HP Serviceguard for LinuxクラスターをRed Hat Enterprise Linux 5 Advanced Platform上のRed Hat Cluster Suiteへ移行する方法

2009年5月

エグゼクティブサマリー

このWhite Paperでは、HP Serviceguard for Linux（SGLX）クラスターをRed Hat Cluster Suite（RHCS）が稼動しているクラスターへ移行する手順について説明します。SGLXとRHCSのクラスターの違いについては、メンバーシップ、クォーラム、データの破損を防ぐフェンシング、アプリケーションのフェイルオーバー制御などを説明します。また、既存のSGLXクラスターの構成情報を利用し、同様の機能を備えたRHCSクラスターを短時間で作成する方法についても、順を追って説明します。

SGLXにはあって、類似のものがRHCSにはない機能のいくつかを一覧にまとめました。このような違いの多くは、カスタムスクリプトで管理できます。

このWhite Paperは、Red Hatと共同で作成しました。

はじめに

Red Hat Cluster Suiteは、いくつかの構成オプションを提供して異なる可用性のニーズに対応し、多様なハードウェア構成をサポートします。計画する際、管理者はビジネスにおける可用性のニーズやクラスターとして選定されたハードウェアを基に、正しい構成を選択しなければなりません。このWhite Paperでは、SGLXユーザーにとって重要な意味を持つ構成の2つの側面に重点をおいています。1つは、クラスターから削除されたノードでデータ破損が発生しないことを保証するフェンシングであり、もう1つはクラスターメンバーシップです。

Red Hat Cluster Suiteが提供するフェンシング方式は、大きく分けて2つあります。パワーフェンシングとストレージフェンシングです。パワーフェンシングでは、ネットワークアドレスの割り当てが可能なパワーディストリビューションユニット（PDU）またはHP Integrated Lights-Out（iLO）といった統合管理カードなど、外部のパワーフェンシング機器を使って電源をリセットすることができます。ストレージフェンシングでも、共有ストレージへのアクセスを2つの異なる方法で制限できます。1つは、サーバーに接続したスイッチからストレージエリアネットワーク（SAN）のポートを制御する方法で、もう1つはSCSI-3 Persistent Reservationを介してポートを制御する方法です。

このWhite Paperでは、 HP Integrated Lights-Out（iLO）をベースにしたパワーリセットフェンシングと、Persistent Reservationフェンシングの両方について説明します。パワーリセットフェンシングは、長年RHCSで利用されてきました。Persistent Reservation（PR）フェンシングは、現在RHCSでサポートされています。このWhite Paperでは、HP iLOフェンシング方式によるフェンシングとSCSI-3 PRフェンシング方式によるフェンシングという、2つの「使い慣れた」Red Hat Cluster Suite構成について検証し、詳細を説明します。これらの2つのRHCS構成は、SGLXと同等とは言えないまでも、似たレベルで可用性の高い環境を実現します。移行の際は、環境に最も適した方式を選択できます。

RHCSとSGLXは、異なるメンバーシップアルゴリズムを有しています。このWhite Paperでは、RHCSクォーラムディスク（qdisk）を使って、非対称クラスター構成を導入せずに既存のクォーラムの方式を強化する方法について説明します。このほか、「自動リセット方式」を使用してRed Hatのメンバーシップ機能を拡張することもできます。これはSGLXクラスターにもあるように、障害の発生したノードをクラスターへ自動的に復帰させることで実現します。

このWhite Paperの第1部では、SGLXクラスターから移行するときに使用する、2つの「使い慣れた」Red Hat Cluster Suite構成を定義します。1つめの「使い慣れた」RHCS構成では、「自動リセットメカニズム」を持った「Persistent Reservation」フェンシングを使用します。これにより、データの破損を防止しながら、SGLXクラスターと同様に障害の発生したノードを自動的にクラスターへ復帰できるようになります。2つめの「使い慣れた」RHCS構成では、HP iLOパワーフェンシングメソッドを使用します。いずれの「使い慣れた」構成でも、さまざまな障害シナリオを扱えるように、既存のクォーラムを強化するqdisk構成が必要です。

SGLXユーザーは、パッケージの概念や、ツールキットを使ってパッケージ制御スクリプトの開発を簡易化する方法について熟知していることでしょう。Red Hat Cluster Suiteでは、同様の概念にある「リソース」、「サービス」、そして「フェイルオーバードメイン」を用いてアプリケーションのフェイルオーバーを制御します。このWhite Paperでは、これらのRHCS機能がどの部分でSGLXパッケージとツールキットの構造と類似しているのか、まとめて説明します。

このWhite Paperの第2部では、移行プロセスについて説明します。SGLXクラスター構成情報を収集し、その情報に基づいてRHCSクラスターを作成する方法については、順を追って説明します。説明では、サンプルアプリケーションを使用します。(OracleやApacheなど)特殊なツールキットで構成されたSGLXクラスターの移行については、別途、他のWhite Paperで説明します。

このWhite Paperでは、Red Hat Enterprise Linux Advanced Platform 5.2に含まれるRed Hat Cluster Suiteを基に説明します。このWhite Paper内の情報は、以降のRed Hat Enterprise Linux 5についても大幅に変更されることはありません。

対象読者

このWhite Paperは、RHEL5上で実行されるHP SGLXを使用していてRHEL5.2以降のRed Hat Cluster Suiteへ移行したいと考えているユーザーを対象としています。

また、読者がHP SGLXとRed Hat Cluster Suiteについて理解していることを前提としています。Red Hatのドキュメントにある詳細については、すべてをこのWhite Paperで説明しているわけではありません。各ソリューションについて詳しくは、HPのWebサイト http://h50146.www5.hp.com/products/servers/proliant/svglinux/index.html
または、http://www.hp.com/go/sglx (英語)とRedHat社のWebサイト
http://www.redhat.com/docs/manuals/csgfsを参照してください。

HPとRed Hatの正式ドキュメント以外にも、Red Hat Cluster Suiteのアップストリームプロジェクトが管理するwiki(RedHat社のWebサイトhttp://sources.redhat.com/cluster/wiki)には豊富な情報が掲載されています。ただし、アップストリームにある機能は完全に正式リリースへ実装されるわけではないので、wikiの情報を参照する際はその点にご留意ください。

