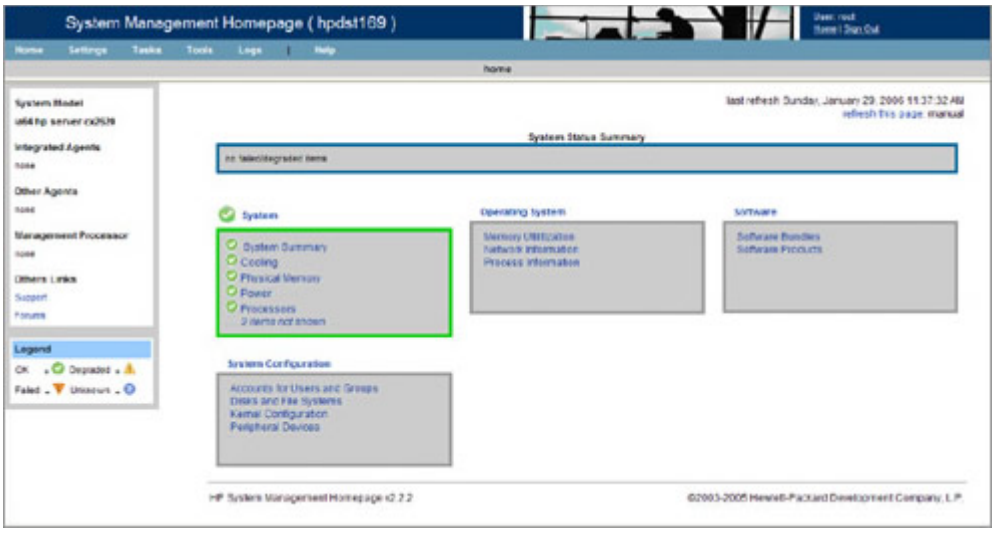


図 2：SMH の画面例

また HP-UX 専用のツールである SAM とは違い、SMH はマルチプラットフォーム対応のシステム管理ツールである。HP-UX をはじめ、Windows および Linux など、Integrity サーバと ProLiant サーバ上で動作する 3 種類の OS をサポートする。
さらに SMH では、SMH プラグインと呼ばれる拡張メカニズムをサポートしており、kcweb やパーティション・マネージャといった既存の Web ベースのツールも SMH に統合されている。

表：SMH に統合された Web ベースの管理ツール群

ツール名	内容
HP-UX Accounts for Users and Groups (ugweb)	ユーザおよびグループの管理
HP-UX Disks and File Systems (fsweb)	ディスク、ファイルシステム、LVM の管理
HP-UX System Fault Management (evweb)	イベント管理およびイベント表示
HP-UX Kernel Configuration (kcweb)	カーネル構成管理
HP-UX Peripheral Device (pdweb)	周辺デバイス管理
HP-UX Security Attributes Configuration (secweb)	セキュリティ管理
Partition Manager (parmgr)	nPartitions 管理
VM Manager	Integrity VM 管理

これらの他にも、プリンタ管理やネットワーク管理、監査機能、PAM 設定、リソース管理、ソフトウェア管理、時刻設定など、これまで SAM から起動して利用していた X-Windows ベースの管理ツールも、SMH から起動可能である。ちなみに、これら X ベースのツールは Web ベースへの移行が随時進められており、将来的にはその大半が Web ベースとなる予定だ。

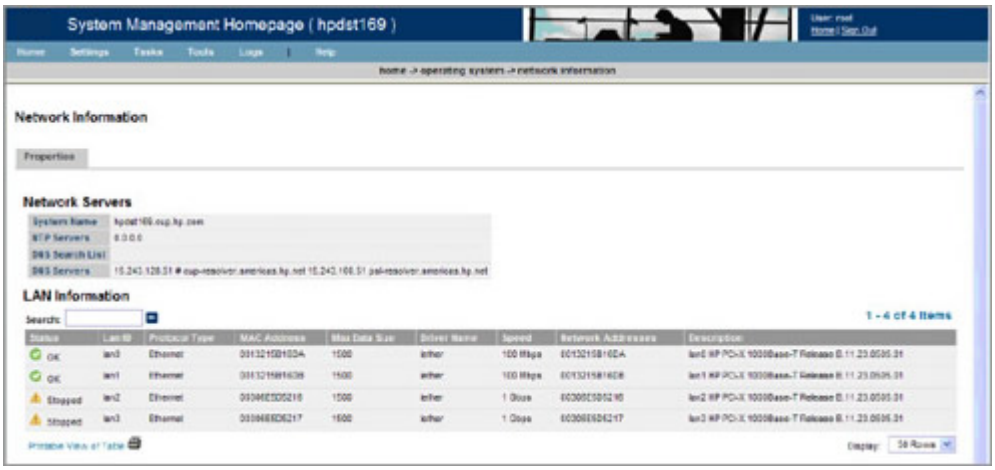


図 3：SMH によるネットワーク設定の表示例

このように、「X ベースもしくは TUI（Terminal User Interface）ベースで利用する SAM」から、「Web ベースで利用する SMH」へと、HP-UX のシステム管理は大きく変貌しようとしている。こうした Web 化の流れの背景には、Linux の「Webmin」に代表される Web ベースのシステム管理ツールが新しい世代のエンジニアに浸透しつつある現状がある。これから新たに HP-UX に触れる若手エンジニアにとって、PC の Web ブラウザから簡単に操作できる SMH は、より親しみやすいシステム管理ツールとなるはずだ。

また、もともと SMH は ProLiant サーバ向けの Windows／Linux 対応システム管理ツールとして提供されていた製品だ。その SMH が、Integrity サーバ上の HP-UX にも対応し、これまで SAM が担ってきた役目を引き継ぐ格好である。HP-UX・Windows・Linux のシステム管理ツールを統一化することで、OS ごとに別々のツール操作を習得する必要もなくなる。サーバ・コンソリデーションの普及とともに、ひとりの管理者がこれらの OS を扱うケースも今後は珍しくなくなるだろう。SMH への統合は、そうした管理者にとって大いに歓迎されるはずだ。
では、今後の HP-UX のシステム管理はすべて SMH に集約されるのだろうか？　SAM や SIM の位置づけは？　つづいては、こうした疑問に答える HP-UX システム管理のロードマップを紹介する。

HP-UX システム管理のロードマップ

HPE では、HP-UX システム管理の手段として、以下の 4 種類のツールを引き続き提供する予定だ。

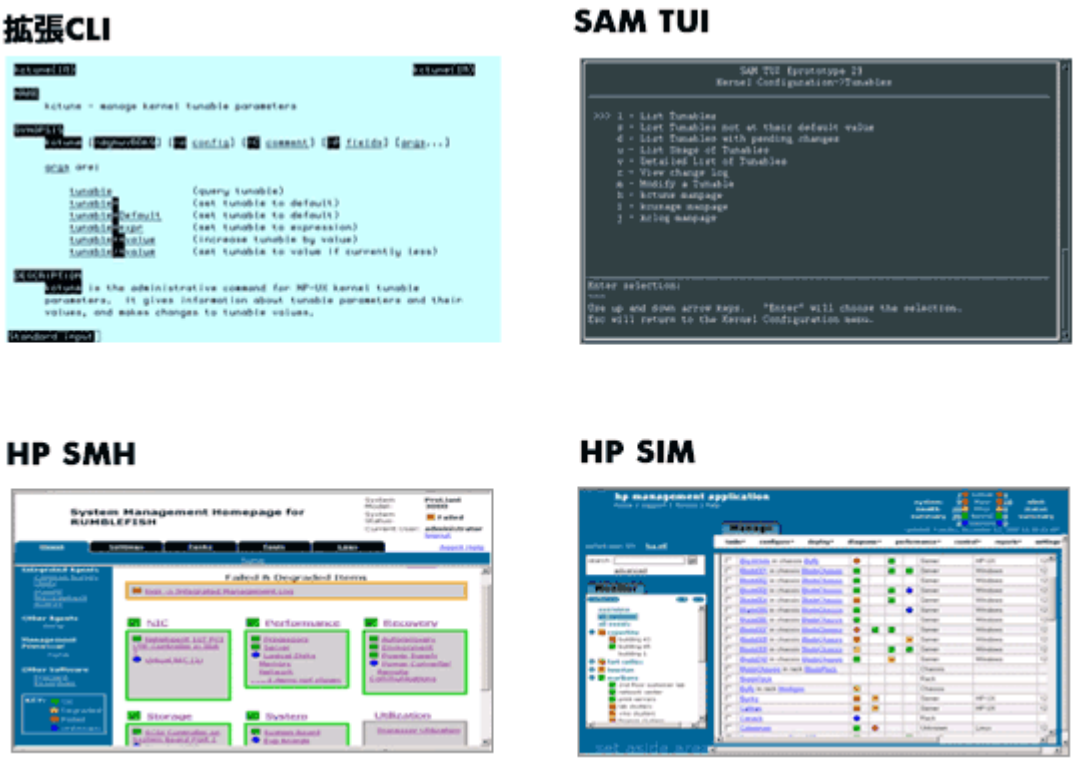


図 4：HP-UX の 4 つのシステム管理ツール

【拡張 CLI】

拡張 CLI（Enhanced Command Line Interface）とは、要するにシェルからコマンド入力で利用できる管理コマンド群のことである。当然のことながら、こうした CLI による管理機能も引き続き提供される。熟練した UNIX 管理者にとってはもっとも効率のよい管理手段である上、シェルスクリプト作成による管理作業の自動化が最大のメリットだ。また、ネットワーク接続が未設定の Integrity サーバであっても、シリアルケーブルで接続しさえすれば利用できる。

【SAM TUI】

SAM の TUI、すなわちメニュー形式のコンソール UI による管理機能も、これまで通り利用可能である。拡張 CLI と同様に、シリアルケーブルや ssh、telnet などで接続するだけで利用できる軽量さが魅力だ。一方で、メニュー形式の UI ではオプションや操作が一覧表示されるため、拡張 CLI のようにコマンドオプションを頭に入れておく必要性が少ない。コマンド入力と GUI ツールの中間的な存在だ。

【SIM】

SIM（Systems Insight Manager、以下 SIM）は、複数システム向けの Web ベース管理ツールである。SAM や SMH はあくまでシステム単体用の管理ツールであり、複数台のサーバを効率よく管理する機能は備えていない。一方、SIM はデータセンターなどにおいて多数のサーバを効率よく管理するために開発されたツールである。例えば、ネットワーク上の各サーバから自動的にサーバ・リソース情報を収集するインベントリ管理。または、何人もの管理者がデータセンターを共同運用するために必要な、ロールベースのセキュリティ管理とシングルサインオン機能などを提供する。

【SMH】

先に述べたとおり、SMH の位置づけは「Web 版の SAM」である。すなわち、システム単体の Web ベース管理機能を網羅したツールだ。Web ブラウザから利用できるわかりやすい UI を提供するほか、kcweb、pdweb といった既存の Web ベースのツール、および SIM とシームレスに統合されている。拡張 CLI のように個々のコマンドやオプションを覚える必要がなく、いわば“Web 世代”の HP-UX 管理者に最適なシステム管理手段といえるだろう。今後この SMH は、HP-UX のシステム管理ツールとしてメインストリームの役目を担い、積極的に機能追加されていく予定だ。

SMH のアドバンテージ

では最後に、SMH への移行によって得られるメリットをまとめておこう。

- **Web ベースである**—現時点で Web ベースへ移行していないツールを除いて、SMH ではシステム管理機能の大半が Web ブラウザからアクセスできる。つまり、HTTP および HTTPS での接続さえ可能であれば、リモート管理も容易に実施でき、SAM のように ssh／telnet 接続のためのファイアウォール設定などをあらかじめ済ませておく手間がかからない。また SSL（Secure Socket Layer）による通信の暗号化をサポートしており、SMH 上での操作・設定内容が漏洩するリスクもない。ちなみに SMH をインストールしたサーバでは、SMH 内蔵のアプリケーション・サーバ（Tomcat）が自動的に起動し、Web ブラウザからのリクエストを受け付ける状態となる。

- **Web ベース／X ベースのプラグインをサポート**—SMH では、Web ベースおよび X ベースの既存のシステム管理ツールをプラグインとしてサポートしている。これまでは SAM がいわゆる“ランチャー”（ツールを起動する窓口）の役目を担ってきたが、今後は SMH が統合ランチャーとして用いられる（なおリモートの PC などから X ベースのツールを起動するには、あらかじめ X サーバを PC にインストールしておく必要がある）。つまり、HP-UX のシステム管理では、どのような作業を始めるにも、まずは SMH のページを開くという動作が基本となるだろう。

- **マルチプラットフォーム対応**— ProLiant サーバや Integrity サーバ上で動作する Windows、Linux、HP-UX の各プラットフォームをサポート。それぞれのプラットフォームに個別のシステム管理ツールを習得する労力が少なくなる。

- **SIM とのシームレスな統合**—SAM にはない SMH の特徴のひとつは、SIM とシームレスに統合されている点だ。例えば、SIM 上に表示されたサーバー覧からいずれか 1 台を選択すると、SMH によるシステム単体の管理画面を開ける。また、いずれかのサーバにハードウェアのトラブルが検出された場合、まずは SIM の画面上で該当サーバに赤く印が付き、それをたどっていくことで SMH のプラグインである evweb の障害管理画面を開くことができる。つまり管理者から見れば、すべてがひとつの総合的な管理ツールに見えるのである。SIM や SMH、evweb といった各種ツールを個別に起動して、ツールを切り替えながらトラブル・シューティングする必要はない。

- **WBEM ベースの障害管理をサポート**—SMH では WBEM（Web Based Enterprise Manager）ベースの障害管理をサポートしている。この点については、つづく後編で説明する。

以上、ここでは次世代の HP-UX システム管理ツールである SMH の概要を紹介した。つづく後編では、SMH のもうひとつの特徴である WBEM ベースの障害管理機能について掘り下げる。

WBEM ベースの障害管理ツール SFM と evweb

2006 年 6 月　テクニカルライター 吉川和巳

WBEM（Web Based Enterprise Management）とは、標準化団体 DMTF によって標準化された「Web ベースのエンタープライズ管理」のための仕様だ。HP-UX では、この WBEM に基づいて多種多様な情報取得が可能であり、Integrity サーバーのハードウェア情報（プロセッサーやメモリ、ファン、電源）から、OS 情報（プロセス、ファイルシステム、LVM、カーネルパラメータ）、ネットワーク（IP、DNS、NIS）、仮想化関連の情報（nPars、vPars、Integrity VM）、さらには Serviceguard に至るまでを網羅する。ここでは、WBEM による新しい世代の障害管理メカニズムについて紹介する。

WBEM とは

前編で紹介した SMH（以下、SMH）のもうひとつの特徴、それは WBEM（以下、WBEM）ベースの障害管理ツール「evweb」を統合している点にある。その説明に入る前に、まずは「WBEM とは何か」を簡単に説明しておこう。

WBEM（Web Based Enterprise Management）とは、マイクロソフトやインテル、シスコシステムズ、旧コンパック・コンピュータなどによって提案され、標準化団体 DMTF（Desktop Management Task Force）によって標準化された規格である。その名の通り、「Web ベースのエンタープライズ管理」のための仕様であり、ネットワーク上に存在するサーバーや OS、ネットワーク機器などの統合管理に必要なプロトコルやデータ構造を規定する。要するに、従来の SNMP（Simple Network Management Protocol）や syslog に代わる、HTTP と XML をベースとした新世代のネットワーク管理プロトコルである。すでにマイクロソフト Windows や HP-UX などのおもな OS をはじめ、サーバー製品、ストレージ製品、ネットワーク機器、そして OpenView をはじめとするネットワーク管理ツールにおいて、業界標準のネットワーク管理プロトコルとしてサポートされている。

WBEM と CIM

この WBEM では、CIM（Common Information Model）と呼ばれるデータ構造（スキーマ）を用いて、管理対象のサーバーや OS から多種多様な情報を取得できる。例えば以下は、Integrity サーバーに対して WBEM 経由で OS 情報を要求したときに得られる CIM 情報（CIMXML）の例である。赤字の部分にホスト名や OS 名が記されている点に注目していただきたい。

```
<?xml version ="1.0" encoding="utf-8"?>
 <CIM CIMVERSION="2.0" DTDVERSION="2.0">
  <MESSAGE ID="51000" PROTOCOLVERSION="1.0"
   <SIMPLERSP>
   <IMETHODRESPONSE NAME="EnumerateInstances">
    <IRETURNVALUE>
     <VALUE.NAMEDINSTANCE>
      <INSTANCENAME CLASSNAME="PG_OperatingSystem">
```

```
＜中略＞
      <KEYBINDING NAME="CSName">
       <KEYVALUE VALUETYPE="string">
        mycomputer.hp.com
       </KEYVALUE>
      </KEYBINDING>
      <KEYBINDING NAME="Name">
      <KEYVALUE VALUETYPE="string">
       HP-UX
      </KEYVALUE>
      </KEYBINDING>
     </INSTANCENAME>
＜以下略＞
```

このように、HTTP および XML という分かりやすい Web ベースのプロトコルを通じてサーバーや OS、ネットワーク機器の管理が可能な点が、WBEM のアドバンテージである。

WBEM Services for HP-UX

HP-UX 11i v2 では、HP-UX のさまざまな管理情報を WBEM 経由で提供できる「WBEM Services for HP-UX（以下、WBEM Services）」および「WBEM Providers for HP-UX（以下、WBEM Providers）をサポートしている。以下の図は、この両者の関係を示した図である。

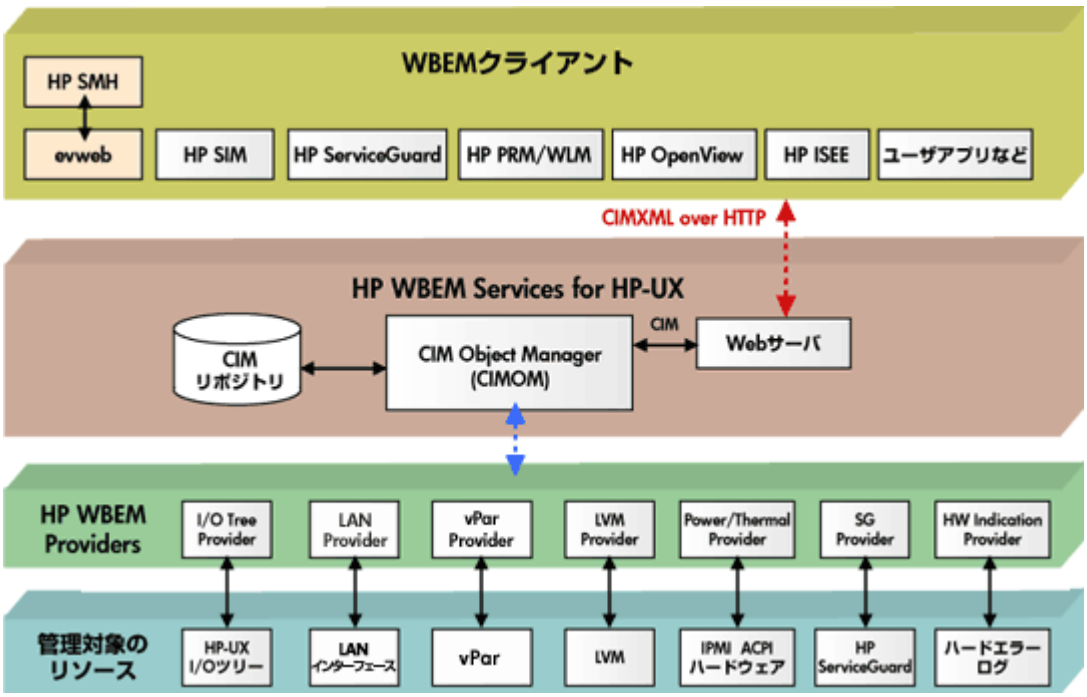


図 1：WBEM Services と WBEM Providers の位置づけ

WBEM Provider とは、管理対象となるリソースから情報を収集し、WBEM Services に伝えるソフトウェアである（SNMP における SNMP エージェントに相当する）。HP-UX では豊富な WBEM Providers に対応しており、Integrity サーバーのハードウェアまわりの情報（プロセッサーやメモリ、ファン、電源など）から、OS 情報（バージョン、プロセス、ソフトウェア、LVM

など）、ネットワーク（IP、DNS、NIS など）、仮想化関連の情報（nPars、vPars、Integrity VM など）、さらには Serviceguard に至るまでを網羅している。

以下の表は、HP-UX 11i v2 における WBEM Provider の対応状況をまとめたものだ。

表：HP-UX 11i v2 対応の WBEM Provider

WBEM Provider
システム情報（サーバーモデル、シリアル番号など）
nPars 情報
vPars 情報
Integrity VM
プロセッサー
メモリ
ファン
電源
Ethernet
ホストバスアダプタ（FibreChannel および SCSI）
IO ツリー拡張
OS 情報（バージョンなど）
SD-UX
プロセス
LVM
IP、DNS、NIS、NTP
Serviceguard

これらの WBEM Provider が取得した情報は、WBEM Services に含まれる CIMOM（CIM オブジェクトマネージャ）に集められる。そこで上記例のような CIMXML に変換され、Web サーバーを通じて WBEM クライアントに提供される流れだ。

WBEM クライアント

WBEM クライアントとしては、SMH に統合された evweb をはじめ、SIM、OpenView などを利用する。 さらに、HP-UX 上では WBEM の開発キットが提供されているので、ユーザー・アプリケーションに WBEM クライアント機能を組み込むことで、WBEM Provider から得た情報をアプリケーションの動作に反映させたり、エンタープライズ管理ツールを自作したりすることも可能だ。上述のとおり、すべての情報は XML として取得できるため、SNMP などに比べてこうした作り込みは容易である。
また、HPE が提供する遠隔モニタリングサービス ISEE（Instant Support Enterprise Edition）も WBEM クライアント機能に対応している。WBEM Provider から自動通知された障害情報に基づき、リアルタイムの遠隔サポートを提供可能だ。