Red Hat Cluster Suiteとフェンシング

どの高可用性クラスターの中核部分にも、クラスターに関連するノードを示す、メンバーシップという概念があります。クラスターソフトウェアは、さまざまな障害シナリオに基づいてメンバーシップを調整するアルゴリズムを持っています。一般的に、クラスターソフトウェアには、どのノードのセットをクラスターとして継続的に定義するかを定めた「クォーラム」という概念があります。データを保護するために、クォーラムを持たないノードはクラスターメンバーシップから削除されます。クォーラムを持たないことで削除されたノードは、共有リソースへアクセスできないようにする必要があります。フェンシングは、このようなアクセス制限を指すときに使う用語です。

クォーラムのルールとフェンシング

Red Hat Cluster Suiteでは、クラスター内の定義済みノードにおける単純な多数決でクォーラムを決定します。再編成を成功させるには、すべての可能な投票（ボート）の過半数が必要です。「ローリングメンバーシップ」はありません。オペレーターがノードを停止させた場合でも、そのノードのボートは「可能性のある」ボートの1つとして残ります。RHCSでは、各クラスターノードにいくつかのボートが割り当てられ、メンバーである間はクラスターに貢献します。可能性のあるボートすべてにおいてクラスターが過半数を獲得すると、クォーラムを持つ（またはクォーラムに達した）ことになり、それ以外はクォーラムを持たないことになります。

SGLXの場合、クラスターの最終状態（「ローリングメンバーシップ」）のメンバーシップボートを基にそれを決定します。たとえば、当初クラスターに6つのノードがあり、管理者が1つを削除した場合、障害に関するボーティングは、5つのノードについて実行されます。メンバーシップの仕組みの違いは、小さなクラスターで1つ以上のノードに同時に障害が発生したとき、またはスプリットサイト（耐障害性）クラスターで最も明確になります。このほか、SGLXはロックLUNまたはクォーラムサービスを使用して「タイブレークする」こともできます。

Serviceguardは、「自己リセット」方式で障害ノードが共有ストレージに書き込むことを防止します。「デッドマンドライバー」は、「ハング」状態にあるノードをリセットする重要なコンポーネントです。Red Hat Cluster Suiteは、さまざまなフェンシング方式をサポートします。

SGLXとRHCSのいずれにおいても、「サイズが不均衡なパーティション」（言い換えれば、ネットワーク障害が発生したことで、異なるメンバー数でパーティションが作成されること）があると、多数のボートを備えたパーティションにより、クォーラムが成立し、新しいクラスターが構築されることになります。RHCSの場合、障害ノード（またはクォーラムを失ったパーティション）はクォーラムに達したパーティションによってフェンシング、つまりクラスターから削除されます。一方のSGLXでは、クォーラムを失ったパーティションのすべてのノードは「自己リセット」を行います。

SGLXでは、クォーラムサーバーまたはロックLUNをタイブレーカーとして使用し、パーティションでクォーラムを成立してクラスターを構成することができます。同様にRHCSでも、クォーラムディスク（qdisk）をタイブレーカーとして使用し、非対称構成を扱うこともできます。

障害検知時間

RHCSクラスターのtotem token（`/etc/cluster/cluster.conf`にある*token*パラメーター）は、障害検知時間を示します。これは、ノードに障害が発生してからクラスターソフトウェアがそれを検知するまでの時間です。HP ServiceguardバージョンA.11.18以前では*NODE_TIMEOUT*が障害検知時間を、ServiceguardバージョンA.11.19では*member_timeout*が障害検知時間を表します。totem tokenパラメーターの設定は、Red Hatが提供するガイドラインで規定されています。たとえば、qdisk（クォーラムディスク）を使用した場合、totem tokenのタイムアウトはqdiskメンバーシップのタイムアウトの少なくとも2倍に設定する必要があります。別のガイドラインとして、クラスターの通信とクライアントトラフィックの両方が同じネットワークを使用している場合、間違ったクラスターの再編成が行われないよう、totem tokenのタイムアウトは長く設定します。

クォーラムの強化におけるqdiskの使用

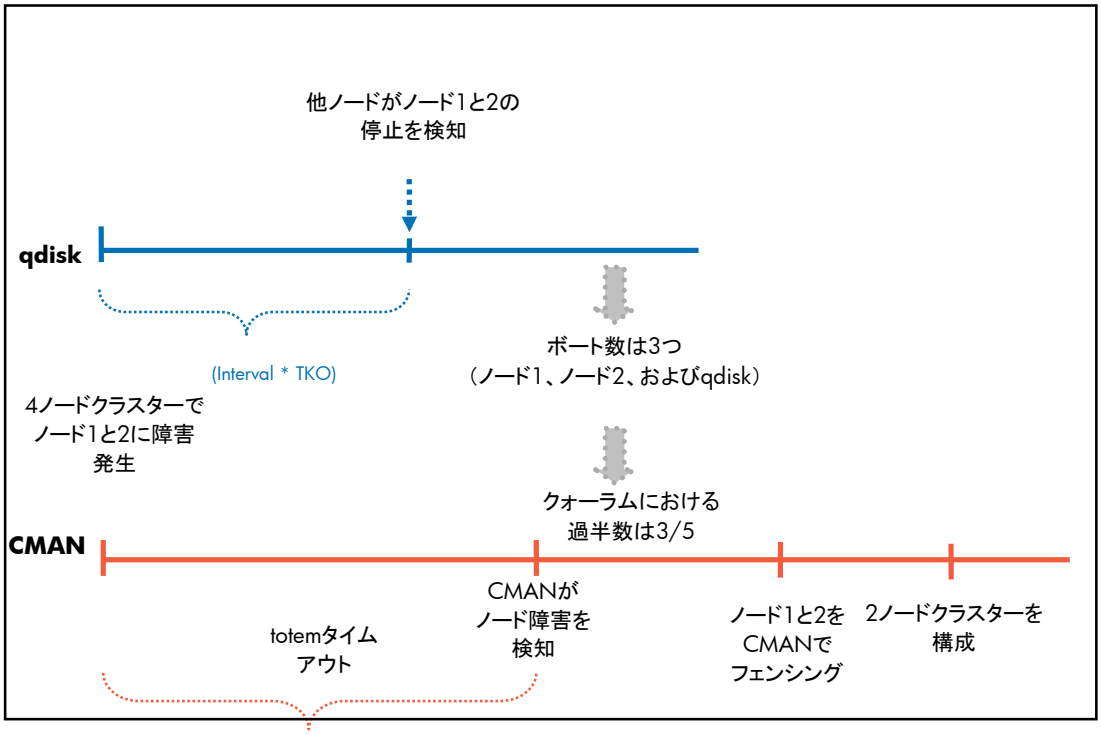
クォーラムは、過半数という意味です。クォーラムを作成するにはクラスター内の過半数のノードが必要で、デフォルトでノードに1つ与えられたボートを使用して決定します。稼動するには、4つのノードクラスターであれば3つのアクティブノードが、6つのノードクラスターであれば4つのノードが必要となります。ただし、4つのノードクラスターの場合、2つのノードを失うとクォーラムを失います。

RHCSクラスターであれば、正確に半分またはそれ以上のノードが同時に停止した場合にクォーラムを失います。クォーラムを失うと、フェンシング、GFSの復旧、クラスターサービスの起動などのクラスター操作が無効になる、という一時停止状態に陥ります。このような場合、オペレーターが介入して、停止していないノードを手動でリセットし、クラスターを再起動する必要があります。大規模クラスターで半分以上のノードを失う可能性は、小規模クラスターよりも極めて低くなります。たとえば10ノードクラスターの場合、(クォーラムをなくすだけの)5ノードを失う可能性は、4ノードクラスターで2ノードを失うよりも低いでしょう。SGLXでは、クラスターメンバーの過半数で障害が発生し、残りのノードがクォーラムを失った場合、「自己リセット」を行います。

長年かけて実証されたクォーラム強化のアプローチは、ボートが必要な過半数に達するようにスペアマシンをクラスターに追加するというものです。ただし、RHCSクラスターでは共有ストレージ方式でクォーラムディスク(qdisk)を参照すればクォーラムのカウントを簡単に強化できるため、スペアマシンを追加してクォーラムのカウントを強化する方法はかえって高価となります。

*qdisk*とは、クラスターノードで共有される10MBの小さなディスクパーティションのことです。各ノードのqdiskデーモン(*qdiskd*)はヘルスを定期的に評価し、状態やタイムスタンプをディスク内の割り当てられた部分に更新します。その後、各*qdiskd*はqdiskパーティションのそれぞれの領域にあるクラスターのその他のノードの状態を確認します。ノードに障害が発生した場合(または共有ストレージへアクセスできなくなった場合)、qdisk内の領域は所要の頻度で更新されなくなります。これは、クラスター内の他のノードでも検知されます。稼動状況に問題がなければ、すべてのクォーラムのカウントは各ノードのボート数と*qdisk*に割り当てられたボート数の合計となります。ノードに障害が発生した場合、qdiskは自身のボートを稼動しているノードに渡し、クラスターを構成するのに必要な過半数のボートを確保します。ちょうど過半数のメンバーで障害に耐えるには、すべてのメンバーがボートを1つ持っていると仮定して、*qdisk*にはボートを1つ割り当てます。最後のノードが停止するまでクラスターを運用するには、クラスター内の総ノード数よりも1つ少ないボート数をqdiskに割り当てます。つまり、ノードに1つのクォーラムパーティションを追加すれば、クラスター運用を継続させるのに十分なクォーラムを保持できるということです。クラスターのハートビートタイムアウトが発生すると、qdiskはボートを提供し、停止していないノードはクォーラムを獲得して障害ノードをフェンシングし、クラスターを構成します。図1に、4ノードクラスターで2ノードに障害が発生した場合、クォーラムを獲得するために、どのようにqdiskが必要なボートを提供するのかを示します。

図1：過半数のノード障害が発生したときのqdisk



次の*quorumd*構成では、qdiskdが2秒ごとに評価を実行し、5回実行した中でノードが更新されていない場合、ノードが停止したと判断します。

```
<cluster alias="rhcluster" config_version="1" name="rhcluster">
 . . . .
    <totem token="20000"/>
 . . . .
    <quorumd interval="2" label="qdisksgrh" tko="5" votes="1">

</cluster>
```

Red Hatは、CMANメンバーシップタイムアウト（totem tokenタイムアウト）の値を、*qdiskd*メンバーシップタイムアウト値の少なくとも2倍に設定することを推奨しています。したがって、この例ではtotem tokenを20秒に設定しています。

注：上記の方法は、qdiskでどのように（停止している）障害ノードを検知してフェンシングするかを示したものです。同じ方法で、ストレージにアクセスできなくなったノードを検知し、フェンシングすることもできます。RHCSクラスターでは、ストレージにアクセスできなくなったノードは自動的にフェンシングされません。ノードが共有ストレージにアクセスできなくなると、そのノード上のサービスはストレージへのアクセス時に停止し、クラスター内の別ノードにフェイルオーバーされます。このノードは、サービスをホストできなくなっても、クラスター内に参加し続けます。SANの障害が復旧する前に障害ノードへサービスを戻すような行為をした場合、サービスは開始せずに別のノードへ移動させられます。上記で示した（停止している）障害ノードで使用するqdisk構成は、SANから切り離されたノードをクラスターに参加させないよう防止するのに活用できます。

ここまでは、クラスターメンバーがクラスターのハートビートに応答しない障害ノードを、ネットワークのパーティションではなく停止していることが原因であると判断できるようにするqdiskの役割を見てきました。ここからは、クラスターメンバーが

クラスター参加の適合具合を判断し、クラスターから削除されるよう「自己申請」するためのqdiskの拡張機能を見ていきます（自己申請は、一般的に同数のネットワークパーティションで使用されます。詳しくは、後で説明します）。

qdiskへアクセスする前に実行する、1つ以上の異常ノードの発見方法を構成に追加できます。これは、ノードがクラスター参加に「適合する」か「適合しない」かを自ら宣言するために使用する「適合」チェックです。必要最低限のスコアを超えたノードのみが、クォーラムディスク経由で「適合する」と申し立てできます。最低限のスコアよりも低いノードは、クォーラムディスク経由でクラスター参加に「適合しない」と申し立てます。別ノードで実行中の*qdiskd*は、「適合しない」ノードのフェンシングをCluster Managerに要求します。