つづく後半では、この WBEM のメカニズムによる HP-UX の障害管理について説明する。

障害管理は EMS から SFM へ

こうした WBEM Services および WBEM Providers の機能は、これまで EMS（Event Monitoring Services、以下 EMS）が担ってきた役割の一つ、障害情報のモニタリング機能と重なる部分が多い。実のところ、SAM から SMH への移行と同様に、今後 EMS はそのモニタリングの役割を段階的に、SFM(System Fault Management、以下 SFM)と呼ばれる新しいツールに譲る方向だ。

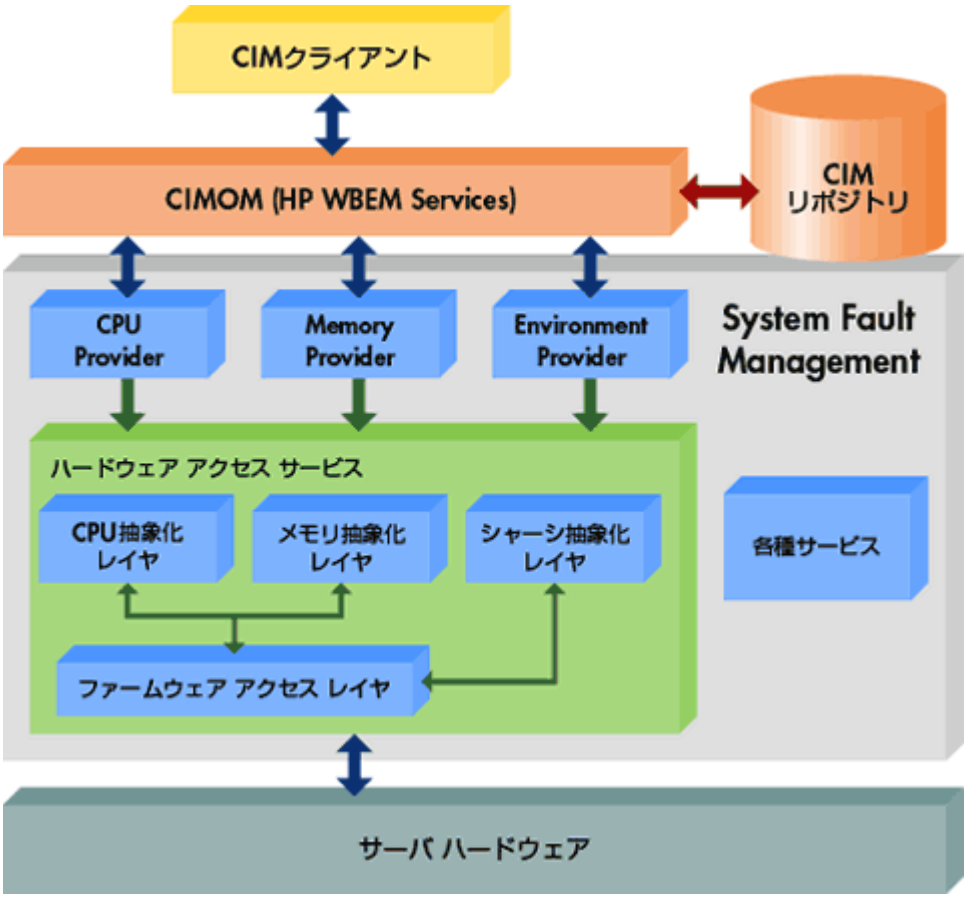


図 2：SFM のアーキテクチャ

図 2 に示すとおり、SFM は Integrity サーバーのプロセッサーやメモリ、シャーシ（ファン・電源・温度・電圧など）、そしてファームウェアといったハードウェア・コンポーネントから情報を収集する WBEM Provider から構成されている。さらに、EMS が発したイベントを WBEM Indication（EMS イベントに相当するメカニズム）に変換する機能も備える。

また SFM では、以下の各種ツールを提供する。

- **evweb**
  WBEM Indication を表示する SMH プラグインおよびコマンドラインツール
- **fmdcontrol**
  evweb 上に表示する WBEM Indication をフィルタリングするコマンドラインツール

evweb の機能

ここに示したように、SMH において障害管理を司るプラグイン evweb は、SFM のツールのひとつとして提供される位置づけだ。例えば SFM によって収集された情報は、evweb の「イベントアーカイブ」に保存された後、以下の画面例のように SMH 上に表示される。

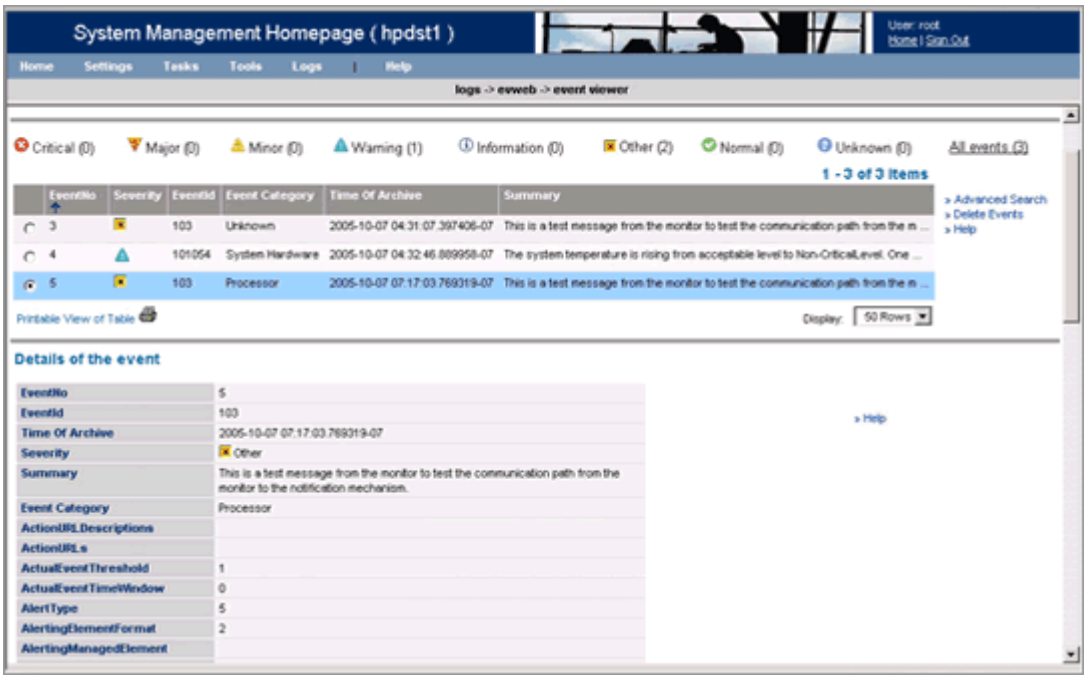


図 3：SMH の evweb 上に表示された障害情報

ちなみに evweb ではコマンドラインツールも提供しており、図 3 と同様の情報をコマンドベースで取得することも可能である。また、WBEM Indication に対するサブスクリプション（WBEM イベント通知登録）もサポートしており、イベントアーカイブに保存するだけでなく、例えばメールによる通知なども実現できる。

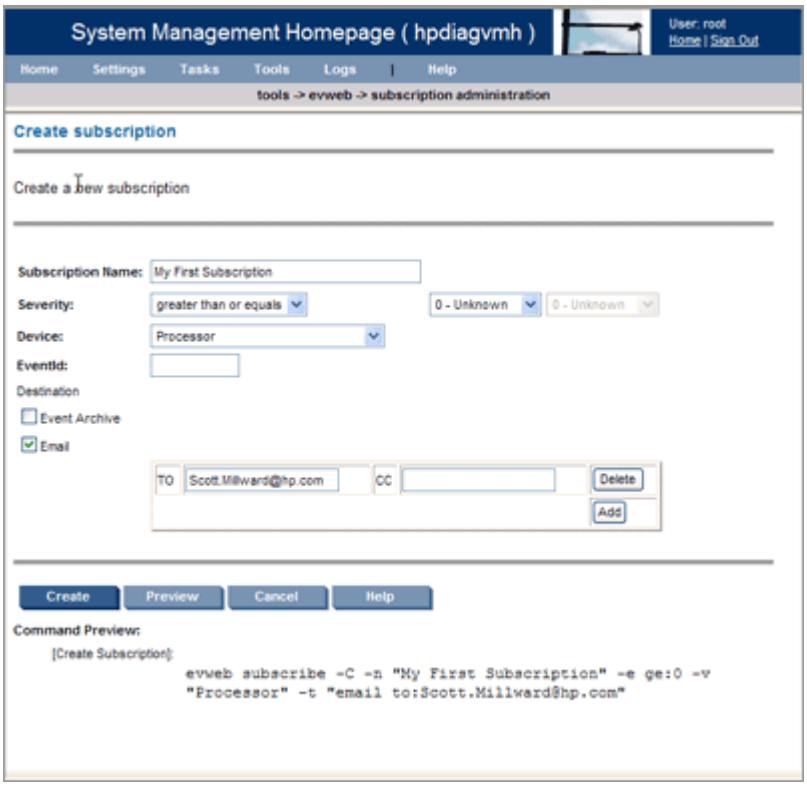


図 4：SMH の evweb 上でのサブスクリプション登録例

以上見てきたとおり、WBEM の登場によって HP-UX のシステム管理手法は大きく様変わりしつつある。EMS を中心としたプロプライエタリな障害管理から、WBEM および CIM による標準ベースの障害管理への変化である。Integrity サーバーや HP-UX に備わるきめ細かな情報収集能力を引き出し、アプリケーションレベルで活用することも格段に容易になるはずだ。

「アイデンティティ管理」のための Red Hat Directory Server

「アイデンティティ管理」のための Red Hat Directory Server・ 前編

2006 年 7 月　テクニカルライター 吉川和巳

例えばあなたは職場で、いくつの ID(アカウント)を利用しているだろうか？「アプリケーションやシステムごとに ID が別々」という状況のままでは、「日本版 SOX 法」といわれる内部統制やコーポレートガバナンスに関する制度化により、IT 部門に負担が一気に押し寄せることになる。そこで「アイデンティティ管理」の導入こそが、同法の施行後も IT 管理者が"涼しい顔"をしていられるかどうかの分かれ目となるのである。HP-UX では、アイデンティティ管理の基盤となるディレクトリ・サーバ「Red Hat

Directory Server for HP-UX」が無償バンドルされるほか、安心のベンダーサポートも利用できる。つまり HP-UX を扱う SIer にとって日本版 SOX 法は追い風となっているのだ。

HP-UX にとっては追い風の「日本版 SOX 法」

例えばあなたは職場で、いくつの ID(アカウント)を利用しているだろうか？まずは、朝出社して PC を起動し、Windows 環境にログオンするための ID。つづいて、社内のポータルサイトや各種 Web アプリケーションにアクセスするための複数の ID。さらにエンジニアならば、開発や運用に使用する Linux サーバや UNIX サーバにログインするための ID もいくつか所有しているだろう。

こうした「アプリケーションやシステムごとに ID が別々」という状況のままでは、「日本版 SOX 法」といわれる内部統制やコーポレートガバナンスに関する制度化により、IT 部門に負担が一気に押し寄せることになる。パスワードや ID 情報の管理体制に関するチェックや監査を、それぞれのアプリケーションやシステムについて個別に実施しなければならないからだ。同法は早ければ 2008 年 3 月期の導入が見込まれているため、今から準備することが必要となる。

そこでもし、あらゆるアプリケーションやシステムの認証機能を統合し、アクセス管理を一元化することができれば、日本版 SOX 法への対応の労力は格段に減らせる。つまり、こうした「アイデンティティ管理」の導入こそが、同法の施行後も IT 管理者が"涼しい顔"をしていられるかどうかの分かれ目となる。実際に SOX 法が施行済みの米国では、アイデンティティ管理を中核とした IT ガバナンスが現在の大きなトレンドとなっている。

こうした状況は、HP-UX を扱う SIer にとっては追い風である。その理由のひとつは、アイデンティティ管理の基盤となるディレクトリ・サーバ「Red Hat Directory Server 7.1 for HP-UX 11i (以下、RHDS)」が、2006 年 3 月から HP-UX に「無償」でバンドルされていることだ(ただし社内向けの利用に限る)。この RHDS は、OpenLDAP に代表されるオープンソースのディレクトリ・サーバとは異なり、HPE によるベンダーサポートが提供されており、大規模な基幹業務のにも安心して導入できる。また HPE では、ディレクトリ・サーバ以外にも、シングルサインオンをはじめ、ID プロビジョニング、ID 連携など、アクセス管理の構成要素となる各種ソリューションを用意している。つまり「アイデンティティ管理をどうするか？」という企業の悩みに対し、HP-UX であればベンダーサポート付きの実績あるソリューションをタイムリーに提案できるのである。

図 1：アイデンティティ管理の構成要素と HPE のソリューション

事実、日本ヒューレット・パッカードによると、HP-UX + RHDS によるディレクトリ・サービスに関する問い合わせが、今年に入って急増しているという。とりわけ製造業や通信業、金融機関など、国内屈指の大手企業や官公庁による導入が目を引く。例えばある国内最大規模の金融機関では、HP-UX + RHDS で構築されたアクセス管理により、Windows 環境と UNIX 環境、そして Web アプリケーションの認証を一元化するプロジェクトが進行中だ。こうした大手でのアイデンティティ管理導入が加速する背景には、企業規模が大きいほど内部統制の負担が重くなるという切迫した事情がある。このように RHDS は、HP-UX や Integrity サーバにアイデンティティ管理基盤としての今日的な付加価値を与えている。

そこで後半ではこのアイデンティティ管理の基盤をなす「ディレクトリ・サービス」とは、そもそもどのようなテクノロジーなのかをあらためて概観したい。

ディレクトリ・サービスの位置づけ

ディレクトリ(directory)とは、住所録や電話帳を意味する。つまりディレクトリ・サービスとは、「ネットワーク上の電話帳サービス」と言えば分かりやすいだろう。具体的には、以下のような情報の保管や検索に広く用いられている。

- ID 情報(ユーザ名とパスワード、アクセス権など)
- 社員名簿(名前、メールアドレス、電話番号、組織名など)
- PKI(公開鍵基盤)におけるデジタル証明書
- ネットワーク機器やアプリケーションの設定情報
- セキュリティ・パッチ情報

ディレクトリ・サービスは、ネットワーク上の OS やアプリケーション、ネットワーク機器などが、さまざまな情報を保管し共有するリポジトリとして利用できるサービスだ。例えばディレクトリ・サービスの登場当初は、メールソフトのアドレス帳を企業内で共有する用途が想定されていた。その後さまざまに応用が広がり、例えば PKI 構築時のデジタル証明書の保管場所として用いられるほか、大規模ネットワークでのルータやスイッチの設定情報およびセキュリティ・パッチなどの一括配布、そして J2EE 開発ではデータベースやメールサーバへ接続するための電話帳サービスとして利用されている。要するにディレクトリ・サービスにどのような情報を保存するかは、ネットワーク管理者やアプリケーション開発者の自由なのである。

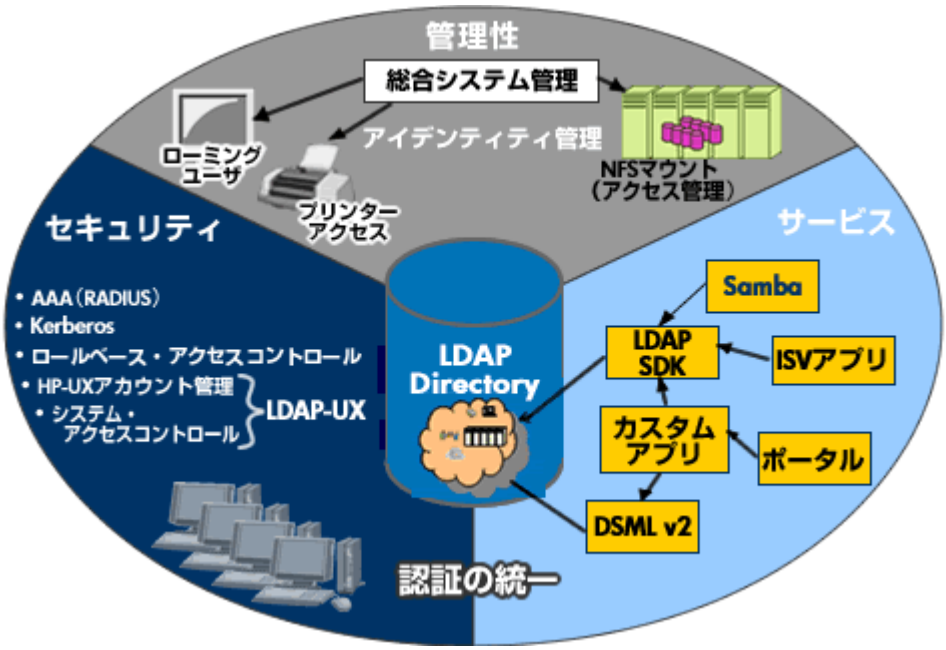


図 2：ディレクトリ・サービスの用途

とはいえ、ディレクトリ・サービスの使われ方としてもっともポピュラーなのは、やはり「アイデンティティ管理」である。例えば企業の Windows 環境において、社員が PC やファイルサーバを利用するためのユーザ名やパスワード、アクセス権などを全 PC に個別登録していたのではきりがない。そこで一般的には、Windows 2003 Server が内蔵するディレクトリ・サーバ「Active Directory」にユーザ名とパスワードを保存しておき、それを各 PC から共有することで個別登録の手間を省く。また ISP(Internet Service Provider)では、数 10 万人、数 100 万人という加入者のユーザ名とパスワードをディレクトリ・サーバ

で管理し、ダイヤルアップ接続時や ADSL モデム接続時、もしくはメールサーバ接続時に同サーバから認証情報を取得するシステム構成が広く用いられている。

ディレクトリ・サーバと LDAP

こうしたディレクトリ・サービスの構築と運用の中核を担うのが、ディレクトリ・サーバと呼ばれるソフトウェアだ。代表的なものとしては、以下の各製品がある。

- マイクロソフト Active Directory
- レッドハット Red Hat Directory Server(旧 Netscape Directory Server)
- OpenLDAP(オープンソース)

これらのディレクトリ・サーバは、いずれもディレクトリ・サービスの標準プロトコル「LDAP(Lightweight Directory Access Protocol)」をサポートしており、「LDAP サーバ」とも呼ばれる。よってディレクトリ・サービスを利用するクライアント、すなわち OS やアプリケーション、ネットワーク機器などは、この LDAP プロトコルを通じてディレクトリ・サーバにアクセスし、図 3 に示すようなツリー構造に基づいて情報の検索・更新・認証を実施する。

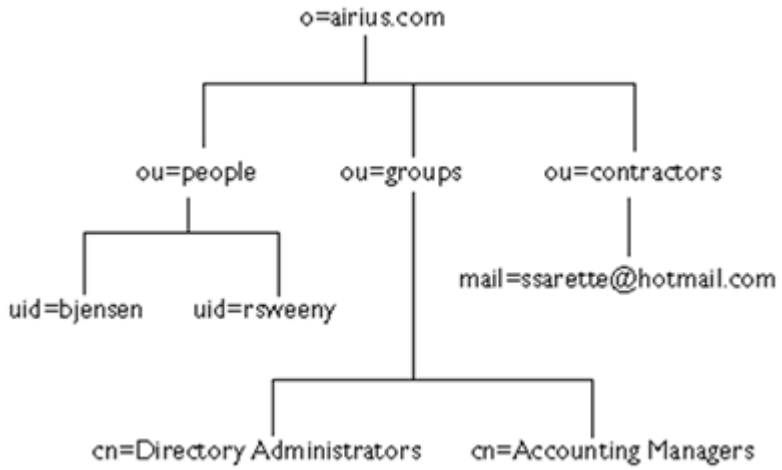


図 3：LDAP プロトコルのツリー構造

ちなみに、LDAP サーバの役割は「LDAP プロトコルを通じて情報の検索・更新・認証が行えるサーバ」であり、単純なユーザ認証以上のセキュリティ機能は原則として提供しない。よって、PKI やシングルサインオンといった高度なセキュリティやアクセス管理は、LDAP クライアント側(OS やアプリケーション、ネットワーク機器など)で実装する。例えばアイデンティティ管理の主要な構成要素であるシングルサインオンを実現するには、IceWall SSO や Kerberos といったシングルサインオンのソリューションを組み合わせる必要がある。実際、ある HP-UX + RHDS の導入事例では、RHDS のフロントエンドとして IceWall SSO を用いることで、Web アプリケーションのシングルサインオンを実現している。

ディレクトリ・サーバとデータベース

ここで、「情報の検索・更新・認証の手段ならば、データベースとそれほど違わないのではないか」という疑問も生じるだろう。実のところ、ディレクトリ・サーバは「LDAP プロトコルでアクセスできるデータベース」ととらえることもできる。実際に、ディレクトリ・サーバの多くは軽量なデータベースを内蔵(もしくは外部データベースに接続)することでその機能を実装しており、

例えば Active Directory は Active Directory データベース、そして RHDS は Berkley DB を内蔵する。その逆に、Oracle データベースを LDAP サーバとして使用するためのオプション製品(Oracle Internet Directory)も存在する。

とはいえ、ディレクトリ・サーバとデータベースは、それぞれの用途や歴史的経緯によって機能や特性に大きな違いがある。例えばディレクトリ・サーバは検索機能に的を絞って設計されており、頻繁な更新処理やトランザクション処理は想定していないため、その分ソフトウェア全体を軽量に設計することができる。Oracle データベースなどと比べれば、インストールに必要な手間やリソースも格段に少なくてすむ。また歴史的経緯により、シングルサインオンやセキュリティ管理ソフトウェア、ネットワーク機器の多くが LDAP を標準サポートしているという点も重要である。

ディレクトリ・サーバのもうひとつの特徴として、LDAP という標準プロトコルの存在がある。データベースには(SQL 言語や API の標準はあるものの)標準プロトコルが存在しないため、データベース接続にはベンダー製のデータベース・ドライバをクライアント側に組み込む必要がある。一方、ディレクトリ・サーバへの接続には特別なドライバは必要ない。LDAP プロトコルさえ実装すれば任意の LDAP サーバに接続でき(ただし実際には製品間の非互換性も存在する)、ベンダー依存度の低いインテグレーションが可能だ。