2ノードだけでなく、2ノード以上のクラスター内の同数のネットワークパーティションでは、一般的に異常ノードの発見方法のメカニズムを持つ*qdisk*を使用します。あるパーティションがクラスター参加に「適合しない」と宣言し、もう1つのパーティションが「適合する」としたままになれば、「適合しない」パーティションがフェンシングされます。この仕組みにより、HP iLOのようなノードごとの電源管理を行うときに発生する可能性のある「フェンシング競争」を防ぐことができます。

2ノード以上あるクラスター内の同数に分割されたパーティションでは、いずれのパーティションもクォーラムを獲得できず、クラスター全体が一時停止の状態になります。異常ノードの発見方法を持つqdiskをタイブレーカーとして使用し、これにより、パーティションの1つはクラスターの構成を続行できます。

注：サイズが均衡のパーティションでタイブレーカーとして使用した異常ノードの発見方法を持つ同じqdiskは、障害ノードを検知して削除することができます。

異常ノードの発見方法を持つqdiskにおける別の利用方法は、非対称構成の取り扱いです。たとえば、4つのノードクラスターがある場合、過半数のパーティションを占める3つのノードはクラスターに対して「すべて」適合しないと判断しておき、少数のパーティションを占める1つのノードでは継続して運用することができます。SGLXにおいて、クラスターメンバーの半数以上で障害が発生すると、残りのノードがシャットダウンします。このほか、異常ノードの発見方法を持つqdiskはパブリックルーターにアクセスできないなどの外的要因から、メンバーが「適合しない」と宣言する事にも使用できます。

次のサンプルは、2ノードクラスターにてネットワークパーティション（クラスター通信ネットワーク障害）時のフェンシング競争を防ぐのに使用する異常ノードの発見方法の定義です。クラスターは、クライアント用とクラスター通信用の別のネットワークを使用するよう設定します。クラスター通信は、スイッチで接続された単一NICを使用するものとします。このサンプルでは、異常ノードの発見方法は`ethtool`を使ってEthernetリンクの状態をチェックします。両ノードのリンクは接続されていないので、クロスオーバー ケーブルでは動作しないことに注意してください。

異常ノードの発見方法を持つqdiskにおけるXML定義は、次のとおりです。

```
<quorumd interval="2" label="qdisksgrh" tko="5" votes="1">
   <heuristic program="/usr/local/sbin/checkpvtlink.sh eth1"
      score="1" interval="2" tko="3"/>
</quorumd>
```

Red Hatでは、totem tokenを、上記のXML定義では10秒に設定されている*qdiskd*メンバーシップタイムアウト値（`quorumd interval * quorumd tko`）より最低2倍の20秒に設定します。スクリプトの`/usr/local/sbin/checkpvtlink.sh`は、次のように実装します。

```
#!/bin/sh

#
# Check for link state
#
ethtool $1 | grep -q "Link detected.*yes"
exit $?
```

注：異常ノードの発見方法によるチェックは、パーティション間の認識できる違いに基づいて行われていることを理解することが重要です。これにより、1つのパーティションのみが必要最小限の異常ノードの発見方法のスコアを獲得でき、もう一方は最小限のスコアを獲得できなくなります。そうしないと、両パーティションは「適合する」と宣言して「フェンシング競争」に突入してしまい、クラスターを停止に追い込む可能性があります。上記サンプルの異常ノードの発見方法がパブリックネットワークを基にしていた場合（アップストリームルーターをpingする）も、両ノードはすべて参照できてしまい、結合は解除されず、「フェンシング競争」に突入してしまいます。

qdiskにおける推奨事項

Red Hatは、半数（またはそれ以上の）メンバーに障害が発生したときの障害処理、同数分割パーティションのタイブレーカー、およびSAN障害の検知を処理するため、qdisk構成を使用してクォーラムを強化することを推奨しています。

Red Hat Cluster Suiteでは、クォーラムルール（すなわちクォーラムは過半数のボートが必要）での例外によって2ノードクラスターを作成でき、クォーラムを作成するためのノードは1つで十分と判断されます。ネットワークパーティションの場合は、クォーラムを持つ各ノードが別ノードをフェンシングしようと競争になります。HP iLOで使用されている（クラスターノード間で電源制御機器を共有しない）ノードごとの電源管理において、ノードが互いをフェンシングする可能性は非常に低いながらも存在し、その結果、クラスター全体を停止してしまうことがあることに注意してください。また、永続的なネットワークの問題は「AがBをフェンシングして、BがAをフェンシングする」ようなフェンシングループを招く可能性があります（フェンスデバイスが引き続きアクセス可能な場合）。明確に定義された異常ノードの発見方法を持つqdiskで設定したクラスターであれば、2ノードクラスターのネットワークパーティション時での「フェンシング競争」や「フェンシングループ」を防ぐことができます。SCSI-3 PRフェンシング手法を使用すると、ノードは同時に互いをフェンシングしなくなります。これは、SCSI-3 PR方式の原子的な性質によるものです。一度削除されたノードは、別のノードを削除できなくなります。ただし、永続的ネットワーク問題時には「フェンシングループ」の可能性が残ります。これは、*qdisk*構成で防ぐことができます。

2ノード以上あるクラスター内の同数に分割されたパーティションでは、いずれのパーティションもクォーラムを獲得できず、クラスター全体が一時停止の状態になります。このような状況では、オペレーターが介入して、パーティションノードを手動でリセットし、クラスターを再起動する必要があります。明確に定義された異常ノードの発見方法を持つqdiskをタイブレーカーとして使用し、これにより、パーティションの1つはクラスターの構成を続行できます。

半数メンバーの障害に耐えるには、「1」のボートを割り当てたqdisk構成が必要です。これは、すべてのメンバーが「1」のボートを持つと想定するものです。半数のノードで同時に障害が発生した場合、qdiskは停止していないノードがクラスターを構成するのに必要な過半数を獲得できるよう、ボートを渡します。異常ノードの発見方法を持つqdisk構成は、同数分割パーティションまたはメンバーの半数における障害に耐えられるよう、SGLX機能をRHCSへ実装するのに使用します。これは、RHCSには同一のものがない（または簡単に使用できない）SGLXオプションを、スクリプトを使用してプログラムで簡単に実装できることを示した例です。

RHCSクラスターでは、ストレージにアクセスできなくなったノードはフェンシングされません。ノードが共有ストレージにアクセスできなくなると、そのノード上のサービスは同時に停止し、クラスター内の別ノードにフェイルオーバーされます。このノードは、サービスをホストできなくなっても、クラスター内に参加し続けます。SANの障害が復旧する前に障害ノードへサービスを戻すような行為をした場合、サービスは開始せずに別のノードへ移動させられます。qdisk構成を使用して、SANから切断されたノードがクラスターに参加することを防ぐことができます。SGLXでは、ディスク監視サービスを使用して、FCリンク障害時にパッケージフェイルオーバーできるようにします。

ネットワーク構成の検討事項

SGLXのハートビート要件は、複数のハートビートネットワークまたは(Linuxボンディングを使用した)単一のHAネットワークを使用する事です。ハートビート通信とクライアント通信の両方とも、同一ネットワーク上または別のネットワーク上に存在できます。推奨構成は、クライアント通信とハートビート通信を異なるHAネットワークで使用することです。ネットワークの「最小」構成であれば、同一ネットワーク上でクライアント通信とハートビート通信の両方を使用することができます。

Red Hatは、(Linuxチャネルボンディングを使用した)NICのボンディングされたペアをクライアントアクセスで、別のNICのボンディングされたペアをクラスターハートビートで使用することを推奨しています。チャネルボンディングは、追加のHA保護を提供し、クラスターの再編成の回数を削減します。ネットワークの信頼性を最大限に高めるには、ネットワークのボンディングで1つのマルチポートNICから複数の接続を受け付けないことです。

注: クラスター通信ではIPv6アドレスを使用するよう設定できますが、現時点ではRed Hatでサポートされていません。

別のネットワーク構成は、ネットワークの「最小」構成です。ここでは、クライアントネットワーク用にボンディングされたNICペアを使用し、同じNICペアをハートビートでも使用します。ネットワークの「最小」構成を採用する前に、次の制限を検証してください。

1. ネットワークトラフィックの増大時にはハートビートが失敗することもあり、その結果、不要なクラスターの再編成が発生することがあります。totem tokenのタイムアウトは、「ハートビートの失敗」を防ぐためにも、高い値に設定する必要があります。これは、ハートビート通信とクライアント通信の両方が同一ネットワークを共有する場合のServiceguardの制限でもあります。

2. ハートビートメッセージは通常、共有鍵で暗号化されますが、インターネットに接続するネットワークからハートビートメッセージを外すセキュリティ上の理由があることも考えられます。

ServiceguardとRed Hat Cluster Suiteのクライアント通信およびハートビート通信では、同じネットワーク構成の設定を引き続き使用します。

ネットワーク構成の検討事項の最後は、Red Hat Cluster Suiteはハートビート通信のデフォルトをローカルサイトのマルチキャストネットワークと同じ設定にする点です。マルチキャストアドレスは、利用環境の要件に応じて変更してください。

HP iLOフェンシングに基づく「使い慣れた」構成

クォーラムを持たないノードはクラスターメンバーシップから削除され、共有リソースへのアクセスが禁じられます。HP ProLiantシステムやIntegriyシステムにおいて、HP iLOフェンシングはRed Hat Cluster Suiteで共有リソースへのアクセスを禁止するための方法の1つです。HP iLOは、HP ProLiantサーバーやIntegriyサーバーにリモート管理機能を提供します。追加でインストールする必要はありません。システム管理作業は、サーバーのオペレーティングシステムの状態に関わらず、リモートで行えます。基本のiLO機能以外でも、サーバーの電源をオンとオフに切り替える機能が利用できるようになります。Red Hat Cluster Suiteでサポートする HP iLOフェンシング方式は、iLO電源オン/電源オフ機能に基づいています。

この項では、HP iLOフェンシングに基づく「使い慣れた」Red Hat Cluster Suite構成を検証していきます。

ネットワーク構成

HP iLOは主にシステムのリモート管理向けに設定されているので、ルーティング可能なネットワークで構成する必要があります。SGLXクラスターにおいて、HP iLOネットワークはクライアントアクセスネットワーク上で設定するか、または別のルーティング可能なネットワーク上で設定するかを選択できます。ルーティングできないプライベートネットワークでは設定できません。

Red Hat Cluster Suiteにおいて、HP iLOはCluster Manager（CMAN）で使用するクラスター通信ネットワークまたは別のネットワークのいずれにも接続できます。ただし、ネットワークのパーティション時にノードがフェンスデバイスへのアクセスを失わないよう、HP iLOはクラスター通信ネットワークに接続してはなりません。接続してしまうと、クラスターは一時停止状態になり、パーティションをフェンシングするにはオペレーターが介入し、手動でノードをリセットしてクラスターを開始しなければならなくなります。HP iLOを別のネットワーク（クラスター通信ネットワークとは別のネットワーク）に接続すれば、ノードはネットワークのパーティション時でもフェンスデバイスにアクセスすることができます。なお、HP iLOは、クラスターノード間で共有されないノードごとの電源管理を行います。2ノードクラスターでネットワークパーティションが発生すると、（クォーラムを持った）両ノードは互いに対してフェンシングを実行し、クラスター全体を停止させる可能性があります。2ノードクラスターではフェンシング競争を防止するため、1つのノードだけがクォーラムを獲得できるよう、*異常ノードの発見方法を持つ*`qdisk`が使用されています。この詳細については、「クォーラムの強化におけるqdiskの使用」で説明します。

次のガイドラインは、RHCSクラスターのHP iLOネットワーク構成において遵守すべき項目です。

1) HP iLOは、クライアントアクセスネットワークまたは別のネットワークのいずれにもアクセスできます。ただし、ネットワークはルーティング可能である必要があります。
2) HP iLOは、CMANがクラスター通信で使用するネットワーク上に配置するべきではありません。
3) 各クラスターシステムのHP iLOは、他のクラスターシステムすべてからネットワークを介してアクセスできなければなりません。