さらにディレクトリ・サーバでは、汎用のスキーマ(データ構造)が標準化されており、例えば UNIX のアカウント情報を LDAP サーバ上に保存する際の標準スキーマなどが規定されている。そのため、異なるアプリケーションやプラットフォーム間での情報共有が容易だ。これに対しデータベースの場合、スキーマは標準化されておらず、アプリケーション間連携には必ず作り込みが必要となる。

引き続き後編では、HP-UX + RHDS によるディレクトリ・サーバ構築のメリットや、そのポイント、ライセンス形態などを詳解する。

「アイデンティティ管理」のための Red Hat Directory Server・ 後編

2006 年 7 月　テクニカルライター 吉川和巳

数あるディレクトリ・サーバのなかでも、Red Hat Directory Server はいわば「元祖」といえる LDAP サーバである。シングルシステムで数 100 万ユーザをサポートできるスケーラビリティ、そして毎秒数 1000 件の検索を実行できる優れたパフォーマンスがその特徴である。また OpenLDAP などのオープンソース・ソフトウェアとは異なり、HP-UX 版の RHDS は、HPE が「オペレーティング環境(OE)の一部分」として提供し、OS 本体と同レベルのベンダーサポートを提供する商用製品である。企業の根幹を支えるディレクトリ・サービスにはうってつけの選択肢と言えるだろう。

“元祖”LDAP サーバ、RHDS

数あるディレクトリ・サーバのなかでも、Red Hat Directory Server(以下、RHDS)はいわば“元祖”といえる LDAP サーバである。

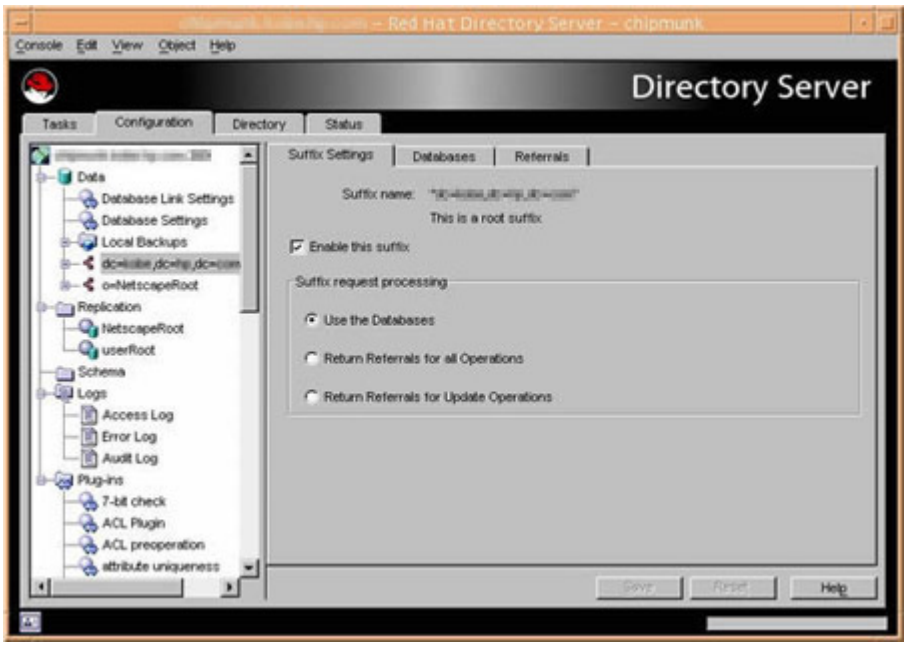


図 1：Red Hat Directory Server の管理画面

そもそも LDAP プロトコルは、OSI ネットワーク向けの複雑なディレクトリ・サービス・プロトコルであった X.500 を TCP/IP ネットワーク向けに改良し、シンプルで軽量なプロトコルとして生み出されたという経緯がある。この LDAP プロトコルの標準化を主導した旧ネットスケープ社のエンジニアが、1996 年に最初の LDAP 対応ディレクトリ・サーバとなる Netscape Directory Server(以下、NDS)を開発。以降 NDS は、LDAP サーバの代名詞として非常に広く普及した。ちなみに HPE では、HP-UX 版の NDS(Netscape Directory Server for HP-UX 11i)を 2003 年から提供している。

その後、AOL によるネットスケープの買収、そしてレッドハットによる NDS の買収を経て、現在 NDS はレッドハットの製品 Red Hat Directory Server 7.1(オープンソース版は Fedora Directory Server)として提供されている。RHDS は、業界標準の LDAP サーバであるという点に加えて、以下のような特徴を備える。

- シングルシステムで数 100 万ユーザをサポートできるスケーラビリティ
- 毎秒数 1000 件の検索を実行できる優れたパフォーマンス
- マルチマスタ・レプリケーション機能による高可用性
- サーバの設定管理用コンソールおよび管理ツールを提供

また今回 HPE では、2006 年 3 月より、HP-UX 版の RHDS の販売を開始した。注目すべきポイントは、ユーザの社内向け利用に限り、無償で利用できる点である。

- 製品名：Red Hat Directory Server for HP-UX 11i
- 対応 OS：HP-UX 11i v1 および v2
- 対応サーバ： Integrity サーバおよび HP 9000 サーバ

このライセンス体系では、例えばユーザ企業の社内システムへのアクセス管理構築に用いる場合、社員数がどれだけ多くてもライセンス費用は発生しない。一方、ISP などコンシューマー向けサービスに RHDS を用いる場合はプロセッサ・コア・ライセンス費用が発生するものの、ユーザ数無制限となっているので、以前の NDS に比べればユーザあたりのコストは格段に低くなっている。

Active Directory と RHDS

では、他のディレクトリ・サーバと比較して RHDS にはどのようなアドバンテージがあるのだろうか。

ディレクトリ・サーバの“東の横綱”が RHDS だとすれば、“西の横綱”は Active Directory だ。前編で説明したとおり、この両者はいずれも LDAP プロトコルを実装した LDAP サーバである。とはいえ実際には、Active Directory は Windows 環境に閉じたプロプライエタリなディレクトリ・サーバとしての性格が強い。例えば Linux や UNIX から Active Directory に LDAP 経由で接続し、Windows 環境とのユーザ認証を統合する場合、マイクロソフト独自のスキーマや非互換性の影響で一筋縄ではいかないケースが多いとも言われる。

やはり LDAP プロトコルを中軸とするマルチプラットフォーム環境のインテグレーションでは、RHDS のような LDAP との親和性が高いディレクトリ・サーバを選択した方が無難であろう。実際、ディレクトリ・サーバとして RHDS を導入し、Active Directory を一切使用せずに Windows、Linux/UNIX、Web アプリケーションの認証の統合化を実現した事例も多い。この場合、Windows 環境のユーザ認証を司るドメイン・コントローラとしては、HP-UX 上で動作する Samba サーバ(CIFS Server)を用いる。この Samba サーバと RHDS を LDAP プロトコルで結ぶことで、Windows クライアントのログオン認証時に RHDS 上のユーザ情報を参照する仕組みである。

また Web アプリケーションのユーザ認証を RHDS に統合するには、Apache Web サーバに含まれる LDAP モジュール(mod_auth_ldap)を利用する方法や、アプリケーション・サーバの LDAP クライアント機能を使う方法、そして IceWall SSO などのリバース・プロキシ型シングルサインオン・ソフトウェアをフロント・エンドとして配置する方法がある。

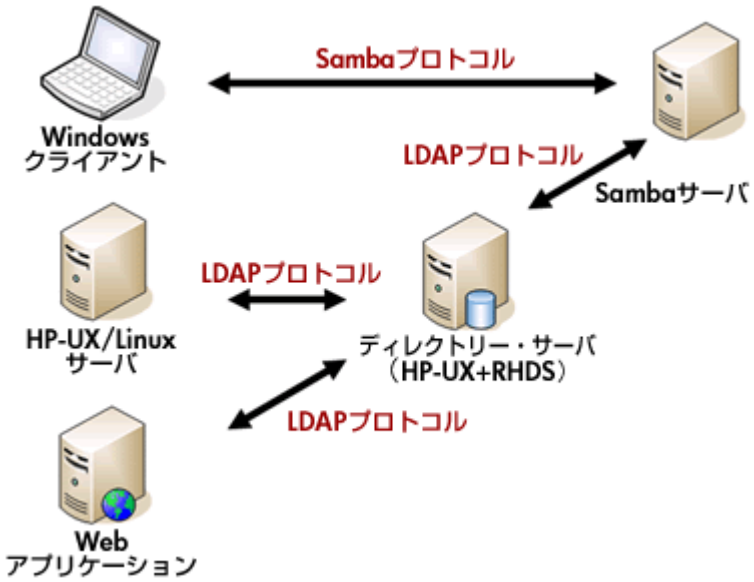


図 2：RHDS による Windows・HP-UX・Web アプリケーションの認証統合

つづいては、OpenLDAP と RHDS それぞれのメリット・デメリットを比較する。

OpenLDAP と RHDS、どちらを選ぶか

Active Directory、RHDS と並ぶポピュラーな製品が、オープンソースのディレクトリ・サーバである「OpenLDAP」である。Linux ベースのディレクトリ・サーバ構築では、必ずといってよいほど OpenLDAP が選択される。OpenLDAP のメリットは、

当然のことながら、どのような用途、どのような規模でも無償で利用できる点だ。またインターネット上から得られる情報量も豊富である。

OpenLDAP のネックは、オープンソース・ソフトウェアであるため正式なベンダーサポートを受けられない点だ。ディレクトリ・サーバは、企業全体の ID(アカウント)管理や認証統合に用いられることが多く、そうしたアイデンティティ管理を実施する企業はたいてい大規模である。そのため、ディレクトリ・サーバが万が一ストップすると、企業全体の業務が停止してしまう。例えば、会社の始業時に全社員の PC がログオンできなくなった状況を想像していただきたい。このように、部門レベルのファイル・サーバや個別のアプリケーションとは全く異なるレベルの可用性とサポータビリティが要求されるのが、ディレクトリ・サーバだ。OpenLDAP を導入する場合は、そうした可用性やスケーラビリティ、サポータビリティを提供できるか熟慮しなくてはならない。

これに対し HP-UX 版の RHDS は、HPE が「オペレーティング環境(OE)の一部分」として提供し、OS 本体と同レベルのベンダーサポートを提供する商用製品である。HPE では RHDS の開発でレッドハットと密接に協力しているほか、RHDS のソースコードにアクセスする権利を有している。よって、例えば RHDS にバグが発見された場合も、ソースコード情報をもとに HPE が独自にパッチを作成することができる。さらにスケーラビリティに関しても、エントリ規模が数 10 万件を越える状況では OpenLDAP より高速に動作するという報告もある。

もうひとつは、Linux と HP-UX の可用性やセキュリティの違いである。そもそも HP-UX はミッションクリティカル用途に強いプラットフォームである。またシステム単体のセキュリティを考えても、Linux では root 権限を奪われるとディレクトリ・サーバの内容がたやすく漏えいしてしまう。HP-UX では Security Containment や RBAC の組み合わせにより、OS 全体をいくつかのコンパートメントに分割し、さらに管理者のアクセス権限も細分化できる。これにより、ディレクトリ・サーバという企業の基幹情報の集積場所について、厳格なセキュリティ・ポリシーを適用できる。これらの仕組みを活用することで、日本版 SOX 法で求められる内部統制のポリシー履行や監査も実施しやすくなるはずだ。

HP-UX の LDAP ソリューション

また HP-UX では、RHDS 以外にも以下のような LDAP ソリューションを提供している。

【LDAP-UX Client Services】

「LDAP-UX Client Services」は、HP-UX のログイン時のユーザ認証に LDAP サーバを参照するための機能である。具体的には、UNIX の標準的な認証アーキテクチャである PAM(Pluggable Authentication Modules)および NSS(Name Service Switch)によるユーザ認証を LDAP 経由で実施するためのモジュールが提供されている。またこのモジュールを利用すれば、ログイン認証以外にも、Apache やメール、Samba サーバなどのユーザ認証の LDAP 統合も実装できる。

【NIS/LDAP Gateway】

「NIS/LDAP Gateway」は、いわば「LDAP ベースの NIS サーバ」である。具体的には、NIS ドメインごとに設置された NIS/LDAP Gateway が NIS スレーブサーバとして動作する。同サーバは、NIS クライアントからの認証リクエストを受け付け、それを LDAP 照会に変換する仕組みだ。これにより、NIS マスターサーバが不要になるだけでなく、大規模な NIS ドメインにおけるスケーラビリティやレスポンスの問題も解消される。

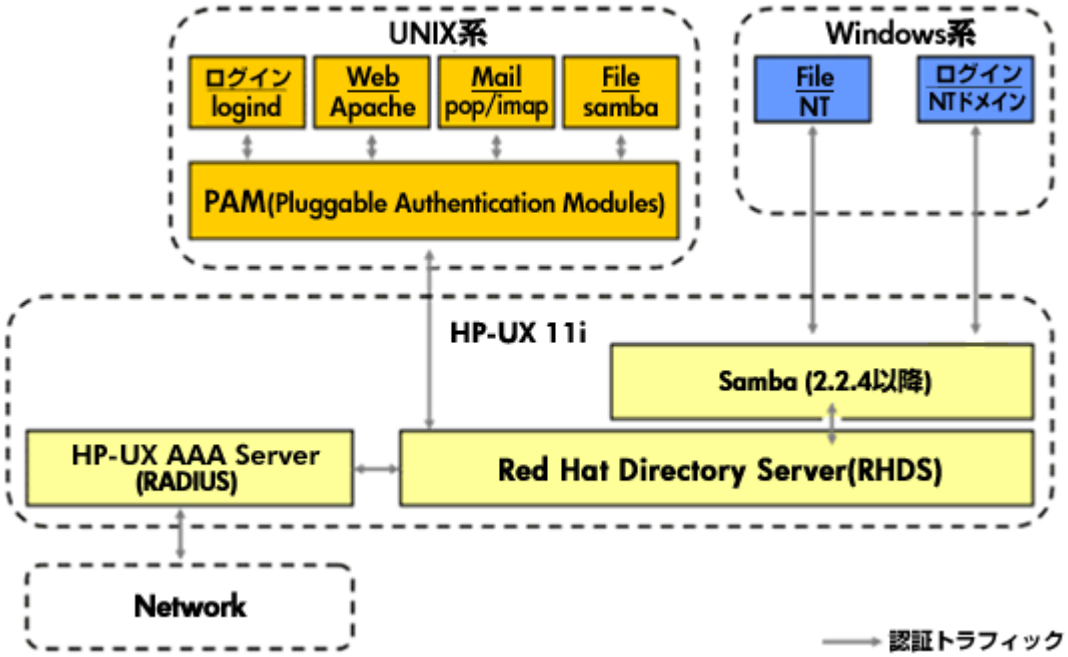


図 3：RHDS と LDAP-UX による認証統合の例

以上、今回は HP-UX 版 RHDS にスポットを当て、同ディレクトリ・サーバを中心とするアイデンティティ管理の構築手法を紹介した。

ここで説明したとおり、HP-UX + RHDS のアドバンテージは、社内向け利用では無償でありながら、基幹レベルのシステム構築に耐えるベンダーサポートを提供する点だ。よってアイデンティティ管理導入のひとつのアプローチとして、RHDS のライセンス体系を活用した「スモールスタート」が可能となる。つまり、まずは Windows 認証や UNIX 認証といった個別の機能を試験的に評価し、ノウハウを蓄積しながら全社規模へと展開していく。日本版 SOX 法の施行を前に、大がかりなシステム構築をいきなり検討するのではなく、まずは手元で"アイデンティティ管理の手応え"をつかんでみてはいかがだろうか。

DB クラスタ構築でハマらないための「HA Simple Cluster リファレンス・アーキテクチャ　v.1.0」

DB クラスタ構築でハマらないための「HA Simple Cluster リファレンス・アーキテクチャ　v.1.0」・前編

2006 年 8 月　テクニカルライター 吉川和巳

あなたは、データベース・クラスタ(以下、DB クラスタ)の構築に手を焼いた経験はないだろうか？実は、DB クラスタをやみくもに導入することで、思わぬ TCO の増加を招いてしまったケースも増えている。そこで HPE では、多くの IT エンジニアが持つこの悩みを解消すべく、「HA Simple Cluster リファレンス・アーキテクチャ　v.1.0(以下 Simple Cluster)」の提供を開始した。
Simple Cluster は、負荷分散を行わず、できるだけシンプルな構成で高速なフェイルオーバーを実現することを目指している。技術的に見れば時代に逆行していると言っても良い。だが、UNIX システムでのフェイルオーバー構成が全盛だった 2000 年前後に比べ、現在ではハードウェアの低価格化によりスタンバイ機に要するコストは劇的に安価になっている。その結果、一見無駄なスタンバイ機を用意している Simple Cluster だが、構築費用や運用費用を含んだ TCO 削減への貢献を可能にしている。

DB クラスタの必要性

ミッションクリティカル環境において DB クラスタが必要とされる理由、それは以下の 2 点に集約できる。

- 可用性の確保(HA クラスタ)
- 負荷の分散化(負荷分散クラスタ)

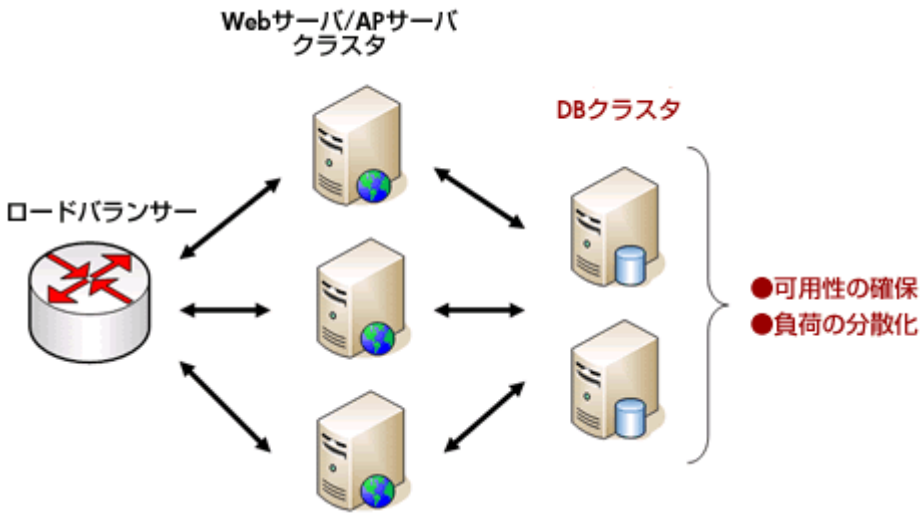


図 1：Web サイトの一般的な構成

DB クラスタの構築によって得られる、もっとも重要なメリットは「可用性の確保」である。データベースという名が示す通り、DB サーバにはサービスや業務のすべての情報が集まる。Web サーバやアプリケーション・サーバ(以下、AP サーバ)とは異なるレベルの高い可用性が求められることが多く、ミッションクリティカル環境では DB サーバのクラスタ化が常套手段となっている。例えば、2 台以上の DB サーバを用意し、いずれか 1 台がダウンした場合でもほかの DB サーバが肩代わり(フェイルオーバー)する「高可用性クラスタ(以下、HA クラスタ)」の構築である。
DB クラスタのもうひとつのメリットは、「負荷の分散化」である。近年では、DB クラスタの高機能化により、スケールアウトにより性能を確保する「負荷分散クラスタ」が実用段階に入っている。これにより、DB クラスタ内におけるハードウェアの使用効率が格段に上がっていることも事実だ。

DB クラスタの課題

IT システムの構築を「設計」、「構築」、「試験」、「運用」の 4 段階に分けたとき、DB クラスタで、次のような課題を耳にすることがある。

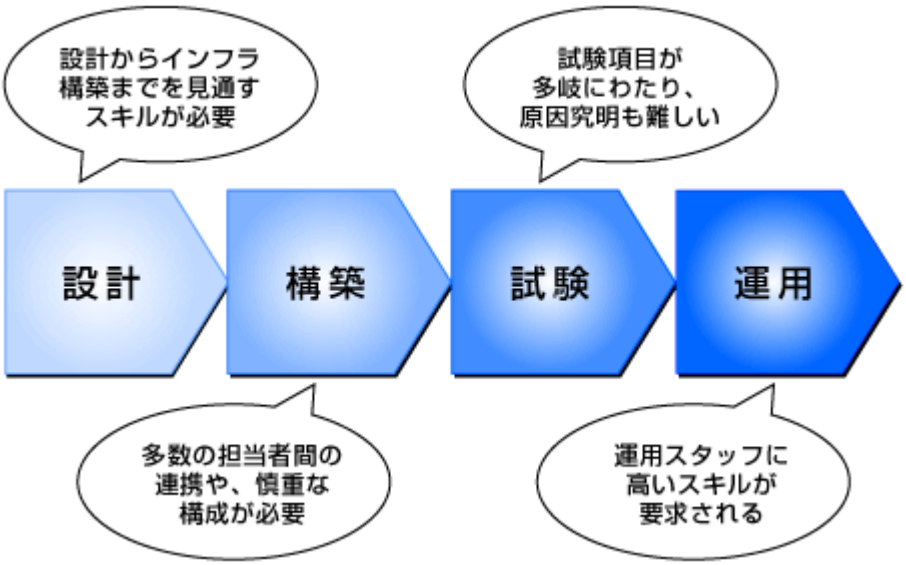


図 2：高機能な DB クラスタ構築のための課題

まず設計段階では、業務内容に基づく要件定義からアプリケーション設計、そして DB サーバなどのインフラ構築までのすべてを見通せる、有能な IT アーキテクトの確保が必須となることがある。インフラや DB クラスタなど様々ソフトウェアの特性やふるまいを理解した上で、アプリケーションを適切に設計する必要があるからだ。
次に構築段階である。DB ベンダーやミドルウェア・ベンダー、OS ベンダーの提供するコンポーネントを、パズルのピースのように慎重に組み上げていく。そのため、多数のベンダーとの協議や共同作業、そして担当者間のコーディネートが不可欠となる。実際、IT システムの構築担当者からは、「ミーティングのたびに多数の担当者を呼ばなくてはならない」、「単に DB を作成したいだけなのに、いくつもの選択肢を考慮する必要がある」、「DB クラスタのためにアプリ側の設計変更が生じることもある」といった声が聞かれる。
また試験段階では、DB クラスタを構成する個々のコンポーネントについて試験項目や試験順序を検討し、各担当者間で調整しながら作業を進めていく。この段階についても、「予想外の試験結果が出たとき、原因究明に手間取る」、「実運用環境とまったく同じ DB クラスタ構成を試験環境で用意することが困難」、「試験項目が多岐にわたり、カットオーバーまでに十分に試験する時間がない」などの意見がある。
さらに運用段階では、DB クラスタの複雑なふるまいに対処するための大量のドキュメントや徹底したトレーニングが必要だ。また運用スタッフにも高いスキルが要求され、運用コストの上昇を招く。
これらすべてが TCO を増加させる要因であり、しっかり考慮しないと思わぬ TCO の増加を招くことになってしまう。

「身の丈」に合ったシンプルな DB クラスタ

もちろん、予算や人員がふんだんに投入される大規模プロジェクトでは、こうした課題を乗り越え、望みどおりの DB クラスタを構築可能であろう。これによりプロジェクト規模にふさわしい高い可用性の確保と負荷の分散化を実現できるはずだ。
しかし、要件が高可用性の実現のみであれば、できるだけ構成製品を少なくし、それによりコストやリスクをできるだけ減らすという方法も考えられる。この多くのユーザの声をもとに登場したのが、「HA Simple Cluster リファレンス・アーキテクチャ v.1.0」である。負荷分散を行わず、できるだけシンプルな構成としながら、今まで以上の高速なフェイルオーバーを実現する DB クラスタの参考構成である。

Simple Cluster は、負荷分散を行わず、できるだけシンプルな構成で高速なフェイルオーバーを実現することを目指している。技術的に見れば時代に逆行していると言っても良い。だが、UNIX システムでのフェイルオーバー構成が全盛だった 2000 年前後

に比べ、現在ではハードウェアの低価格化によりスタンバイ機に要するコストは劇的に安価になっている。その結果、一見無駄なスタンバイ機を用意している Simple Cluster だが、構築費用や運用費用を含んだ TCO 削減への貢献を可能にしている。

そこで後半では、この Simple Cluster についてさらに掘り下げていきたい。

Simple Cluster の概要

まずは、Simple Cluster の概要を説明しよう。このリファレンス・アーキテクチャは、以下の要素から構成されている。

- Integrity サーバ(2 ノード)＋HP-UX 11i v2 (11.23)
- Serviceguard (11.17)
  ・Serviceguard Extension for Faster Failover (1.00)
  ・Quorum Server(A.02.00)
- データベース
- 「HA Simple Cluster 構築手順」ドキュメント

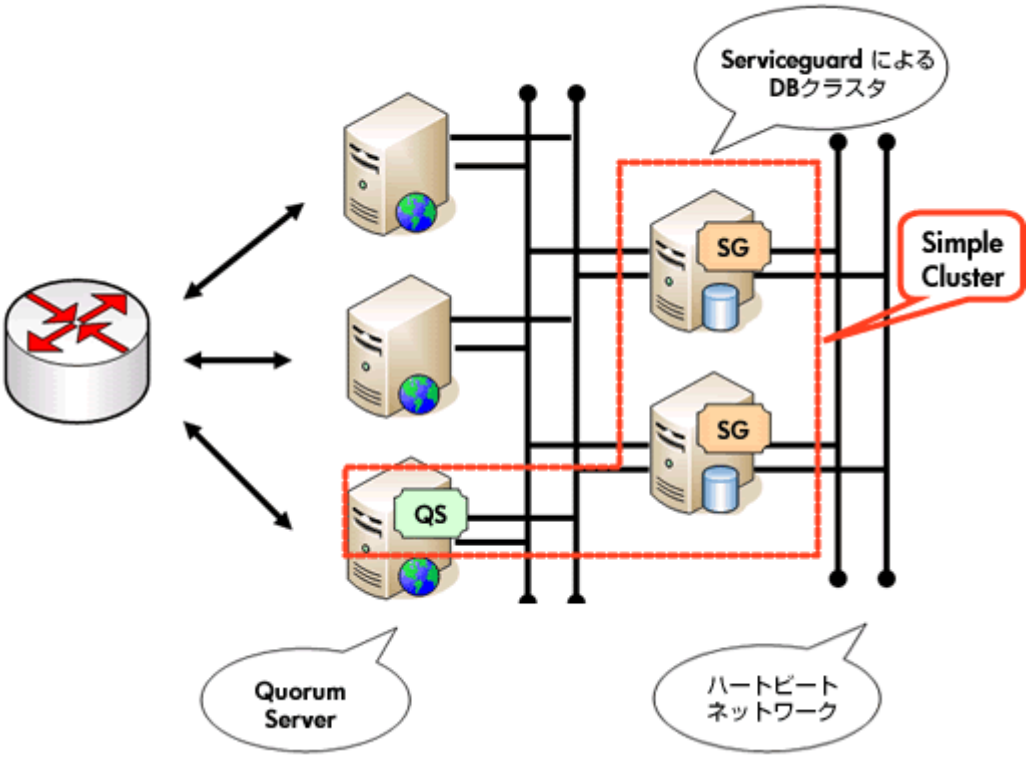


図 3：Simple Cluster の構成例

Simple Cluster は、HP-UX 11i v2 (11.23)を搭載した Integrity サーバを動作環境とする。クラスタウェアとしては Serviceguard (11.17)を使用し、同時に高速フェイルオーバー・オプションである Serviceguard Extension for Faster Failover (1.00)(以下、SGeFF)を用いる。
データベースには、Oracle Database 10g R2 を用いている。また Simple Cluster における重要なポイントは、ドキュメント「HA Simple Cluster 構築手順」を提供している点だ。同ドキュメントは、HPE のテクニカル・コンサルタントが手がけた多数の導入事例から得られた、いわば"DB クラスタ構築のベストプラクティス"を集約したものだ。上述した各要素をどのように組み合わせ、どのような設定を施すことで最良の結果が得られるか、そのエッセンスが示されている。

Simple Cluster の高速フェイルオーバー

Simple Cluster の最大の特徴は、非常に高速なフェイルオーバーを、できるだけシンプルな構成で実現することである。
では、具体的にどの程度のスピードでフェイルオーバーが可能なのだろうか。それを示す前に、フェイルオーバー時間の構成要素を考えてみよう。フェイルオーバーでは、大きく分けて以下の 2 種類の処理が実行される。

- クラスタ再構成
- リカバリ処理

「クラスタ再構成」とは、障害を検知し、クラスタ上のどのノードが処理を引き継ぐかを決定するプロセスである(詳細については、以前の特集「HA クラスタの疑問を解く・後編」を参照していただきたい)。一方、「リカバリ処理」としては、共有ディスクのマウントや DB インスタンスの起動、そして DB トランザクションのリカバリが含まれ、DB の規模によって変化する。
Simple Cluster では、これらのうちクラスタ再構成にかかる時間を 6 秒にまで短縮することに成功している。Serviceguard やその他の一般的なクラスタウェアによるフェイルオーバーでは、通常は数 10 秒を要する処理である。なぜこうした高速化が可能になるのか、以下の図を用いて説明しよう。

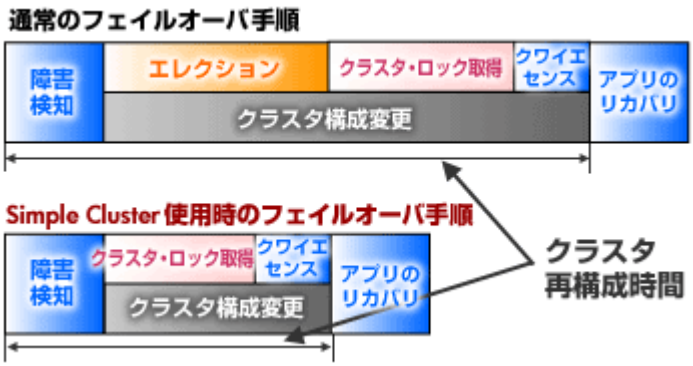


図 4：Simple Cluster によるフェイルオーバー手順

まず Simple Cluster では、従来は必要とされた「エレクション」の手順を省略している。「エレクション(election)」とは、障害検出後に生き残ったノードが「新たに形成されるクラスタに参加する権利」を獲得するプロセスである。多数のノードからなる複雑なクラスタでは、このエレクションの手順が不可欠となる。一方、Simple Cluster ではクラスタのノード数を 2 ノードに制限することで、信頼性を確保したままエレクションの手順を省略している。
もうひとつの高速化のポイントは、「クラスタ・ロック取得」である。クラスタ・ロックは、いわゆる「スプリット・ブレイン(split brain)現象」を避けるために欠かせないメカニズムである。一般的には共有ディスクによりクラスタ・ロックを実装する方法が広く用いられるが、その場合、クラスタ・ロック取得には時間を要してしまう。そこで Simple Cluster では、クラスタ・ロック取得を瞬時に完了できる Quorum Server を導入し、このプロセスに要する時間を短縮している。
Simple Cluster では、これらの最適化によって、従来に比べてクラスタ再構成時間の大幅な短縮を実現している。つづく後編では、Simple Cluster の導入による効果を検証したい。

DB クラスタ構築でハマらないための「HA Simple Cluster リファレンス・アーキテクチャ　v.1.0」・後編

2006 年 8 月　テクニカルライター 吉川和巳

これまで多くの案件において、フェイルオーバー時間をいかに高速に実現するかが論議されてきた。したがって IT エンジニアにとっては、「Simple Cluster が実際にどの程度の速さでフェイルオーバーできるのか」が、同ソリューションを選択する上で最大の関心事となる。この点を明らかにするため、HPE では 1 TB のデータを保持する DB を対象とした実地検証を行った。この検証により、「Simple Cluster は、非常に高速なフェイルオーバーを実現可能」ということが明らかにされた。Simple Cluster の導入によって得られるメリットとデメリットについて考察する。

Simple Cluster 導入の効果

これまで多くの案件において、フェイルオーバー時間をいかに高速に実現するかが論議されてきた。したがって IT エンジニアにとっては、「Simple Cluster が実際にどの程度の速さでフェイルオーバーできるのか」が、同ソリューションを選択する上で最大の関心事となるはずだ。

この点を明らかにするため、HPE では実地検証を行った。具体的には、1 TB のデータを保持した Oracle データベースを作成し、高負荷の OLTP トランザクションの実行中にノード障害を発生させ、そのフェイルオーバー時間を計測するという試験である。このときのフェイルオーバー時間とは、「障害発生からスタンバイ機での Oracle の起動終了までに要する時間」を意味する。ちなみに「Oracle の起動」では、以下の一連の処理が Oracle 内部で実行される。よって、これらすべての手順が終了した時点で、フェイルオーバーが完了したと見なされる。

表：Oracle の起動手順

コマンド	処理名	処理内容
startup nomount	Oracle インスタンス起動	・初期化パラメータの読み込み ・SGA の割り当て ・バックグラウンド・プロセスの起動
alter database mount	Oracle データベースのマウント	・制御ファイルの読み込み
alter database open	Oracle データベースのオープン	・データファイルの読み込み ・オンライン REDO ログファイルの読み込み ・必要に応じてクラッシュ・リカバリ (ロールフォワード、ロールバック)

なお今回の検証では、以下の環境が用いられた。

- DB サーバ―― Integrity Superdome(16 CPU、96 GB メモリ)
- ストレージ―― StorageWorks XP12000
- OS――HP-UX 11i v2 11.23
- クラスタウェア―― Serviceguard 11.17、SGeFF 1.00
- データベース――Oracle Database 10g R2(10.2.0.2) (データベース・サイズ 1 TB、SGA サイズ 64GB)

では、検証結果を見てみたい。

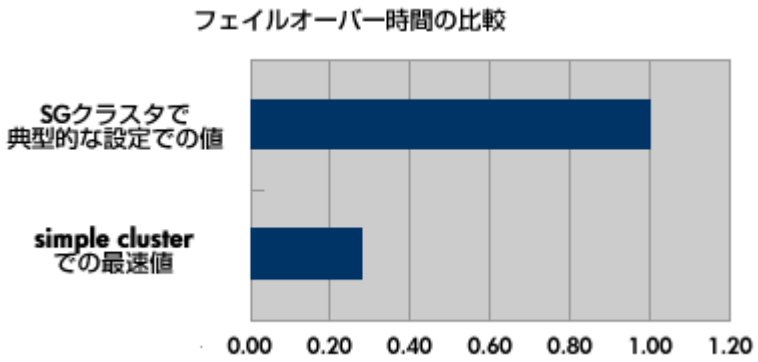


図 1：Simple Cluster によるフェイルオーバー時間

図 1 は、Oracle を用いて、従来の SG クラスタの典型的な設定で測定したフェイルオーバーの時間と Simple Cluster での最速のフェイルオーバー時間を比較したものである。この結果から分かるとおり、1 TB という大規模な OLTP システムに関わらず、Oracle の起動を含めた Simple Cluster でのフェイルオーバー時間は従来のフェイルオーバー時間の約 27%に短縮されている(ただし Oracle データベースのオープンに要する時間はクラッシュ・リカバリの状況に応じて変化する)。
この結果から言えることは、「Simple Cluster は、非常に高速なフェイルオーバーが可能」ということだ。その理由としては、以下の 2 点が挙げられる。ひとつは、前編で説明した最適化によって、6 秒という非常に短い時間での障害検出とクラスタ再構成が実現したこと。もうひとつは、近年のハードウェアやソフトウェアの大幅な性能向上により、以前であれば長時間を要していた Oracle の起動が短時間で終了すること。とりわけ後者のおかげで、スタンバイ機で Oracle インスタンスを起動する時間が気にならなくなったことが大きいと言えるだろう。

Simple Cluster がもたらすもの

この検証結果によって、負荷分散を必要としない環境では、Simple Cluster が十分に魅力的な選択肢であることが理解いただけたはずだ。では、Simple Cluster の導入によって得られるメリットを考えてみよう。
まず設計段階では、業務からインフラまでのすべてを理解する有能な IT アーキテクトが不要となる。Simple Cluster の構築手順を理解しているエンジニアであれば、DB クラスタの設計が可能だ。また負荷分散を行わない Simple Cluster では、「DB がダウンしたらすぐさま別ノードで DB を起動する」という単純な切り替えを実施しているに過ぎない。よってアプリケーション側から見ればスタンドアロンの DB であり、クラスタ化のための設計上の配慮は不要となる。
次に構築段階では、IT システムを構成するコンポーネントの数が減り、少数の担当者によるスピーディーな連携が可能だ。また Simple Cluster は単純な構成であるがゆえに DB クラスタとしての実績も豊富で、構築手順ドキュメントに従って作業を進めておけば、予想外のトラブルを招く可能性も少ない。さらに構成がシンプルであるため、実運用環境と同等の開発環境をそろえることも容易となる。
そして試験段階や運用段階でも、試験項目の削減、運用手順の定型化、運用スタッフに要求されるスキルの低減といった効果がある。またプロジェクト工程全体としては、インフラ構築のための協議や試行錯誤、トラブルが減ることで、その時間をアプリケーションの試験に費やせるようになり、カットオーバーまでの時間が短縮できるようになる。

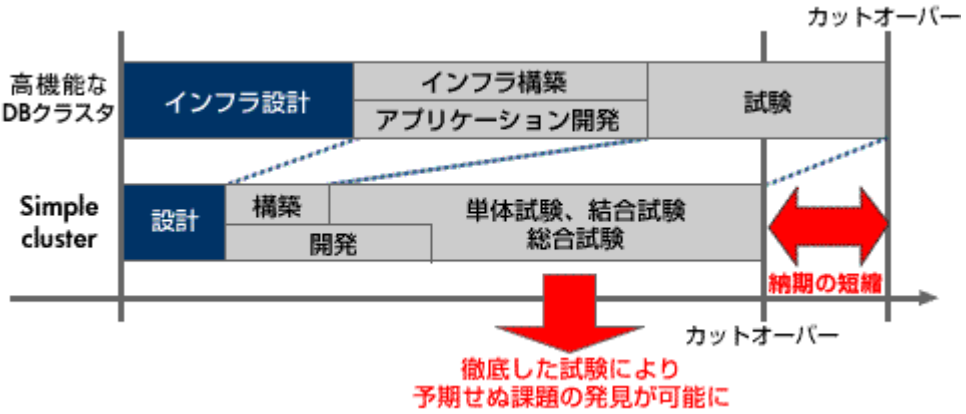


図 2：Simple Cluster 導入によるプロジェクト工程の変化

つづいては、Simple Cluster の導入時に検討すべきポイントを概観する。

Simple Cluster 導入のポイント

ここまでは Simple Cluster のメリットを見てきたが、最後に Simple Cluster 導入に際して考慮すべきポイントを説明したい。前編で述べたとおり、Simple Cluster では Serviceguard の高速フェイルオーバー・オプションである SGeFF を利用している。この SGeFF では、DB クラスタの構築に際して以下のような制約を課している。

- クラスタサイズは最大 2 ノードまで
- クラスタ・ロックとして Quorum Server を用いる
- 2 系統以上のハートビート LAN を用意する

Simple Cluster では、クラスタの構成ノード数を 2 ノードに制限することで、前編で述べたクラスタ再構成手順の最適化を実現している。またクラスタ・ロックとしては Quorum Server を用いることが必須となる。ただ、同サーバは非常に軽量なソフトウェアであり、また物理的に独立したサーバを用意する必要もない。したがって、通常はアプリケーション・サーバなどに間借りすることが可能だ。また、Quorum Server がダウンしても Simple Cluster は Quorum Server がダウンしたログを出すだけで、クラスタは動作し続ける。
さらに Simple Cluster では、ハートビート LAN(ノード相互の死活監視用の LAN)を 2 系統以上持たなければならない(ただし、それぞれの LAN がスタンバイ LAN を持つ必要はない)。これは 6 秒という高速な障害検出とクラスタ再構成には欠かせない要件となる。

アクティブ/スタンバイ方式のメリット

Simple Cluster の導入に際してネックとなりうる点は、おそらくスタンバイ機の確保だろう。Simple Cluster は、2 ノードからなるアクティブ/スタンバイ方式の DB クラスタである。よってスタンバイ機はトラブルが発生するまで待機状態にあり、サーバ・リソースを持て余すことになる。
もっとも、アクティブ/スタンバイ方式にもメリットは存在する。例えば障害時の縮退運転が許容される場合は、プライマリ機とまったく同じ構成のハードウェアをスタンバイ機に用意しておく必要はない。たとえば 1/2 規模の構成にしたり、テスト機をスタンバイ機と兼用したりすることができる。またデータベース製品やアプリケーション製品には、スタンバイ機についてライセンス費用や保守費用が発生しないものも少なくない。この場合、DB サーバ 1 台分のライセンス・コストで DB クラスタを構築できる。

Simple Cluster 導入のベストプラクティス

HPE では、Simple Cluster 構築に際して、高速なフェイルオーバーを実現するためのベストプラクティスとして、以下のポイントを提示している。

- マウント・ポイントを少なくする
- Oracle の設定の最適化

まず、DB サーバと共有ディスク間のマウント・ポイント数をできるだけ少なくするのが、高速フェイルオーバーのコツである。HP-UX では、ジャーナル・ファイルシステム JFS の採用により、マウント時の整合性チェック(fsck)が高速化している。とはいえ、やはりマウントには数秒を要する。よってマウント・ポイントが増えるほど、フェイルオーバー時間も延びるのである。
以前の DB サーバ構築では、ディスクアレイではなく単体ディスクの利用が主流であったため、パフォーマンス向上のためにマウント・ポイントを細かく分けることも多かった。しかし SAN によるディスクアレイ利用が主流の今日では、マウント・ポイントを分ける意義はそれほどない。例えば StorageWorks XP シリーズの OPEN-V、または EVA シリーズの Vraid といった技術を使って、大容量の論理ユニットをそのまま単一のマウント・ポイントにする方法が考えられる。
一方、Oracle の設定を最適化することも高速なフェイルオーバーに不可欠となる。例えば raw デバイス向けの設定オプション「disk_asynch_io」を FALSE に設定したり、クラッシュ・リカバリ時のオンライン REDO 読み込み量を設定する「fast_start_mttr_target」を最小値にしたり、といった作業である。これらについて詳しくは、Simple Cluster の構築手順ガイドを参照していただきたい。

以上、今回は Simple Cluster が提案するシンプルな DB クラスタの構築手法を紹介した。本当に必要とされる機能だけを実装するのであれば、DB クラスタは意外に簡単に、かつローコストに構築できる。Simple Cluster は、IT アーキテクトにそうした"新たな発見"を与えてくれるソリューションと言えるだろう。HPE では Simple Cluster の構築支援を、エクスプレスサービス*の一部として提供できるように準備を進めている。

*注) HPE のエクスプレスサービスは、HPE がこれまでに培ったコンサルティングのノウハウや経験、技術を集約し、ご提供するサービス商品です。

HA クラスタの達人に聞く「Serviceguard が選ばれる理由」

HA クラスタの達人に聞く「Serviceguard が選ばれる理由」・前編

2006 年 9 月　テクニカルライター 吉川和巳

「ミッションクリティカル・システムの 4 台に 1 台は、クラスタウェア Serviceguard を搭載している」と述べるのは、過去 15 年余にわたって Serviceguard の開発を指揮してきた HPE のアーキテクト、ボブ・サワーズ氏である。同氏は、高可用性クラスタ(HA クラスタ)分野における Serviceguard の圧倒的な実績が、通信事業者をはじめ、厳格な可用性を求める企業のニーズに応えてきた結果であると説明する。今回は、HA クラスタ・ソリューションとしての Serviceguard の確固たる優位性がどのような経緯で培われてきたのかを明らかにすべく、この分野の第一人者であるボブ・サワーズ氏に話を伺った。

SwitchOver/UX 誕生のいきさつ

Serviceguard の前身は、80 年代後半に開発された HP-UX 対応クラスタウェア SwitchOver/UX である。サワーズ氏は、SwitchOver/UX 誕生のいきさつについて次のように振り返る。