HP iLOフェンシングのタイミング

`fence_ilo`スクリプトでは、ノードをフェンシングするために、リモートInsightボードコマンド言語（RIBCL）インターフェイスを介してiLOと複数回ハンドシェイクする必要があります。RIBCLインターフェイスは`fence_ilo`スクリプトがiLOと通信するたびに新規接続が必要となり、接続の構築と解除に膨大な時間が費やされます。そのため、前リリースのRHEL5におけるHP iLOのフェンシング方式は、SCSI-3 PRよりも遅かったのです。`fence_ilo`の最新アップデートでは、この応答時間を改善しました。

x86_64システムのiLO2 ASICでは、従来の`fence_ilo`リリースは実行に約40～50秒かかり、その速度はクラスターのノード数に左右されないことが分かりました。（DLMやGFSなど）クラスターインフラストラクチャコンポーネントがノード障害の通知を受けると、すべての関連操作（GFS復旧、アプリケーションフェイルオーバーなど）はノードのフェンシングが完了するまで一時停止されます。フェンシングに時間がかかるほど、アプリケーションのフェイルオーバー時間は長くなります。

フェンシングが遅いと、障害ノードのリブートと重なってしまい、再度リセットの原因にもなりかねません。たとえばノード障害が発生し、システムパニックによってリセットが実行されたとします。フェンシングが遅い場合、iLOのリセットはブートシーケンスの最終フェーズで実行される可能性があり、その結果、二度目のリブートを招いてしまいます。iLOフェンシングが速ければ、ノードはノードパニック時に数秒でリセットできます。余分なリセットまたはリブート以外では、クラスターへのその他の影響はありません。

HP iLOとサードパーティハードウェア

HP iLOフェンシング方式は、内蔵iLOハードウェア対応のHP x86システムとIntegrityシステムでのみ使用できます。このフェンシング方式は、サードパーティシステム上のSGLXからの移行では使用できませんが、他ベンダーのサーバーにはその他の有効な「電源制御」フェンシング方式があります。

HP iLOフェンシングの信頼性

HP iLOフェンシングの通信は、iLOデバイスが接続されているネットワークに依存します。ネットワーク障害が発生するとフェンシング操作は失敗し、クラスターは一時停止状態になります。

Red HatのHP iLOフェンシングスクリプト(`fence_ilo`)は、RIBCL iLOインターフェイスを使って実装します。フェンシングのロジックは、(フェンシングするノード上の)iLOハードウェアへ次々に送信されるステータス/アクションコマンドのシーケンスとして実装されます。

`fence_ilo`スクリプトがiLOデバイスに送信するコマンドシーケンスの例は、次のとおりです。

1) サーバーの電源ステータスを取得
2) サーバーの電源を(オンになっている場合は)オフにする
3) サーバーの電源ステータスを取得
4) サーバーの電源を(オフになっている場合は)オンにする
5) サーバーの電源ステータスを取得

このようなハンドシェイクをベースにした実装は、SCSI-3 PRフェンシング方式で使用されている単一の原子的な操作と比べて、(頻度が低いとは言え)信頼性が低くなります。

このような頻度の低いHP iLOフェンシングの障害に対応するために、冗長性を確保するバックアップフェンシング方式を設定できます。HP iLOフェンシングが失敗した場合、構成されたバックアップ方式が実行されます。

結論

HP iLOをフェンシング方式として採用する際にユーザーが留意すべき重要な結論を、次にまとめます。

1. フェンシング方式としてのHP iLOは、その他の方式に比べて安価で、管理も簡単です。
2. システムが「ハードハング」という、非常にまれな状態に陥った場合でも、HP iLOフェンシングはノードをリセットします。SGLXでは、これは実行されません。
3. HP iLOは、CMANがクラスター通信で使用していないかぎり、どのルーティング可能なネットワークにも接続できます。

4. 同数ネットワークパーティション、半数のクラスターメンバーによる障害、またはSAN障害などの障害シナリオを処理するには、異常ノードの発見方法を持つqdiskが必要です。これは、SGLX環境では維持されます。詳しくは、「qdiskにおける推奨事項」の項を参照してください。
5. HP iLOフェンシング方式は、SCSI-3 PRなど他の方式と比べて遅くなります。フェンシングに時間がかかるほど、アプリケーションのフェイルオーバー時間は長くなります。
6. Serviceguard対応のサードパーティシステムに、HP iLOハードウェアは存在しません。その場合、DellのDRACやIBMのRSAといったハードウェア、またはAPCやWTIなど、外部ネットワークアドレスの割り当てが可能なPDUが有効です。
7. 頻度の低いHP iLOフェンシングの障害（スイッチの障害など）に対応するために、冗長性を確保するバックアップフェンシング方式を設定できます。

SCSI-3 PRフェンシングに基づく「使い慣れた」構成

この項では、SCSI-3 PR Persistent Reservation（PR）フェンシング方式に基づく移行のための「使い慣れた」Red Hat Cluster Suite構成を定義します。

フェンシング方式としてのSCSI-3 PRはRHEL5.2以前のリリースでサポートされていましたが、マルチパス環境での使用で大幅な制限がありました。SCSI-3 PRフェンシングはRHEL5.3のリリースより、特にDM-MPIOのマルチパスをサポートできるようになりました。「Persistent Reservationとマルチパス環境」の項では、RHEL5.2以前のリリースにおけるマルチパス環境でSCSI-3 PRを使用したときの問題について説明します。

注：共有ストレージ環境では、HP iLOフェンシング方式よりもSCSI-3 PRフェンシング方式を使用することが推奨されます。この項で説明するSCSI-3 PRベースの構成は、特にRHEL5.3およびDM-MPIOマルチパスで使用する場合、別のRHCS構成よりもSGLXクラスターに似ています。