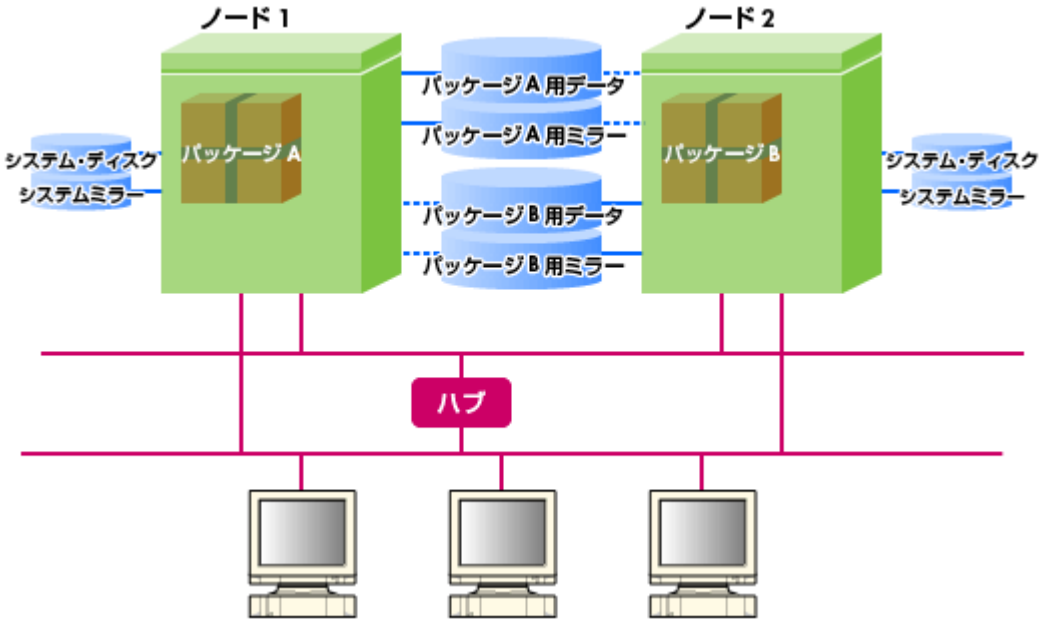
HPE アーキテクト
ボブ・サワーズ氏

「SwitchOver/UX は第一世代の HA クラスタ製品です。その当時の HA クラスタ製品開発のおもな動機は、ハードウェア障害への対処でした。HPE のサーバ製品は当時も十分な信頼性を備えていましたが、とりわけ通信分野ではさらなる高信頼性が必要とされていたことも、SwitchOver/UX の開発を後押しする要因となりました。実際に、初期の導入事例としてもっとも多かったのが通信事業者でした。2 番目に多かったのは、データベースを運用する大企業です。当時は、この 2 つの用途に向けて HA クラスタのソリューションを提供していました。

年を追うごとにハードウェアの信頼性も向上しましたが、ハードウェア障害の可能性をゼロにすることはできません。またもちろん、ハードウェア障害だけがシステムダウンの要因ではなく、ソフトウェア障害やヒューマンエラーも原因となり得ます。統計では、現在のシステムダウンの 2 割がハードウェア障害によるもので、4 割がソフトウェア障害、そして残る 4 割がヒューマンエラーによるものとされています。一方、SwitchOver/UX が登場した当時は、システムダウンの 8 割がハードウェア障害に起因していました。現在とはまったく異なる状況だったのです」

また SwitchOver/UX の機能も、現在の Serviceguard とは大きく異なるものであったという。

「例えば SwitchOver/UX では、障害の発生したサーバからサービスを引き継ぐために、サーバのリブートが必要でした。したがって、フェイルオーバには 15 分～30 分もの時間を要していました。一方、現在の Serviceguard ではリブートの必要がなく、数 10 秒という格段に短い時間でフェイルオーバが可能になっています。もう一つ、当時の SwitchOver/UX が大きく異なる点は、それが『アクティブ/スタンバイ形式』にのみ対応しており、待機状態のスタンバイ・サーバを必ず必要としたことです。これに対し Serviceguard では、クラスタ上のすべてのノードを実運用に利用できる『アクティブ/アクティブ形式』にも対応します」



Serviceguard による HA クラスタの構成例

通信分野に HP-UX が浸透

この SwitchOver/UX の成功をきっかけに、HP-UX は通信分野でのミッションクリティカル・システム構築に広く浸透し始めていく。

「通常、電話交換機そのものには、わずかな大きさのメモリしか搭載されていません。したがって、電話網の運用に必要な大量のデータの管理は、他のシステムに任されます。例えば HP-UX が通信事業者へ導入され始めた当初は、電話網を監視するシステムに利用されていました。また当時は、課金システムの大半がメインフレームによって構築されていました。

当時と現在で異なる点は、通信事業者が大半のアプリケーションをメインフレームからハイエンドの HP-UX システムへと移行しつつあることです。なぜなら、メインフレームに比べて、HP-UX 上ではよりすばやく簡単にアプリケーションを開発でき、さらにハードウェアのコストも抑えられるからです。例えば課金システムについても、その後はハイエンド UNIX システムへの移行が進み、現在では通信事業者向けに出荷される Serviceguard の大半が課金システムに用いられており、通話記録などの情報が HP-UX ベースのシステムで管理されています。

現在の通信分野における HP-UX のもうひとつの用途は、CRM(Customer Relationship Management)システムやコールセンター・システムです。例えば、加入者が引っ越しをして新しい住まいに電話を移設する場合を考えてみます。加入者が通信事業者

に移設を依頼すると、オペレータは CRM システムを通じて新しい住所の登録などを行います。これらの情報はプロビジョニング・システムに渡され、そこから個々の交換機に対して、新しい電話番号は何番か、そしてどの回線がその番号に割り当てられたのかといった情報が届きます」

もちろん、こうしたメインフレームから UNIX へのオープン化は、一朝一夕になし得たものではないとサワーズ氏は説明する。

「HPE では、長い時間をかけてハードウェアの信頼性を高めてきたことに加えて、万が一のハードウェア障害に対しても、Serviceguard により他のノード上でアプリケーションをすばやく再起動できることを示してきました。課金システムなどに求められる可用性は 99.999%程度で、許容ダウンタイムは 5 分間に相当します。ちなみに、電話交換機に求められる可用性は 99.99999%程度であり、1 年間に許容されるダウンタイムが数秒間です」

ひきつづき後半では、Serviceguard をつらぬく設計思想について、サワーズ氏に話を伺う。

Serviceguard をつらぬく設計思想

Serviceguard を設計する上で、サワーズ氏がもっとも重視したポイントは何か。その点について、同氏は次のように述べる。

「HA 環境を設計する上で考慮すべきポイントは、『データの整合性』、『可用性』、そして『パフォーマンス』の 3 つです。これらすべてを同時に最大限に満たすことは困難であるため、これらのうちのいずれが最重要であるか、優先付けが必要になります。

HPE では、伝統的に、データの整合性をもっとも重要な設計要件として位置づけてきました。なぜなら、整合性の失われたテラバイト単位のデータを復元するよりも、システムを再起動する方が格段に短い時間で済むからです。もしデータの整合性が失われると、バックアップされたデータを読み込み直す必要があり、システム復旧までに何日もの時間がかかるでしょう。一方、システムの再起動は、ほんの 2〜3 分で済みます。このデータの整合性こそが、HA クラスタ分野における Serviceguard の大きな優位性になっています。

例えば、あるノードのアプリケーションや OS に非常に負荷がかかり、クラスタの他のノードに対してハートビート（動作状況を通知する間欠的なパケット）を送信できず、ハングアップしたかのように見えるケースがあります。こうした場合、クラスタのフェイルオーバーを速やかに実施するには、実際にはノードで障害が起きていない場合でも、そのノードをクラスタから外し、データの書き込みを停止させます。こうした措置は、いわゆる『スプリット・ブレイン』と呼ばれる現象を防ぐために必要です」

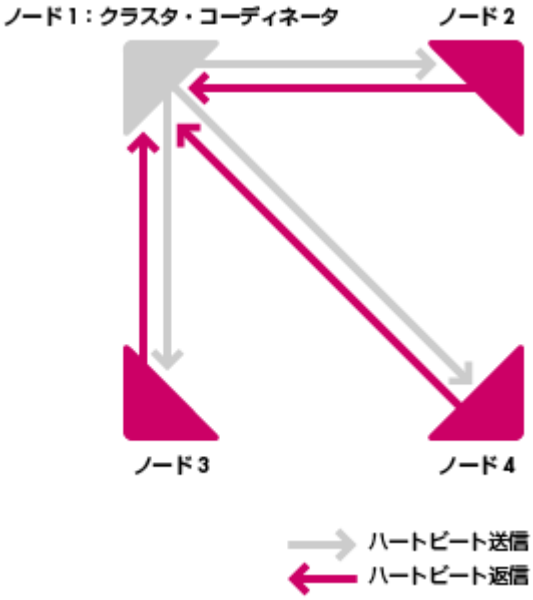


図 2：ノード間のハートビートの送受信

スプリット・ブレインとは、何らかの理由により、クラスタ全体が 2 つのグループに分裂してしまう状態を指す。それぞれのグループが、自分こそがクラスタの主導権を握っていると認識してしまうのである。こうした状態のままディスクへの書き込みを行うと、データは壊れてしまう。Serviceguard では、このスプリット・ブレインの発生を許さない設計になっているという。

「実のところ、ベンダーによっては、こうしたデータ整合性よりも可用性を重視する企業もあります。つまり、とにかく高速なフェイルオーバーを実現し、『データが破損する可能性は小さいですよ』で済ましてしまうのです。HPE では、こうした考え方はしません。我々にとって、データの破損はできるだけ避けるべきことなのです。ミッションクリティカル分野のお客様は、データの破損を許容することができません」

Serviceguard の進化

Serviceguard は、現在でも着実な進化を遂げつつある。そのひとつが、きわめて高速なフェイルオーバを実現するオプション Serviceguard Extension for Faster Failover (以下、SGeFF) である。サワーズ氏は、この SGeFF もやはり通信分野のニーズに応じて生まれた製品であると説明する。

「Serviceguard は、例えば Oracle Real Application Clusters(RAC)をはじめとする多種多様なアプリケーションに対応し、あらゆる目的に沿うように設計されています。一方、現在の通信分野では、一段と高速なフェイルオーバが求められています。通信分野は、ある意味で特殊な用途であり、従来の Serviceguard のような汎用のソリューションでは必ずしも最良の能力を提供できないこともありました。そこで HPE では、Oracle RAC などにも対応する汎用性と、通信事業者のニーズにも応える高速性を両立させるべく、Serviceguard に改良を加えました。それが、SGeFF です。

SGeFF では、より高速なフェイルオーバを実現するために、クラスタのノード数が 2 台に限定されます。とはいえ、現在の HA クラスタでは、2 ノード構成がもっとも広く用いられています。10 年前であればほぼすべての HA クラスタが 2 ノード構成であり、現在でもおよそ 40～50%が 2 ノード構成です。またミッションクリティカル分野では、3～4 ノード構成の HA クラスタも一般的になりつつあります」

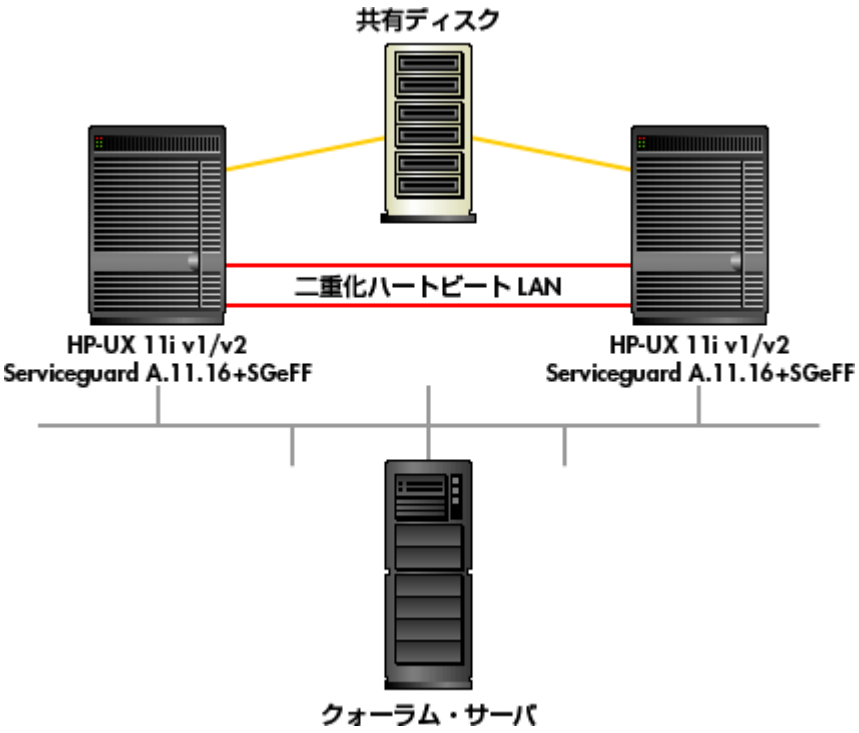


図 3：SGeFF の構成

サーバ・コンソリデーションへの応用

サワーズ氏によれば、Serviceguard の大規模事例の多くは、サーバ・コンソリデーションへの応用が中心であるという。

> 「最近では、コンソリデーションがとても重要な課題になっています。どの IT 部門にも共通する大きな問題は、設置スペース、電源容量、設備コスト、そして運用コストの 4 点です。とりわけ日本では、多くのお客様が電源容量の問題に直面していると聞いています。
>
> Serviceguard のユーザの 2～5%は、最大構成の 16 ノードの HA クラスタを運用しています。こうした大規模な HA クラスタのおもな用途は、サーバ・コンソリデーションです。HA クラスタを特定の用途にのみ用いるのではなく、多彩なアプリケーションを抱える大規模な HA クラスタを構築するというアプローチです」

後編では、Serviceguard と密接な関わりを持つデータベースである Oracle との連携について話を伺う。

HA クラスタの達人に聞く「Serviceguard が選ばれる理由」・後編

2006 年 9 月　テクニカルライター 吉川和巳

Serviceguard を語るとき、サワーズ氏が強調する点は、HP-UX や Integrity サーバとの親和性の高さである。例えば、I/O カードのバッファに残留したデータがディスクに書き込まれる現象「ゴースト I/O」への対処といった機能など、ハードウェアからカーネル、クラスタウェアまですべてを自社開発する HPE ならではのアドバンテージが現れているという。さらにインタビュー

では、Serviceguard とは切っても切れない関係である Oracle との連携について、サワーズ氏が教えるノウハウや勘どころを伺った。

HP-UX だからできること

サワーズ氏は、Linux ベースの HA クラスタと比較しながら、HPE が提供するソリューションのアドバンテージを説明する。

「Linux はオープンソース製品であるため、HA クラスタの機能が Linux カーネルや Linux 市場全体に大きく依存してしまうのです。これに対し、HP-UX のようなベンダー製 OS の場合、システムの可用性と堅牢性を高める特定の機能を追加したいとき、すぐさまカーネルに変更を加えられます。

例えば、まもなく登場する最新バージョンの HP-UX である HP-UX 11i v3 では、オンラインでのリソース追加・削除機能を更に強化する予定です。セルボード、さらに I/O やメモリ、そして CPU といったコンポーネントのマイグレーションがパーティショニング環境上で実施できます。そのため、システムの運用を継続したまま、OS をリブートすることなくコンポーネントの増強を行えます。こうした機能を実現するには、OS カーネルに対して多大な投資を行わなくてはなりません。そのためオープンソースとしての Linux では、オンラインでのリソースマイグレーション機能はベンダーが独自にカーネルを変更する必要があり、オープンとしての実装は難しいのが現状です」

このように、クラスタウェアである Serviceguard のみならず、OS である HP-UX やハードウェアである Integrity サーバにも多くの労力をつぎ込み、堅牢性と可用性の向上に努めている点が HPE の優位性であると、サワーズ氏は強調する。

図 1： Integrity サーバの信頼性機能

「もうひとつの例が、『ゴースト I/O』と呼ばれる現象への対処です。これは、ハードウェア障害の中でも例外的なケースで発生するもので、アプリケーション・プロセスや OS カーネルが停止した状態にもかかわらず、I/O カードがバッファ上の残留データをディスクへ書き込み続けてしまう状況を指します。ドライバ・ソフトウェアに対する通常の操作だけでは、このゴースト I/O は防げません。そこで HP-UX 版の Serviceguard では、非常に低レベルのハードウェア・コマンドを I/O カードに直接送信

し、ディスクへの書き込み処理を直ちに停止するよう指示します。これにより、データの破損を防止する仕組みです。こうしたメカニズムはカーネルとハードウェアが直接連携する必要があり、Linux のような環境では実現できません。

Linux が動作するハードウェアはさまざまなベンダーによって開発されていますが、Linux そのものはオープンソース・コミュニティによって開発されています。つまりハードウェア・ベンダー各社は、それぞれに異なる Linux を開発するわけにはいきません。そのため、ベンダー独自の機能を実装するのが難しく、特別な高信頼性機能などを提供することで差別化を図るのが困難なのです」

ひきつづき後半では、Serviceguard とは切っても切れない関係である Oracle との連携について、サワーズ氏に話を伺う。

Serviceguard と Oracle の関係

サワーズ氏へのインタビューの締めくくりとして、Serviceguard とは切っても切れない関係である Oracle との連携について話を伺った。

「Serviceguard と Oracle の連携には、とても長い歴史があります。1980 年代後半に Oracle が普及し始め、その時期から Serviceguard と Oracle のインテグレーションを進めてきました。

こうした統合は、『Serviceguard とアプリケーションの統合』と呼ばれる試みで、Serviceguard の動作環境でアプリケーションを単純に動作させるだけではなく、アプリケーションのインテリジェントな制御も目指したものです。これに対して、例えばマイクロソフトのクラスタ・サーバでは、クラスタ環境の能力を引き出すためにアプリケーション側でクラスタを意識した設計を行う必要があります。つまり、マイクロソフトのクラスタ・サーバ環境を理解できるアプリケーションへと変更が必要です。一方、Serviceguard では、アプリケーションをあらかじめクラスタ対応に変更しておく必要がなく、そのままで Serviceguard と非常にうまく連携できます」

ご存じのとおり、現在の Oracle は先進的な DB クラスタ・ソリューション Oracle Real Application Cluster s(以下、Oracle RAC)を備える。この Oracle RAC の登場によって、HA クラスタ構築が大きく変化したとサワーズ氏は説明する。

「Oracle RAC を利用した HA クラスタのアドバンテージは、高速なフェイルオーバーが可能な点です。Oracle RAC を用いない場合、スタンバイ・ノードにおいて Oracle のインスタンスを起動し、リカバリ処理をスタートさせます。このリカバリ処理の量に応じて、フェイルオーバーには通常 5〜10 分といった時間がかかります。これに対し、Oracle RAC では、20〜30 秒でのフェイルオーバーが可能です。たとえ負荷分散を用いない場合でも、こうした高速なフェイルオーバーが Oracle RAC のメリットと言えます」

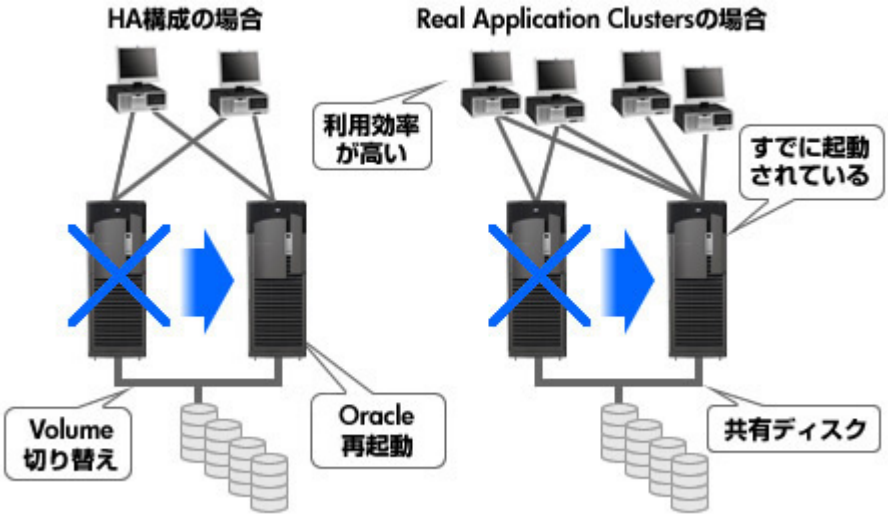


図 2：Oracle RAC による高速なフェイルオーバー

一方、Serviceguard では、Oracle RAC との連携のためのオプション Serviceguard Extension for RAC（SGeRAC）を提供している。

「SGeRAC は、もともと『MC/Lock Manager』と呼ばれる製品として 1995 年にリリースされました。これは当時の Oracle 7 のクラスタ・ソリューションであった OPS (Oracle Parallel Server) に対応するものです。その後、オラクルが OPS の名称を RAC と変更したのにあわせて、MC/Lock Manager も SGeRAC へと名前が変わりました。したがって現在の SGeRAC は、10 年にわたる長い Oracle との連携の歴史を持っているのです。このように HPE では、オラクルとの間で技術面での密接な関係を持っています。実際、我々のリクエストに応じて、Oracle RAC の信頼性をより高める機能拡張もいくつか行われました」

Oracle RAC＋SGeRAC を使いこなす

Oracle 10g RAC と Serviceguard との連携ももちろん可能だ。では、Oracle RAC と SGeRAC を組み合わせることで、どのようなメリットが生まれるのだろうか。サワーズ氏は次のように答える。

「Oracle RAC と SGeRAC の連携によって、大きく 4 つの追加機能が利用可能になります。ひとつは、データにアクセスすべきでないノードの I/O をストップする『I/O フェンシング』。この I/O フェンシングもまた、データの整合性を確保するための技術です。また、もうひとつの追加機能として、ネットワークの監視とネットワーク障害にともなうフェイルオーバーが可能です。その他、ストレージ管理の柔軟性向上や、仮想化との統合などの追加機能を提供できます。これらの 4 つのおもな能力は、Oracle RAC に付属するクラスタウェアに対して、Serviceguard と SGeRAC が提供する付加価値と言えます」

スケールアップで実現する大規模 RAC システム

また、Oracle RAC によるクラスタ構築においてサワーズ氏が指摘するのは、スケールアウト構成の問題点だ。

「Oracle RAC については、クラスタの構成として『スケールアップ』と『スケールアウト』のいずれを選択すべきか、という議論があります。スケールアップとは Oracle を少数のハイエンド・サーバで運用する構成であり、一方のスケールアウトとは多数のエントリーレベル・サーバで分散運用する構成を指します。

しかし現実には、アプリケーションの変更を行わずにスケールアウトできる例はきわめてまれなのが実情です。スケールアウト構成では、アプリケーションの変更を行わない限り、クラスタ上の各ノードが同一のデータ領域にアクセスしようとする競合問題が発生します。そのため、実際に Linux ベースの多数の PC サーバに Oracle RAC を導入しようとすると、それほどうまくスケールアウトしないことが明らかになります。

例えば Serviceguard と SGeRAC の組み合わせでは、最大で 16 ノードによる Oracle RAC をサポートできます。しかし、実際にこれほどの規模で Oracle RAC を構成するユーザは皆無です。私が知る限り、Oracle RAC のもっとも大規模な導入事例でも 8 ノードであり、通常は 4 ノードかそれ以下のノードにより構成されています。つまりアプリケーションにとっては、スケールアウトよりもスケールアップの方が格段に簡単なのです。そうなると、アプリケーションの再設計に多大な投資を行うか、スケールアップ構成へ移行するしか方法はありません。スケールアップ構成では、アプリケーションを再設計する必要がない上に、より高度なスケーラビリティが得られます。

HPE の大きなアドバンテージは、きわめて大型なハイエンド・サーバである Integrity Superdome を提供している点です。Superdome では、最大で 64 プロセッサ/128 コアまでスケールアップが可能です。このスケールアップによって、Oracle RAC のパフォーマンスとスケーラビリティを最大限に引き出すことが可能なのです。また、2 台の Superdome によりクラスタを構成することで、さらなる可用性を得ることができます」

以上、今回はボブ・サワーズ氏へのインタビューを通じて、ミッションクリティカル分野における Serviceguard のアドバンテージを明らかにした。同氏が繰り返し説明するように、HA クラスタ・ソリューションとしての Serviceguard の優位性は、HP-UX および Integrity サーバを含めたシステム全体での可用性の高さにあると言えるだろう。オープンソース全盛の今日でも、ベンダー・ソリューションでしか実現し得ないものもあることを理解していただけたはずだ。

暗号化セキュリティで重要情報を保護

暗号化セキュリティで重要情報を保護・前編

2006 年 10 月　テクニカルライター 吉川和巳

セキュリティ保護の“最後の砦”として用いられるのが、暗号化ボリュームや暗号化ファイルシステムである。HPE では、暗号化ボリューム＆ファイルシステム「EVFS(Encrypted Volume & Filesystem)」の開発を進めている。EVFS は、HP-UX の暗号化ボリュームおよび暗号化ファイルシステム機能を提供するもので、年内には無償で提供が開始される予定だ。EVFS によるデータの暗号化によって、これまで限られたサーバやストレージ上にしか置けなかった機密性の高いデータであっても、SAN や仮想化環境といったあらゆる場所に保管できるようになる。

“最後の砦”となる暗号化ファイルシステム

ひとことに「セキュリティ」と言っても、そこにはさまざまな意味が含まれている。まずは、以下の図を見ていただきたい。

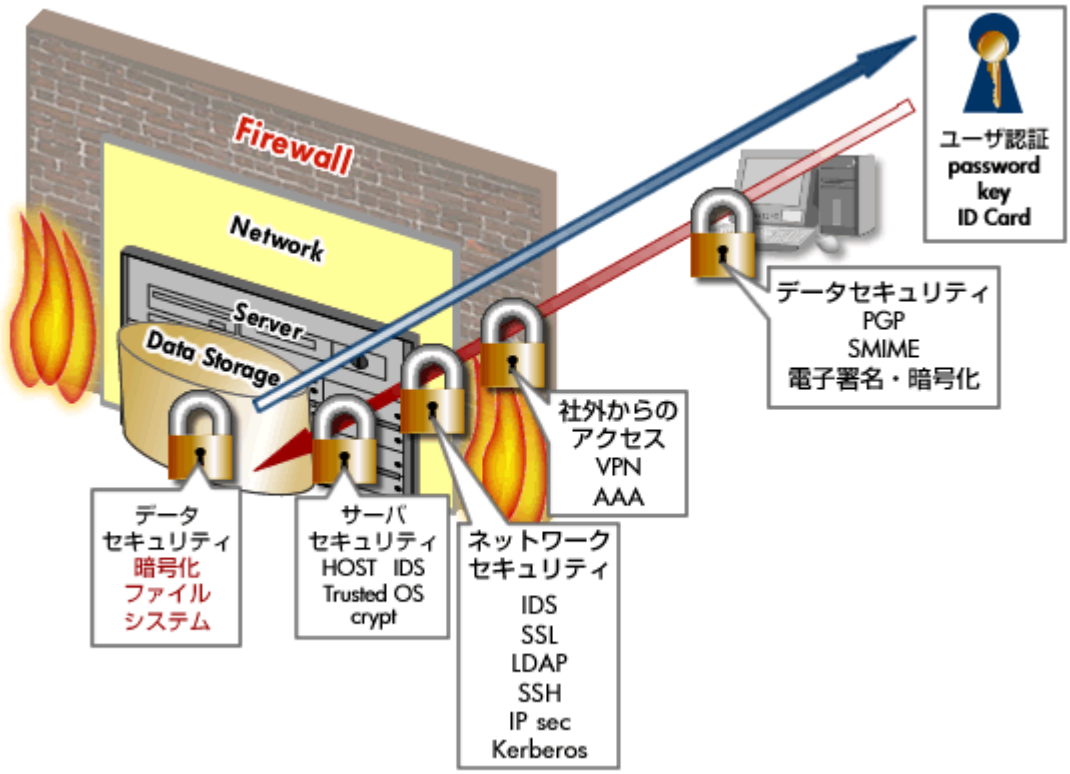


図 1：UNIX システムで用いられるさまざまなセキュリティ技術

この図は、UNIX システムで一般的に用いられているセキュリティ技術を概観したものだ。例えばユーザ認証には、ユーザ名とパスワードによるシンプルなログイン認証をはじめ、Kerberos や AAA、LDAP といった仕組みが用いられる(参考：コラム「LDAP、Kerberos、RADIUS で統合認証を自在にあやつる HP-UX」)。また、システムやネットワークの外部との通信の秘匿性を確保する手段としては、ssh や SSL、IPsec、SMIME などの利用が一般的だ。さらには IDS のように不正アクセスの検知と対処を目的としたセキュリティ技術もある。

このように、セキュリティには複合的な側面があり、いずれかひとつの技術だけですべてをカバーすることは不可能だ。例えば「ウチのシステムは SSL に対応しているからセキュリティは万全だ」とは言えないのである。SSL によりネットワーク上のデータ傍受を防いだとしても、誰かがサーバのディスクを取り外してしまえば、重要データも簡単に読めてしまう。実際、メンテナンスやトラブル対応時に取り外したディスク・ドライブや、古いバックアップ・メディアについて、物理的な破棄まできちんと実施している企業は、意外に少ないのではないだろうか。

そこで、セキュリティ保護の“最後の砦”として用いられるのが、暗号化ファイルシステムである。つまり、ディスク上のデータそのものを暗号化することで、たとえディスクを物理的に取り外したとしても、データを複合化する「鍵」を持たない限り解読できないメカニズムである。この暗号化ファイルシステムを各種のセキュリティ技術と組み合わせることで、システム全体のセキュリティをより強固なものとすることができる。

EVFS とは

そうした暗号化ファイルシステムへのニーズに応えるべく、HPE では、暗号化ファイルシステム「EVFS(Encrypted Volume & Filesystem)」の開発を進めている。EVFS は、HP-UX の暗号化ボリューム機能および暗号化ファイルシステムを提供するもので、年内より無償で提供が開始される予定だ(暗号化ファイルシステムは 2007 年以降に提供予定)。

EVFS の特徴は以下の通りである。

ディスク上のデータを暗号化	EVFS では、ディスク上に記録するデータそのものを暗号化する。 そのため、もしディスク・ドライブやバックアップ・メディアの紛失や盗難があったとしても、データの保護が可能だ。
アプリケーション透過性	EVFS は、アプリケーションからは HP-UX の普通のファイルシステム(VxFS/HFS)やボリューム(LVM)として認識される。よって、アプリケーション側で特別な API を実装したり改変を行ったりする必要はない。
「ボリューム単位の保護」と「ファイル単位の保護」に対応	EVFS では、ファイルごとの暗号化とボリューム全体の暗号化という 2 通りのデータ保護機能を提供しており、用途に応じて選択できる。
故意や過失によるデータへの不用意なアクセスを防止	データにアクセスするには「鍵」と「パスフレーズ(パスワード)」が必要であるため、システム管理者が不用意にデータにアクセスすることを防止できる。

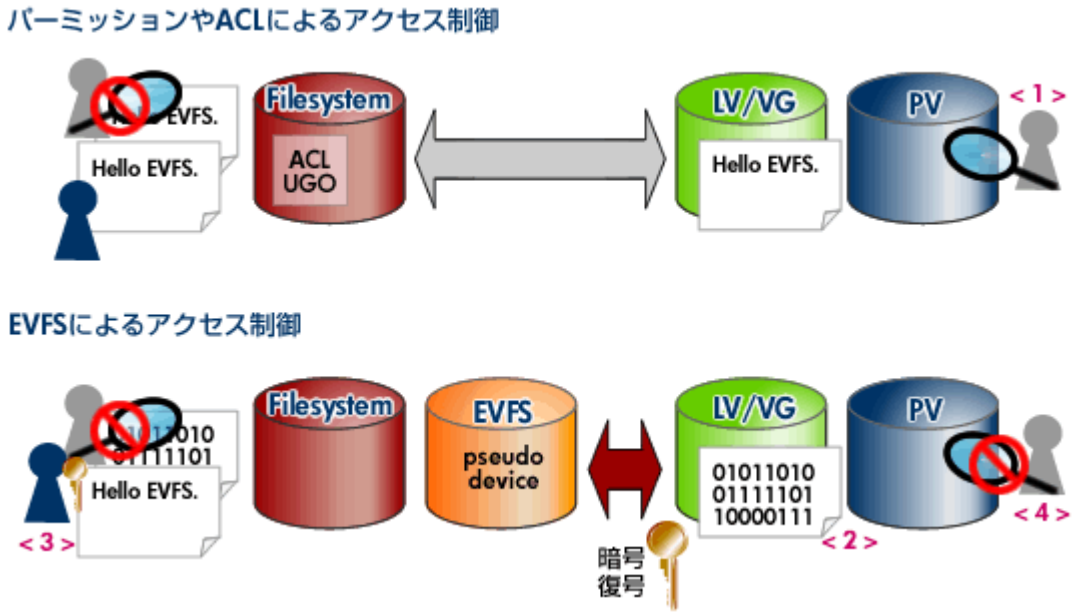


図 2：一般的なファイルシステムと EVFS の違い

図 2 は、ファイル単位の保護について、一般的なファイルシステムと EVFS の違いを示した図である。例えば HP-UX における VxFS や HFS などの一般的な UNIX ファイルシステムでは、ファイルやユーザの単位でパーミッションや ACL(アクセス制御リスト)を管理することで、ファイルへのアクセスを制限する。しかし、ディスク上のデータそのものが暗号化されているわけではない。そのため、特権を持つユーザやアプリケーションからアクセスする場合、ディスクそのものを取り外した場合、バックアップ・メディアから別ディスク上にリストアした場合などは、自由にデータにアクセスできる(図 2：<1>)。

一方、EVFS では、ディスク上に記録された個々のファイルの中身そのものが暗号化されている(図 2：<2>)。よって、適切な鍵を持つユーザやアプリケーションであれば、データを復号化することができる(図 2：<3>)。これに対し、たとえ特権を持つユーザでも、鍵を持たなければ物理的にどのような策を講じてもファイルの中身を見ることができない(図 2：<4>)。

ひきつづき後半では、この EVFS の内部メカニズムをもう少し詳しく見ていきたい。

EVFS の内部構造

図 3 は、EVFS 導入時のディスク I/O の流れを示した図である。

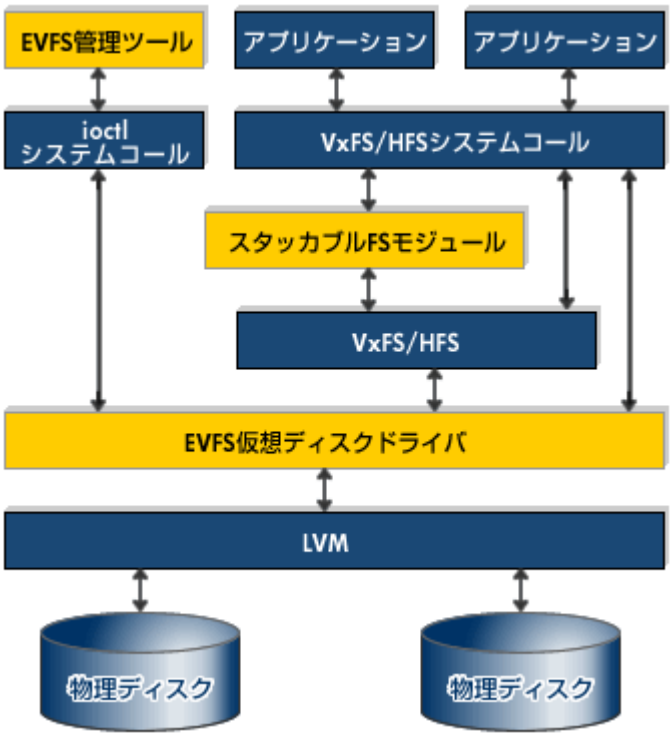


図 3：EVFS 導入時のディスク I/O の流れ

この図において、青色で示した部分は、HP-UX に従来から備わるファイルシステム(VxFS／HFS)やボリューム・マネージャ(LVM)を表している。一方、黄色の部分が、EVFS 導入時に追加されるコンポーネントである。この図が表すとおり、アプリケーション側ではこれまで通り VxFS または HFS のシステムコールを利用するだけであり、EVFS の存在を意識する必要はない。ファイル書き込み時の暗号化や、読み出し時の復号化は透過的に実行されるため、既存のアプリケーションをそのまま暗号化ファイルシステムに移行することが可能だ。

EVFS の暗号化機能を提供するのは、図 3 の中央に記された「EVFS 仮想ディスクドライバ(EVFS pseudo-disk driver)」である。同ドライバは、VxFS／HFS と LVM の中間に位置しており、LVM へのアクセス時にデータの暗号化を実行する。その名が示すとおり、仮想的なディスクドライバとしてふるまうため、VxFS や HFS といったファイルシステムには依存しない仕組みになっている。

また、図 3 左上に黄色で示された「EVFS 管理ツール」は、同ドライバの動作を制御するコマンドツール群である。EVFS における暗号化機能や鍵の管理は、すべてこれらのツールを通じて実施することになる。

「鍵」の管理メカニズム

EVFS では、いくつかの「鍵」を用いることで、ファイルシステムやボリュームの暗号化を実現している。その具体的なメカニズムを紹介したい。

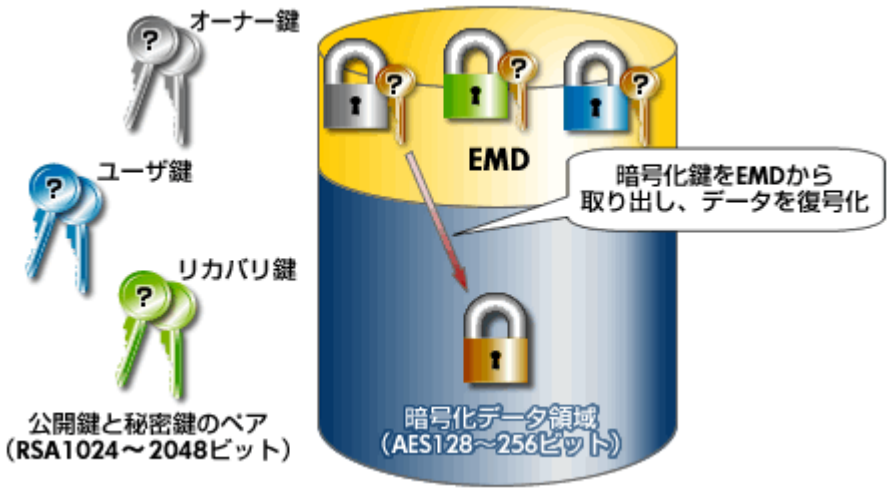


図 4：EVFS の鍵管理のメカニズム

図 4 に示すとおり、EVFS によって暗号化されたボリュームは「暗号化データ領域」と「EMD(Encryption Meta Data)」と呼ばれる 2 つの領域に分割される。前者は暗号化されたデータを保管する場所であり、後者は暗号化に用いる鍵を保管する場所として利用される。

EVFS では、データを暗号化する手段として、共通鍵暗号方式としてもっとも新しい業界標準である AES を採用している。AES は共通鍵暗号であるため、暗号化と復号化のいずれにも同じ暗号化鍵(128 ビット～256 ビット)を利用する。この暗号化鍵は、いわばデータにアクセスするためのパスワードのようなものだ(図 4 の金色の鍵)。

この暗号化鍵は、公開鍵暗号の標準である RSA によって暗号化されたのち EMD に記録される。この「暗号化鍵の暗号化」では、以下の 3 種類の鍵(1024～2048 ビット)を使用する。

オーナー鍵(図 4 の銀色の鍵)	ボリュームの所有者が持つ鍵である。暗号化ボリュームの所有者が持つ鍵であり、1 つだけ作成される。オーナー鍵を使うことで、ボリュームへのアクセスだけでなく、ボリュームの削除などすべての管理権限を有する。また、次に説明するユーザ鍵の作成も可能になる。
ユーザ鍵(図 4 の青色の鍵)	暗号化ボリュームにアクセスしたいユーザが持つ鍵である。ボリュームのマウントやアンマウント、ボリュームへのアクセスは可能だが、ボリュームの削除といった権限は持たない。ユーザ鍵は複数個作成できる。
リカバリ鍵(図 4 の緑色の鍵)	オーナー鍵の破損・紛失時のために、オーナー変更の権限を持つ。リカバリ鍵は複数個作成できる。

EVFS では、これら 3 種類の鍵を用途に応じて使い分ける。例えばシステム管理者は暗号化ボリュームを作成してオーナー鍵を管理する一方で、同ボリュームを利用するユーザやアプリケーションに対してユーザ鍵を発行する、といった具合だ。またオーナー

鍵ファイルが破損したり紛失したりしたときに備えて、暗号化ボリュームの所有者を切り替えできるリカバリ鍵をバックアップ保存しておく。

EVFS の使い方

EVFS はどのような局面で威力を発揮するだろうか。EVFS の応用例を考えてみたい。

まずは、SAN 環境でのデータ暗号化である。SAN を利用してストレージが統合化されている環境であっても、ストレージ上のデータは必ずしも共有されるとは限らない。逆に、ほかのユーザやアプリケーションから見られたくないデータを SAN 上に置きたいというニーズも多いだろう。そこで EVFS によりデータの暗号化を行うことで、そうしたデータも安心して SAN 上で保管することが可能になる。

また、同様に SAN 上に暗号化データを置くことで、特定のユーザやアプリケーション間でのみ安全にデータを共有することが可能になる。つまり、それらのデータ利用者に対して EVFS のユーザ鍵を発行することで、ユーザ鍵を持つものだけが SAN 上のデータを復号化可能になる仕組みだ。

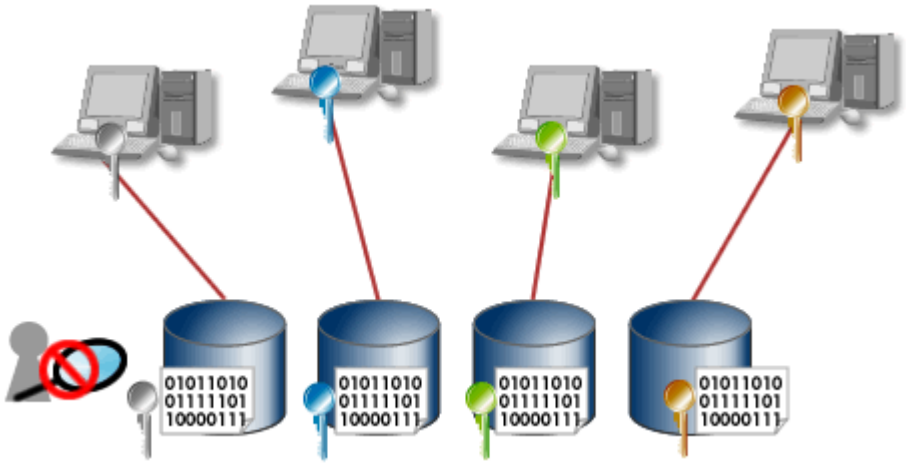


図 5：SAN 上のデータの暗号化

EVFS は、バックアップ・ソリューションの安全性向上にも効果がある。バックアップされるデータが暗号化されるため、バックアップ作業やバックアップ・メディアの管理という面でのセキュリティ確保に神経をとがらせる必要が少なくなる。運用スタッフの人件費などのコストを抑えつつ、セキュリティを高めることができるわけだ。

さらに、仮想化環境での EVFS の利用も考えられる。仮想化環境ではホストとなるシステムがすべてのゲストに対するリソース(ディスク領域など)を提供する。よって、ホストからはゲストのデータが簡単に見えてしまう。そこで EVFS によりゲスト上でデータ暗号化を実施すれば、ホストからもゲストのデータへのアクセスが不可能になる。