SCSI-3 PRフェンシングについて

SCSI-3 PR（Persistent Reservation）は、複数の「参加」ノードが共有ディスクにアクセスしながら、同時に他のノードへのアクセスをブロックする方式です。これを実現するため、SCSI-3 PRは登録と予約という方法を使用します。「参加」を希望する各システムは、共有ディスクに対して独自の「キー」を登録します。システムへのアクセスをブロックすることは、デバイスからノードの登録を削除すること（プリエンプトとアボート）と同じくらい簡単です。ブロック操作は原子的に行われます。ノードは削除されると登録キーを持たなくなるので、ディスクへの書き込みもできず、他のノードを削除することもできません。これにより、スプリットブレイン状態を回避できます。

注：SCSI-2にもPRの概念がありますが、Red Hat Cluster Suiteではサポートしていません。

RHCSの`fence_scsi`フェンシングエージェントは上記で説明したSCSI-3 PRを採用し、クラスターメンバーへのアクセスを提供し、クラスターから削除されたメンバーへのアクセスを拒否します。Red Hat Cluster Suiteでは、すべてのデバイスからノードの登録キーを削除することで、SCSI-3 PR経由のフェンシングを実行します。ノード障害が発生した場合、`fence_scsi`エージェントはすべてのデバイスから障害ノードのキーを削除します。これにより、これらのデバイスへの書き込みを防止できます。

`fence_scsi`エージェントは、障害ノードによるこれらのデバイスへの書き込みを防止しますが、リブート後にノードがクラスターへ再び参加できるようリセットすることはありません。これには、ノードを手動でリセットする必要があります。

フェンシングされたSGLXノードも自動でリセットされるので、「使い慣れた」RHCSクラスターでも同様の機能を実現することが推奨されます。クラスター環境での手動による介入は推奨されず、これを回避できることから、使い勝手は改善されます。このほか、RHCSのメンバーシップアルゴリズムが「定義された」メンバーに依存するため、ノードバックアップは有用です。次の項では、ウォッチドッグタイマーをクラスターサービスとして統合し、SCSI-3 PRでフェンシングされたノードをリセットする方法について詳しく説明します。

ソフトウェア、NMI、およびハードウェアウォッチドッグタイマー

システムのハングはカーネルのループが原因で発生することがあり、他のタスクは実行すらできなくなります。こうしたハングは、2つのグループに分けることができます。ソフトロックアップとハードロックアップです。ソフトロックアップは一時的なロックアップのことで、他のタスクの実行やスケジューリングを遅らせます。ソフトロックアップとハードロックアップは、さまざまな種類のウォッチドッグタイマーで検知および管理できます。

1つめは、ソフトウェアベースのウォッチドッグで、ソフトウェアロックアップのみ扱えます。ハードロックアップは、システムを完全に無反応の状態に陥れます。ロックエラーにより、CPUが割り込みを無効化し、抜け出せなくなったときに発生することがあります。タイマーによる割り込みはハードロックアップで提供されていないので、スケジュールベースのソフトウェアウォッチドッグによる検知は使用できません。NMIハンドラまたはハードウェアベースのウォッチドッグタイマーが使用できます。

2つめは、ハードウェアベースの非マスク割り込み（NMI）ウォッチドッグです。Linuxカーネルでサポートされており、通常はシステムのマザーボードに搭載される特定のサーバーハードウェアに依存します。NMIウォッチドッグハードウェアは、安定したシステムの割り込みを検知できない場合にシステムのリブートを実行します。

最後に、3つのウォッチドッグタイマーの中で最も信頼性が高いのが、従来のハードウェアウォッチドッグタイマーです。これらのデバイスは、デバイスドライバーが定期的にデバイスをリセットしない場合に、システムを強制シャットダウン/リブートします。低いレベルのハードウェアウォッチドッグコンポーネントは統一性がないため、これらのコンポーネントを含む特定のシステムの決定について汎用的な説明はできません。多くの低いレベルのハードウェアウォッチドッグコンポーネントは、自己認識しません。

ソフトウェアベースのウォッチドッグタイマーは、ソフトハングによるシステムハング時でもノードのリセットをするのに十分です。ただし、非常にまれなハードハングの場合、フェンシングされたノードをソフトウェアベースのウォッチドッグタイマーでリセットすることはできません。ウォッチドッグタイマーでノードをリセットする目的は、ノードをフェンシングするためではなく、オペレーターの介入なくクラスターへ再び参加させるためです。SCSI-3 PRのような外部のフェンシング方式は、ノードをフェンシングしてデータの整合性を保証します。したがって、ノードのリセットに失敗しても、ノードが自動でクラスターに再び参加することを妨げるだけで、データの整合性には影響を与えません。このようなまれなケースでは、オペレーターが介入してノードをリセットする必要があります。

SCSI-3 PRでの自動リセット

この項では、SCSI-3 PRフェンシング方式でフェンシングされたノードをリセットする自動リセット方式について説明します。自動リセットすることで、障害ノードはオペレーターが介入しなくても、クラスターへ自動で再び参加することができます。

Linuxカーネルは、ソフトウェアのみのウォッチドッグタイマーまたはハードウェアベースのウォッチドッグタイマーのいずれかを使用してシステムをリセットします。

ウォッチドッグデーモンは/dev/watchdogを開き、システムをリセットさせない頻度で書き込みを続けます。デーモンが書き込みを停止すると、システムはウォッチドッグタイマーでリセットされます。ウォッチドッグデーモンが開始すると、バックグラウンドへ移動してから構成ファイルで指定されたチェックをすべて実行します。2つのテストを実行するごとに、ウォッチドッグデーモンはリセットを防ぐためにカーネルデバイスへ書き込みをします。そして、再度ロジックを実行するまで、事前に定義された時間はスリープに入ります。ウォッチドッグデーモンが設定された時間内に書き込みを行わないと、ウォッチドッグタイマーはノードをリセットします。