つづいて後編では、EVFS の実際の使い方を説明する。

暗号化セキュリティで重要情報を保護・後編

2006 年 10 月　テクニカルライター 吉川和巳

EVFS では暗号化機能を管理するためのコマンド群を用意している。これらのコマンドを利用することで、EVFS サブシステムの管理をはじめ、暗号化鍵の作成や管理、暗号化ボリュームの作成や削除、そして暗号化ファイルシステムの管理を実施することが可能だ。ここでは、EVFS の暗号化機能を利用するまでの操作手順を紹介し、暗号化ボリュームの実際の使い勝手をレポートしたい。

前編で紹介したような強力かつ柔軟な暗号化機能を提供しつつも、コマンドレベルでの操作はとても直感的かつ簡単で、通常の論理ボリュームやファイルシステムとほとんど変わらない使い勝手を提供する。つまり EVFS は、「セキュリティと使いやすさの両立を実現するテクノロジー」と言えるだろう。

EVFS の管理コマンド

前編でも説明したとおり、EVFS では暗号化機能を管理するためのツール群を用意している。以下では、この管理コマンドの実行例を紹介しながら、EVFS の利用の実際を解説したい。 EVFS では、以下の 4 種類のコマンドを提供している。

- evfsadm――EVFS サブシステムの管理
- evfspkey――暗号化鍵の管理
- evfsvol――暗号化ボリュームの管理
- evfsfile――暗号化ファイルシステムの管理(2007 年以降に提供予定)

これらのコマンドを利用しながら、以下の図 1 に示す手順で暗号化ボリュームの管理を進めていく。

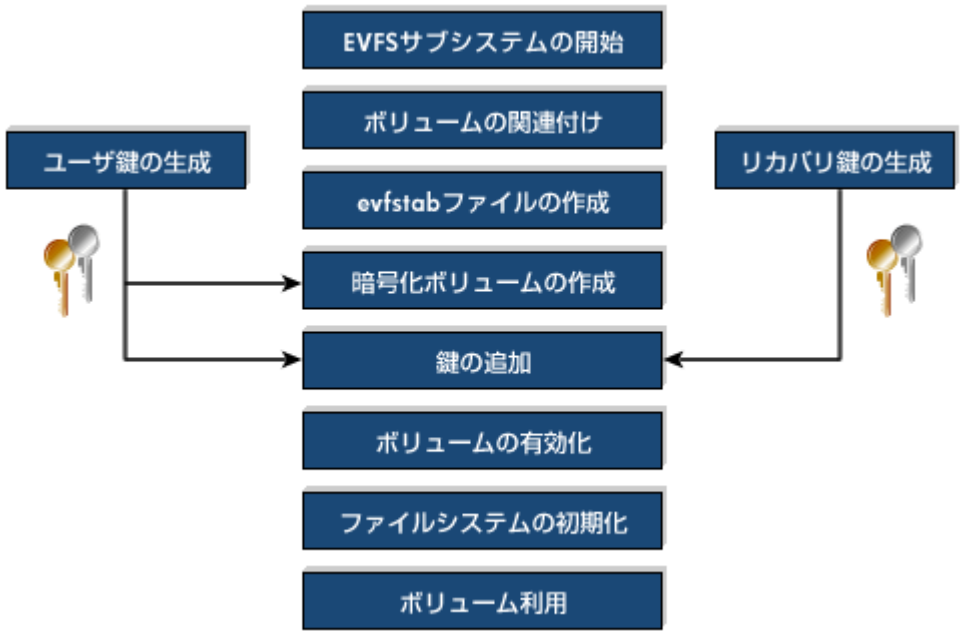


図 1：EVFS 暗号化ボリュームの利用手順

この一連の手順について、順を追って説明していこう。

EVFS サブシステムの開始

EVFS のインストールが完了した HP-UX システムにおいて EVFS の利用を開始するには、まず EVFS サブシステムを起動する。その際に利用するのが、evfsadm コマンドである。同コマンドでは、以下の各オプションが利用可能だ。

＜表：evfsadm コマンドのオプション＞

オプション	機能
start	暗号化サブシステム(EVFS)の開始 (/etc/rc.config.d/evfs 中に開始を記述することが可能)
stop	暗号化サブシステムの停止
map	暗号化ボリュームと論理ボリュームの関連付け
unmap	暗号化ボリュームと論理ボリュームの関連付け解除
stat	ボリュームへのアクセス統計情報
trace	トレースの設定

evfsadm コマンドに start オプションを付けて以下のように実行することで、EVFS サブシステムを起動できる。

```
# evfsadm start
EVFS subsystem started
```

ここで、EVFS サブシステムの動作状況を確認するため、同コマンドに stat オプションを付けて実行する。

```
# evfsadm stat -a
----- EVFS statistics -----
Total Encrypted Volumes: 0
EVFS Subsystem Status: up
Active Encryption Threads: 1
No encrypted volume is currently configured
```

このように、初期状態では暗号化ボリュームが作成されていない状態だ。

次に暗号化ボリュームとして利用する論理ボリュームのマッピングを行う。マップすることによって論理ボリュームと関連付けされた EVFS 仮想デバイスが作成される。

```
# evfsadm map /dev/vg01/vol2
Volume "/dev/vg01/vol2" has been successfully mapped to encrypted volume "/dev/evfs/vg01/evol2"
```

以後、論理ボリュームを暗号化ボリュームとしてアクセスする場合には、この EVFS 仮想デバイスを利用することになる。

暗号化鍵の作成

さて、暗号化ボリュームを作成するには、あらかじめ暗号化鍵を用意しておく必要がある。そこで evfspkey コマンドを用いて、前編で紹介した 3 種類の鍵のうち、オーナー鍵とリカバリ鍵を作成する。

＜表：evfspkey コマンドのオプション＞

オプション	機能
keygen	ユーザ鍵(公開・秘密鍵ペア)およびリカバリ鍵の作成 (作成時にパス・フレーズも同時に決定する)
passgen	パスフレーズ・ファイルの作成
delete	鍵の削除
lookup	鍵の検索

まずは、root アカウントにて「root.key」という名称のオーナー鍵を以下の手順で作成する。

```
# evfspkey keygen -k key
Enter passphrase:
Re-enter passphrase:
Public/Private key pair "root.key" has been successfully generated
```

このように、鍵の作成時には上記例のようにパスフレーズ(鍵を保護するためのパスワード)の入力が求められる。以上の操作によって、/etc/evfs/pkey/root フォルダ以下には、公開鍵と秘密鍵のペアが作成される。このペアがオーナー鍵を構成する仕組みである。

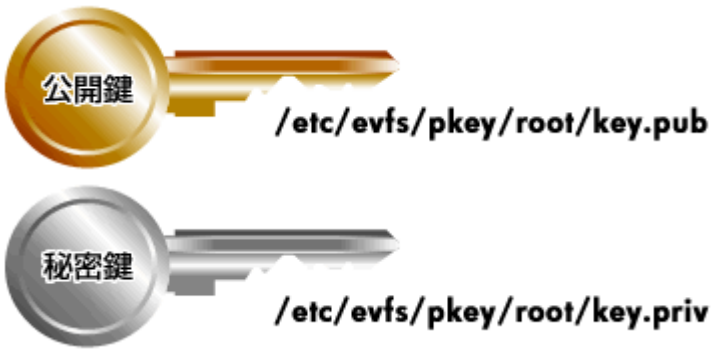


図 2：作成される公開鍵と秘密鍵のペア

つづいて、root 以外のアカウント(この例では evfs アカウントを使用)にて、リカバリ鍵を以下の手順で作成する。

```
# evfspkey keygen -r -k key
Enter recovery passphrase:
Re-enter recovery passphrase:
Public/Private key pair "evfs.key" has been successfully generated
```

これにより、「evfs.key」というリカバリ鍵が作成される。そして最後に、ユーザ鍵「root.subkey」を作成する。

```
# evfspkey keygen -k subkey
Enter passphrase:
Re-enter passphrase:
Public/Private key pair "root.subkey" has been successfully generated
```

以上で、オーナー鍵 root.key、リカバリ鍵 evfs.key、ユーザ鍵 root.subkey という 3 種類の鍵が用意できた。

後半では、これらの鍵を用いて暗号化ボリュームを作成する手順を紹介する。

暗号化ボリュームの作成

つづいては、暗号化ボリュームを作成する。そのために用いるのは、evfsvol コマンドである。

＜表：evfsvol コマンドのオプション＞

オプション	機能
create/destroy	暗号化ボリュームの作成および削除 (作成時、所有者となるユーザと鍵を指定する)
enable/disable	暗号化ボリュームの利用開始・停止
add/delete	利用を許可するユーザ鍵の追加/削除、非常時のリカバリ鍵の追加/削除
assign	所有者の変更(リカバリ鍵を設定してある必要がある)
check	ボリュームのサニティチェックおよび修復
export/import	ボリュームのエクスポート、インポート
scan	暗号化ボリュームの検出

まずは、同コマンドの create オプションを用いて暗号化ボリュームを作成する。ここでは、作成時に指定するオーナー鍵として、先ほど用意した「key」を指定し、作成する論理ボリューム名として「evol2」を指定する。また、同鍵へのアクセスに必要なパスフレーズも合わせて入力する。

```
# evfsvol create -k key /dev/evfs/vg01/evol2
Enter owner passphrase:
Encrypted volume "/dev/evfs/vg01/evol2" has been successfully created
```

これで暗号化ボリューム evol2 が作成された。evfsvol コマンドと display オプションを用いることにより、作成したボリュームの情報を確認することができる。

```
# evfsvol display /dev/evfs/vg01/evol2
Encrypted Volume Name: /dev/evfs/vg01/evol2
Mapped Volume Name: /dev/vg01/vol2
Encrypted Volume State: disabled ← ボリュームは停止中
EMD Size (Kbytes): 520 ← メタデータ領域
Max User Envelopes: 1024
Data Encryption Cipher: aes-128-cbc ← 暗号化強度 128 ビット
Digest: sha1
Owner Key ID: root.key ← オーナー鍵
Recovery Agent Key IDs:
Total Recovery Agent Keys: 0
User Key IDs:
Total User Keys: 0
```

つづいて、以下の手順により、作成したボリューム evol2 にリカバリ鍵を登録する。ここではオーナー鍵のパスフレーズの入力が求められる。

```
# evfsvol add -r -k key /dev/evfs/vg01/evol2
Enter owner passphrase:
Key "evfs.key" has been successfully added to encrypted volume "/dev/evfs/vg01/evol2"
```

また同様にして、ボリューム evol2 にユーザ鍵を登録する。

```
# evfsvol add -k subkey /dev/evfs/vg01/evol2 ← ユーザ鍵の登録
Enter owner passphrase:
Key "root.subkey" has been successfully added to encrypted volume "/dev/evfs/vg01/evol2"
```

以上で、暗号化ボリュームの作成手順が完了したことになる。

EVFS の機能を試す

最後に、この暗号化ボリューム上にファイルシステムを作成し、その機能を実際にテストする手順を説明する。まずは、暗号化ボリュームの有効化を以下のコマンドで行う。

```
# evfsvol enable -k key /dev/evfs/vg01/evol2
Enter user passphrase:
Encrypted volume "/dev/evfs/vg01/evol2" has been successfully enabled
```

つづいて、このボリューム evol2 上に、通常の手順で VxFS ファイルシステムを作成し、マウントする。

```
# newfs -F vxfs /dev/evfs/vg01/revol2
# mount /dev/evfs/vg01/evol2 /mnt2
```

これで暗号化ボリューム上に作成されたファイルシステムを/mnt2 フォルダにて利用する準備が整ったことになる。ここで、テスト用のメール文書 mail.txt を用意し、別途作成した通常ボリューム上の/mnt1 フォルダと、暗号化された/mnt2 フォルダの両方にコピーする。

```
# cp mail.txt /mnt1/mail.txt
# cp mail.txt /mnt2/mail.txt
# diff -c /mnt1/mail.txt /mnt2/mail.txt
No differences encountered
```

diff コマンドの実行結果から分かるように、表面上この 2 つのファイルに内容的な違いはない。しかし、両者をいったんアンマウントし、ボリューム内容を直接ダンプ出力すると、通常ボリュームの方はファイル内容を簡単に読めてしまうのがわかる。

```
# strings /dev/vg01/vol1 | grep -iE 'confidential'
CONFIDENTIAL
This email and any files transmitted with it are confidential and
＜以下略＞
```

これに対し、暗号化ボリュームではファイル内容が暗号化されているため、同様の操作を行っても内容が漏えいすることはない。

```
# strings /dev/vg01/vol2 | grep -iE 'confidential'
#
```

つまり、たとえ root アカウントを有する管理者であっても、オーナー鍵またはユーザ鍵を持ち、かつパスフレーズを知るものでなければ、暗号化ボリュームをマウントしてその内容にアクセスできないのである。

以上、今回は暗号化ボリューム＆ファイルシステム「EVFS」の特徴を概観した。EVFS によるデータの暗号化によって、これまで限られたサーバやストレージ上にしか置けなかった機密性の高いデータであっても、SAN や仮想化環境といったあらゆる場所に保管できるようになる。つまり EVFS は、「セキュリティと使いやすさの両立を実現するテクノロジー」と言えるだろう。

Oracle/Solaris 移行の現場ノウハウ

Oracle/Solaris 移行の現場ノウハウ・前編

2006 年 12 月　テクニカルライター 吉川和巳

精鋭の IT スペシャリスト集団として定評ある SIer「テックファーム」では、ある基幹業務システムの Oracle データベースについて、Solaris から HP-UX への移行を実施した。同社の CTO である小林氏は、今回の移行の理由として、インテル®Itanium®プロセッサ搭載の Integrity サーバのパフォーマンスの高さ、そして 64 ビット OS としての HP-UX の成熟度を挙げる。「いまや UltraSPARC IV+は Itanium に大きく性能差をつけられています。また HP-UX はカーネルからライブラリまで OS 全体が完

全に 64 ビット化されていますが、Solaris ではほとんどのバイナリが 32 ビット版。ライブラリの管理がきわめて煩雑です」(小林氏)。Solaris→HP-UX 移行の現場からのレポートをお届けする。

Oracle サーバの Solaris→HP-UX 移行

2004 年 11 月の特集記事「Solaris 技術者のための『HP-UX への移行のすすめ』」では、テックファーム株式会社の CTO、小林正興氏へのインタビューを通じて、Solaris から HP-UX へのシステム移行の「勘どころ」をレポートした。今回同社では、実際に全国規模で利用されている基幹業務アプリケーションの Oracle データベース・サーバを対象に、Solaris から HP-UX への移行を実施した。このテックファームによる移行事例をベースに、「Solaris→HP-UX 移行の現場のノウハウ」とはどのようなものかを紹介したい。

まずは、今回の移行の対象となったシステム概要を説明しよう。このシステムは、人材派遣ビジネスを手がける企業が全国規模で運用している基幹業務システムである(図 1)。

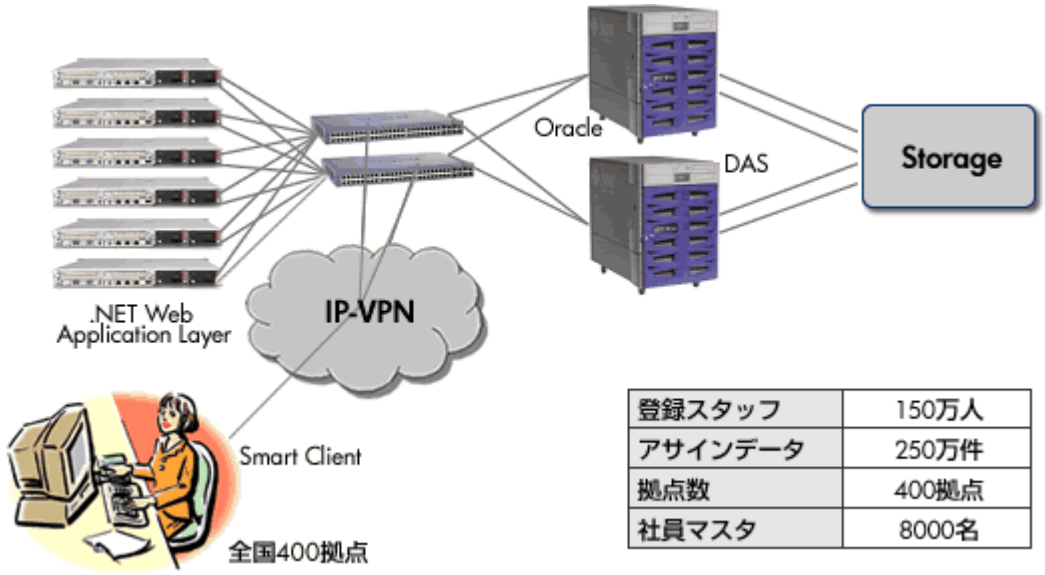


図 1：移行対象システムの概要

このシステムでは、150 万人に達する派遣スタッフの管理や、個々の業務請負案件に対してスタッフを割り当てるアサインデータ(250 万件)の管理などを実現する。図 1 に示すとおり、フロント・エンドでは Windows サーバ上の.NET アプリケーションが動作し、バックエンドでは Solaris サーバ(Sun FireV240)上の Oracle 8i データベースが動作するという構成だ。このシステムに対し、全国 400 拠点のクライアント・アプリケーションから IP-VPN を通じたアクセスを行う。またクラスタウェアとしてベリタスの Veritas Storage Foundation を使用し、アクティブ/スタンバイ型の HA クラスタを構成し、可用性を高めている。

この旧システムはすでに稼働後 4〜5 年を経過している。その間にサーバ・ハードウェアの性能や容量はひと桁違うほどの進化を遂げ、Oracle データベースも Oracle 10g へとバージョンアップした。サービス面や運用面での改善、TCO の抑制、そしてサーバ集約といった観点からも、最新のハードウェアとソフトウェアへ移行する最適な時期を迎えていると言えるだろう。そこで今回、テックファームでは、HP-UX および Integrity サーバを新しいプラットフォームとする新システムへの刷新を実施した。

Itanium と SPARC、それぞれの将来性

では、Solaris ベースの旧システムの移行先として、テックファームはなぜ HP-UX を選択したのだろうか。同社の小林氏は、その理由のひとつとして、ハードウェア・プラットフォームである Integrity サーバのパフォーマンスの高さを指摘する。「Oracle データベースの運用に適したミッドレンジ～ハイエンド・サーバについて、Sun は自社の UltraSPARC IV プロセッサと AMD Opteron プロセッサの 2 本立てでプラットフォームを展開しています。一方、HPE はインテル®Itanium®プロセッサ(以下、Itanium)を搭載した Integrity サーバを中心とした展開です。この両者を比較すると、例えば Sun の主力プロセッサである UltraSPARC IV+は Itanium に大きく性能差をつけられています。Opteron は性能が高いものの、やはり Itanium には及ばないのが現状です」(小林氏)。

また小林氏は、SPARC アーキテクチャそのものの将来性に疑問を呈している。「現在の SPARC アーキテクチャは仕様がオープンソース化された『OpenSPARC』という形態で管理されています。このオープンソース化によってコミュニティを中心とした多様化が期待できる一方で、Linux や Java と同様に、コミュニティ主導の開発プロセスによる開発速度の低下が懸念されます」(小林氏)。さらに同氏は、Itanium 対応ソフトウェアの普及を支援する業界団体 Itanium Solutions Alliance が公開する資料(表 1)を引き合いに、SPARC と Itanium の間で広がりつつあるパフォーマンスやシステムベンダー数の格差を強調する。

表 1：Itanium と SPARC のアーキテクチャ比較

機能		**Itanium アーキテクチャ**	**SPARC アーキテクチャ**
SMP スケーラビリティ	サーバに搭載可能なプロセッサ数 *1	512	128
	アドレス可能メモリ	1,024TB	1,024TB
	グローバル共有メモリ	60TB	576GB
パフォーマンス	SPECfp2000（単一プロセッサ）	2712	1353
	SPECint2000（単一プロセッサ）	1590	845
仮想化		ハードウェアおよびソフトウェアでのパーティショニング	ソフトウェア上でのパーティショニングのみ
可用性		99.99999%*2 複数ベンダーがミッションクリティカルな可用性をサポート	ミッションクリティカルな可用性をサポート
システムベンダー		75	4
オペレーティング・システム（OS）		10,Windows,Linux,UNIX,OpenVMS など幅広い OS で動作	Solaris と Linux。ただし 90%以上は Solaris が提供

対応アプリケーション	6,000 以上および急速に増加中。（および X86 互換アプリケーション*3 にも対応）	多数のアプリケーションがあるが、Windows あるいは x86 アプリケーションはサポートしていない
TCO	RISC ベースのインストールと比較して、Itanium ソリューションでは TCO を 50％まで削減した顧客事例があります*4	

*1:単一 OS イメージをサポートできる機能
*2:現時点で民生品が達成した業界で最も高い可用性
*3:Itanium ベースサーバは、統合 IA-32 Execution Layer を利用して、x86 互換アプリケーションを動作させることができます。
*4:多数の顧客事例に基づく。
注:すべてのシステム・ベンダーの中で最も高い数値を表記しています。特定のシステムに関する特定の数値については各社のベンダーにお問い合わせください。
出典:「Itanium® ベースソリューションと Sun's SPARC® アーキテクチャを比較して」Itanium Solutions Alliance

64 ビット対応で差がつく HP-UX と Solaris

移行先として HP-UX を選択するもうひとつの理由として、小林氏は OS プラットフォームとしての HP-UX と Solaris に多くの類似性が見られることを挙げている。

いずれも System V Unix の系譜に属する UNIX OS であり、基本的なコマンド体系やディレクトリ構成には共通点が多い。とりわけ今回のような基幹業務システムの移行先としては、例えば Linux などに比べてミッションクリティカル環境で豊富な実績を持つ HP-UX が適していると言えるだろう。また、後述するような移行ノウハウを押さえておけば、Solaris 経験者も比較的スムーズに HP-UX になじむことができる。

さらに同氏は、HP-UX には Solaris にはない優位性があると指摘する。それは「64 ビット対応の成熟度」だ。「HP-UX は、カーネルからライブラリまで OS 全体が完全に 64 ビット化されており、安定して動作しています。一方の Solaris は、SPARC 搭載サーバでは安定しているものの、Opteron 搭載の x64 サーバでは初期版における RAID-1 構成の不具合やブート障害があるなど、自社製品でも安定感を欠いています。また Solaris のカーネル部分は 64 ビットながらも、OS を構成するほとんどのバイナリは 32 ビット版で提供されるため、ライブラリの管理がきわめて煩雑です」(小林氏)。データベース・サーバやアプリケーション・サーバの 64 ビット化が一般的になりつつある現在、この違いは大きな意味を持つ。

つづいて後半では、この基幹業務システムの移行の詳細について紹介する。

クラスタウェア Serviceguard の実力

今回の移行では、データベース・サーバを Solaris 8＋Oracle 8i から HP-UX 11i v2＋Oracle10g へと移設する作業を中心に実施された。ハードウェアについては、これまで利用されてきた Sun Fire V240 から、Itanium 2 搭載の Integrity サーバ rx2620 へと変更されている。アプリケーション・コードの変更は実施していない。また HA クラスタを構成するクラスタウェアとして、従来の Veritas Storage Foundation(以下、VSF)から Serviceguard へと変更した。
表 2：今回の移行内容

移行前	移行後

OS	Sun Solaris 8	HP-UX 11iv2
DB	Oracle8i	Oracle10g
クラスタ	Veritas Storage Foundation	Serviceguard
機種	Sun Fire V240	Integrity サーバ rx2620
CPU	UltraSPARC® IIIi 1.5GHz ×2	Itanium2 1.6GHz/6MB ×2
Memory	4GB	8GB
HDD	72GB ×2	72GB ×2
外部ストレージ	Sun StorEdge3320	EVA5000

HP-UX では、クラスタウェアとしてベリタスの VERITAS Cluster Server(以下、VCS)と Serviceguard のいずれを運用することも可能だ。この 2 種類のクラスタウェアについて、小林氏はそれぞれ次のように評する。「VCS は、Solaris 市場で大きなシェアを占める製品であり、リソース別のフェイルオーバーポリシーを GUI 上できめ細かく設定できる点が特徴です。一方の Serviceguard は、世界 10 万本以上の出荷実績があるもっともポピュラーなクラスタウェアであり、金融系でのミッションクリティカル環境における実績も多い。また VCS に比べてデバイス依存性が低く汎用性が高い点や、過去の実績あるスクリプトが数多く付属し、Oracle をはじめとした主要なデータベース等のクラスタ構成を簡単に組むことができる点を高く評価できます。Serviceguard は、HP-UX のミッションクリティカル・オペレーティング環境に標準バンドルされており、コマンドラインツールおよび GUI ともに充実した印象を受けました。さらに Serviceguard では、Metroclusters や Continentalclusters などの大規模な災害対策クラスタにも拡張が可能な点も注目すべきでしょう」(小林氏)。今回の移行では、基幹業務システムの安定運用を優先し、実績ある Serviceguard が選択された。

このように移行先のハードウェアや OS、ソフトウェアの構成が固まった段階で、テックファームでは以下のような移行計画を作成した。

表 3：移行計画の実例

SPU Infomation	Series No	rx2620	Sun-Fire-V240	シリーズ番号
	Host Name	TMHPA	TMSUNA	システムでホスト名として使用される名前 HPE 環境においてはノード名は 8 文字以内
	Memory Capacity	8GB	4GB	メモリーの容量（MB 単位）

LAN Infomation:1	Traffic Type	データ兼ハートビート	データ兼ハートビート	サブネットの目的を指定します。
	Name of Subnet	TFI Type-A	TFI Type-A	サブネットの IP アドレスマスク
	Name of Interface	lan0	bge0	LAN カードの名前
	Node P Addr	192.168.131.220	192.168.131.216	ノードのホスト IP アドレス
LAN Infomation:2	Traffic Type	ハートビート	ハートビート	サブネットの目的を指定します。
	Name of Subnet	Local Heart Beat	Local Heart Beat	サブネットの IP アドレスマスク
	Name of Interface	lan1	bge1	LAN カードの名前
	Node P Addr	10.0.0.220	10.0.0.216	ノードのホスト IP アドレス
Disk I/O Infomation for Shared Disks:	Bus type	Fiber Channel	SCSI	バスの種類を記入します。
	Slot Number	Slot2	PCI1	HBA を挿したスロットの番号
		Slot3	-	
	Disk Device File	c8t0d3	03t0d0	-
		c9t0d3	-	-
		c10t0d3	-	-
		c11t0d3	-	-

カーネルパラメータとネットワーク設定

もう 1 点、移行作業に先立って検討しておくべきポイントは「カーネルパラメータ」である。HP-UX におけるカーネルパラメータ設定の重要性について、小林氏は次のように述べる。「Solaris ではメモリ増設などに対応してパラメータも自動的に変更されるため、カーネルパラメータを変更せずとも使えることが少なくありません。一方、HP-UX では初期設定においてカーネルパラメータを決定しておくことがきわめて重要です。もっとも本来は、Solaris においても、最適なカーネルパラメータをあらかじめ検討することは大切なことです。ちなみに HP-UX では、バージョンアップを重ねるたびに、OS をリブートせずに値を変更可能な『ダイナミック・カーネルパラメータ』(表 4)も増えつつあります」(小林氏)。

表 4：HP-UX のダイナミック・カーネルパラメータ

HP-UX 11i v1	• shmmax • maxuprc • shmseg • maxfiles_lim • maxtsiz • magmnb • msgmax • maxtsiz_64bit • core_addshemem_read • core_addshemem_write • scsi_max_qdepth • semmsl
HP-UX 11i v1.6	HP-UX 11i v1 でサポートされているパラメータに加え、以下が追加されています。 • maxdsiz • maxdsiz_64bit • maxssiz • maxssiz_64bit • nkthread • nproc • max_thread_proc • ksi_alloc_max • max_acct_file_size • shmmni • secure_sid_scripts
HP-UX 11i v2	HP-UX 11i v1.6 のパラメータに加え、以下が追加されます。 • aio_*(aio_physmem_pct を除く) • dbc_max_pct • dbc_min_pct • fs_symlinks • maxfiles_lim • nfiles • nflocks • alwaysdump • dontdump • dump_compress_on • executable_stack • intr_strobe_ics_pct • ioforw_timeout • ncdnode • pagezero_daemon_enabled • physical_io_buffers • st_ats_enabled • st_fail_overruns • st_large_recs • st_san_safe • vxfs_ifree_timelag

注:赤字は J2EE 環境でのチューニング時によく変更されるもの

ネットワーク設定の違いについて同氏は、こう説明する。「Solaris と HP-UX では、ifconfig や route などの基本コマンドの使い方はほとんど同一ですが、ネットワークの設定ファイルの名前や構成が異なります。もっとも、ネットワーク設定は例えば Linux でもディストリビューションによる違いが大きい部分なので、HP-UX だからといって特別にとまどうようなことはありません。またネットワーク設定を含めて、HP-UX ではほとんどの OS 設定作業を SAM や SMH と呼ばれる管理ツールから行えます。このうち SMH(図 2)は、ProLiant サーバとも共通する管理ツールであり、Windows や Linux もほぼ統一されたインターフェース上で管理することができます。それぞれの OS の長所を生かした運用と、安全で低コストなシステム管理を両立するものです。特にシステムのユーザとなる立場の企業であれば、これらのツールを利用した OS 設定をおすすめします」(小林氏)。

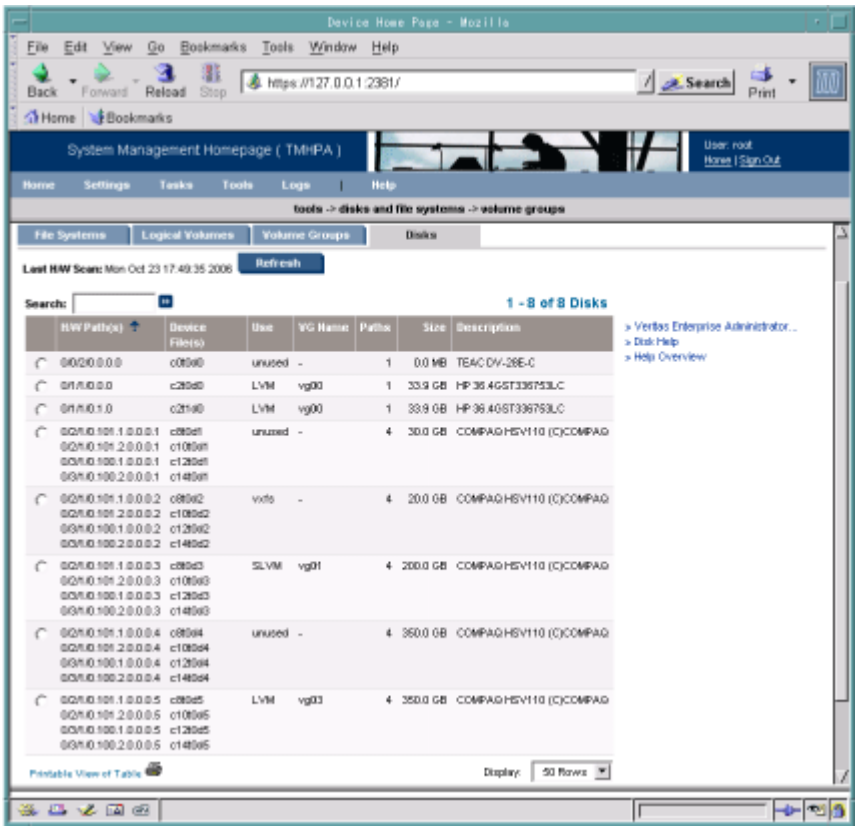


図 2：HP-UX の Web ベース管理ツール SMH

テックファームの小林氏は、今回の HP-UX 移行作業について「今回のように比較的シンプルなクラスタ構成では、OS やミドルウェアの違いを意識する必要はあまりありませんでした。マニュアルに沿って基本的なオペレーションを進めることで移行作業は簡単に終了しました」と振り返る。「最新の Solaris 10 では、Solaris Container や DTrace など派手な新機能が追加されています。これに比べて、HP-UX 11i は細かい使い勝手が着実に向上しており、ツールも充実度を増しています。現在のコストパフォーマンス、性能、将来性を考えたときに、Itanium 2 をベースとする Integrity サーバの魅力はますます大きくなっています」(小林氏)。

つづいて後編では、今回のシステム移行後に実施されたパフォーマンス計測の結果を紹介する。

Oracle/Solaris 移行の現場ノウハウ・後編

2006 年 12 月　テクニカルライター 吉川和巳

Solaris から HP-UX への移行を初めて経験するエンジニアにとっては、例えば ifconfig コマンドなどの Solaris との差異にとまどうこともあるだろう。とはいえ、HP-UX では SAM や SMH などの管理ツールが Solaris よりも充実しており、これらのツールを通じて大半の設定を容易に実施することができる。テックファームでは、Solaris に比較して HP-UX 11i v2 は「細かい使い勝手が着々と向上し、管理ツールも充実しつつある」と評している。要所要所のポイントさえ押さえておけば、Solaris から HP-UX への移行は思ったよりも難しくないはずだ。最新のハードウェアへの移行によって、場合によっては「2 桁台のパフォーマンス向上」も実現しうるのである。

HP-UX 移行のポイント

テックファームでは、今回のような HP-UX 移行におけるポイントを以下のようにまとめている。

- 適切な目標設定をする
- 移行先環境(HP-UX、Serviceguard、Oracle10g R2)の性質を理解する
- 目標設定に従って、ハードウェア構成、ソフトウェア構成の計画を立案し、移行計画のパラメータ・シートに記述する
- 各所の重要な構成要素を確認しながら、移行計画に従って作業を進める
- 構成設計に従って冗長性が確保されていることを確認するテスト項目を作成し、テストを行う

ちなみに今回の移行作業の工数は、表 1 の通りであった。

表 1：移行作業の工数

OS のインストール パッチ割り当て カーネルパラメータ設定 ストレージ設定	1 日
クラスタ設定 Oracle インストール データベース作成	2 日
データ移行（exp/imp）	1 日
アプリケーションの動作確認	1 日

ネットワーク設定のポイント

以下では、ネットワーク設定をはじめ、Serviceguard、Oracle などのインストール作業について、Solaris→HP-UX 移行の個々のポイントを概観しよう。

まずネットワーク設定についてであるが、Solaris と HP-UX にはネットワーク設定ファイルに以下のような違いがある。

表 2：ネットワーク設定の違い

設定項目	**HP-UX**	**Solaris**
IP アドレス	/etc/rc.config.d/netconf	/etc/hosts
ホスト名		/etc/hostname
デフォルトゲートウェイ		/etc/defaultrouter
インターフェース名		/etc/hostname.*** (*** ←インターフェース名)

Hosts	/etc/hosts
DNS 設定	/etc/resolv.conf

また HP-UX の ifconfig コマンドには、Solaris の同コマンドにおける-a オプションに相当するものがない。よって、例えば IP アドレスとネットマスクを確認する場合、HP-UX の ifconfig コマンドを使用して確認するには、あらかじめインターフェース名を調べておかなければならない。以下は、landiag コマンドでインターフェース名を調査した例である。

```
#landiag
Enter command: lan
(省略)
Enter command: display
(省略)
Description = lan0 HP PCI-X 1000Base-T Release B.11.23.0606.01
(省略)
```

ここで得られたインターフェース名「lan0」を ifconfig の引数として実行し、IP アドレスとネットマスクの確認を行う。

```
#ifconfig lan0
lan0: flags=1843<UP,BROADCAST,RUNNING,MULTICAST,CKO>
      inet 192.168.131.221 netmask ffffff00 broadcast 192.168.131.255
```

Serviceguard 設定のポイント

Serviceguard のインストールは、HP-UX のソフトウェア・インストール用コマンド swinstall によって簡単に行える。HA クラスタ構築の手順について、Serviceguard と VCS を比較したのが以下の表 3 だ。

表 3：HA クラスタ構築の手順の違い

No.	項目	HP-UX ー Serviceguard	Solaris ー Veritas
1	共有ディスクの作成	ボリュームグループのデバイスファイル作成	Veritas Volume Manager の「vxdiskadm」コマンドで物理ボリュームの作成
2		物理ボリュームの作成	
3		ボリュームグループの作成	ディスクグループの作成
4		物理ボリュームの追加	
5		論理ボリュームの作成	論理ボリュームの作成
6		ファイルシステムの作成	ファイルシステムの作成
7		ファイルシステムのマウント	ファイルシステムのマウント

8		ボリュームグループのエクスポート	
9		ボリュームグループのインポート	
10	Serviceguard クラスタの作成	クラスタの作成	クラスタの作成
11	Oracle パッケージ作成	Oracle パッケージの作成	Cluster Manager(web コンソール)からリソースの追加(ip、diskgroup、mount、oracle 等)及び依存関係設定。
12		Oracle パッケージを Serviceguard に登録	

また以下の表 4 は、Serviceguard と Solaris の Veritas Storage Foundation のそれぞれについて、設定作業や HA クラスタの管理上で利用するコマンドの対応をまとめたものである。

表 4： Serviceguard と Veritas Storage Foundation のコマンド対応

	HP-UX11iv2 - Serviceguard	Solaris8 - Veritas Storage Foundation
カーネルパラメータ	/usr/sbin/sam	/etc/system
パッチの適用	/usr/sbin/sam	install_cluster
ロードアベレージ確認	/usr/bin/uptime	/bin/uptime
ディスク容量確認	/usr/bin/bdf	/usr/sbin/df
メモリ確認	/usr/bin/vmstat	/usr/bin/vmstat
プロセス確認	/usr/bin/ps	/usr/bin/ps
OS 情報確認	/usr/bin/uname -a	/usr/bin/uname -a
パッチ情報確認	/usr/sbin/swlist	/usr/bin/showrev -p
ネットワークインタフェース確認	/usr/sbin/ifconfig インターフェース名	/sbin/ifconfig -a
Cron の設定	/usr/bin/crontab -e	/usr/bin/crontab -e
GUI 設定ツール	/usr/sbin/sam	/usr/bin/admintool
シャットダウン	/usr/sbin/shutdown	/usr/sbin/shutdown

パッケージのインストールと確認	/usr/sbin/swinstall /usr/sbin/swlist	/usr/sbin/pkgadd -d /usr/sbin/pkginfo
CD-ROM のマウント	/usr/sbin/mount	/usr/sbin/mount
主なログファイル	/var/adm/syslog/syslog.log /var/sam/log/samlog	/var/adm/messages /var/adm/sulog
起動時のログを確認	/usr/sbin/dmesg	/usr/sbin/dmesg
パケット監視	tcpdump 別途インストールが必要	/usr/sbin/snoop
ファイルシステムチェック	/usr/sbin/fsck	/usr/sbin/fsck
クラスタの状態確認	cmviewcl -v	hastatus
クラスタのパッケージの起動	cmrunpkg	なし
クラスタのパッケージの停止	cmhaltpkg	なし
クラスタの起動	cmrunnode	hastart
クラスタの停止	cmhaltnode	hastop

つづいて後半では、Oracle 設定と HA クラスタ構築のポイントについて紹介する。

Oracle 設定と HA クラスタ構築のポイント

HP-UX における Oracle 10g R2 のインストール作業は、HP-UX Developer Edge に掲載されているインストール・ドキュメントの手順に従うことで、特に問題なくインストールできる。

Oracle インストール後の HA クラスタ構築は、以下の手順で実施する。

1. Oracle Database 10g R2 をインストールする(データベースは作成しない)
2. プライマリ・サーバ側でストレージ上に物理ボリューム、ボリュームグループ、論理ボリュームを作成し、これらをマウントする
3. NTP を設定し、冗長構成サーバ間の同期をとる
4. 各ノードにハートビート用 LAN インターフェースの設定を行う
5. プライマリ・ノードから vgexport でマップファイルを作成し、スタンバイ・ノードで vgimport を実行する
6. 各ノードで/etc/lvmrc の AUTO_VG_ACTIVATE を 0 に設定し、同時にマウントしないように設定する
7. プライマリ・ノード側のみボリュームグループ(VG)をアクティベートし、cmquerycl コマンドでクラスタファイルを作成する
8. クラスタファイルを適切に設定し、cmapplyconf によって適用する

9. プライマリ・ノードにて dbca を実行し、データベース作成スクリプトを生成する
10. 上記で生成したデータベース作成スクリプトによってデータベースを作成する
11. VG をプライマリ・ノードからスタンバイ・ノード側に切り替え、Oracle の dump ディレクトリ、初期化ファイル(init.ora)を作成し、spfile、oratab、orapw へのシンボリックリンクを作成する
12. プライマリ・ノード側で Enterprise Cluster Master Toolkit から Oracle 設定をコピーし、haoracle.conf を環境に応じて編集する
13. cmmakepkg でパッケージ設定ファイル、パッケージ制御ファイルを作成し、これを適切に編集する
14. プライマリ・ノード側の設定をすべてスタンバイ・ノード側にもコピーし、cmapplyconf を実行して適用する

移行後のパフォーマンスは大幅に向上

最後にテックファームでは、今回の移行作業による効果を明らかにすべく、移行後にパフォーマンス測定を実施した。このパフォーマンス測定では、同社オリジナルのパフォーマンス測定ツールを利用している。同ツールでは、データベースに一定の負荷をかける検索を 5 つのスレッドで同時に実行しながら、オペレータによる参照・更新処理をシミュレーションする。このツールにより得られた計測結果は以下の通りである。

表 5：移行前システムと移行後システムの性能比較

	Solaris		HP-UX	
	処理回数	Load Average	処理回数	Load Average
開始後 5 分	1.75	7.33	187.5	2.85
10 分	1.50	7.33	179.0	2.81
15 分	1.75	7.29	173.0	2.83
20 分	1.50	7.40	176.3	2.83

ここで「処理回数」の違いに注目すると、およそ 100 倍程度の性能向上が実現していることが分かる。Oracle データベースのバージョンアップによる改善もさることながら、やはり CPU 速度やメモリ容量といったハードウェア性能の大幅な向上による影響が大きいと言えるだろう。

HP-UX への移行は難しくない

テックファームによれば、HP-UX や Serviceguard のインストールに関して、日本ヒューレット・パッカード提供のドキュメントが充実していることもあり、取り立てて大きな問題は起こらなかったという。HP-UX 未経験者であっても、UNIX を扱った経験のあるエンジニアであればインストール作業はスムーズに進められるだろう。

インストール後の設定作業については、上述の ifconfig コマンドをはじめとする Solaris との差異にとまどうこともあり得る。とはいえ、HP-UX では SAM や SMH などの管理ツールが Solaris よりも充実しており、これらのツールを通じて大半の設定を容易に実施することができる。テックファームでは、Solaris に比較して HP-UX 11i v2 は「細かい使い勝手が着々と向上し、管理ツールも充実しつつある」と評している。

HA クラスタの構築についても、HP-UX 上で Serviceguard を利用するためのドキュメントが用意されているため、目立った問題は発生しなかったという。また Solaris と HP-UX とでは、HA クラスタの運用手順に大きな違いはない。今回テックファームではストレージとして StorageWorks EVA5000 を導入し、かつジャーナル・ファイルシステムとして Online JFS を導入した。これにより、ディスク容量の段階的な追加などのスケーラビリティ向上が実現している。

以上、テックファームによる事例に基づいて、Solaris から HP-UX への移行の実際を紹介した。ここで見てきたとおり、要所要所のポイントさえ押さえておけば、Solaris から HP-UX への移行は思ったよりも難しくない。また最新のハードウェアへと移行することで、場合によっては今回の事例のように 2 桁台のパフォーマンス向上も実現しうるのである。

HP-UX
www.hpe.com/jp/hpux

© Copyright 2018 Hewlett Packard Enterprise Development LP.

本書の内容は、将来予告なく変更されることがあります。日本ヒューレット・パッカード製品およびサービスに対する保証については、当該製品およびサービスの保証規定書に記載されています。本書のいかなる内容も、新たな保証を追加するものではありません。日本ヒューレット・パッカードは、本書中の技術的あるいは校正上の誤り、脱字に対して、責任を負いかねますのでご了承ください。

Hewlett Packard Enterprise