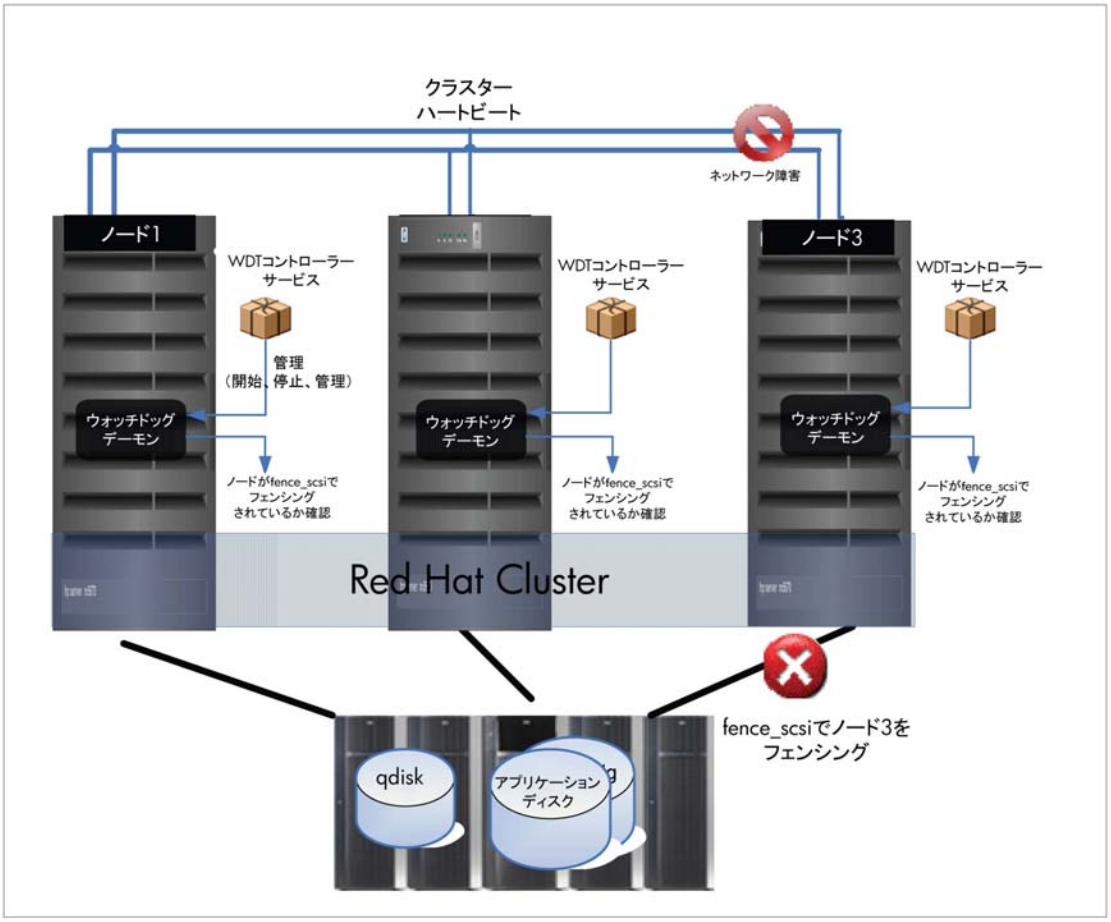
自動リセットはソフトウェアベースのウォッチドッグタイマーと、ノードがSCSI-3 PRによってフェンシングされたかどうかを確認するスクリプトを使って実装します。スクリプトは、システムを稼動させたままにするかリセットするかを判断します。リターンコードに基づき、ウォッチドッグデーモンはカーネルデバイスに書き込みするかどうかを決定します。スクリプトは、「テストバイナリ」パラメーターとして/etc/watchdog.conf内に構成されます。スクリプトはウォッチドッグデーモンによって定期的に起動され、ノードがSCSI-3 PRによってフェンシングされたかどうかを判断します。これは、共有ディスクにノードの登録キーがあるかどうかを確認することで実施します。サンプルのスクリプトは、付録にあります。

注： ソフトウェアベースのウォッチドッグタイマーは、ほとんどのシステムハング時でもノードのリセットをするのに十分です。ただし、非常にまれなケースですが、フェンシングされたノードをソフトウェアベースのウォッチドッグタイマーでリセットできないことがあります。SGLXでも同様です。「デッドマンドライバー」はほとんどの場合、システムをリセットしますが、「ハードハング」ではリセットできません。

クラスターサービスがクラスターノード上で手動にて停止させられた場合、ウォッチドッグデーモンは必ずシャットダウンしなければなりません。これにより、手動でクラスターから削除されたノードが、必要もなくリセットされることを防ぎます。クラスター内の各ノードは、ウォッチドッグタイマー（WDT）コントローラーサービスと呼ばれる、ウォッチドッグデーモンを起動、監視、および停止する制限サービスで構成できます。WDTコントローラーサービスは、ノード上でクラスターサービスが起動したときにウォッチドッグデーモンも起動するよう制御し、ノード上のクラスターサービスをオペレーターが停止させたときはウォッチドッグデーモンを停止させます。WDTコントローラーサービスは、フェンシングされたノードと手動でクラスターから削除されたノードをウォッチドッグデーモンが区別できるよう、単純かつ信頼性の高い管理性を提供します。WDTコントローラーサービススクリプトのサンプル実装は、付録にあります。図2は、それを示したものです。

図2は、次のとおりです。

図2: RHCSクラスターにおけるSCSI-3 PRでの自動リセット



Persistent Reservationとマルチパス環境

この項では、RHEL5.2以前のリリースのマルチパス環境でSCSI-3 PRを使用した場合の問題点を説明します。これは、RHEL5.3のリリースで対応しました。

SCSI Primary Commands-3 (SPC-3)仕様は、SCSI-3 PRの動作を定義したものです。マルチパス環境におけるSCSI-3 PRのサポートを定義しており、(ホストからディスクまでのマルチパスにおける)ノードはHBA/アレイコントローラーのペアを登録することが義務づけられています。RHEL5.2以前のリリースでは、Device Mapperマルチパス(DM-MPIO)ドライバーはPRコマンドをLUNのすべての物理パスに転送しませんでした。これは、SCSI-3 PRを正しく機能させる上でも必要な作業です。

RHCSでは(「start」オプションによる)起動時に、`scsi_reserve` initスクリプトは検出したデバイスすべての登録を作成します。「stop」オプションで`scsi_reserve`は、登録したすべてのデバイスからノードの登録キーを削除(登録解除)します。`scsi_reserve`は`sg_persist`コマンドを使って、キーの登録または登録解除を行います。マルチパス環境では、この`sg_persist`コマンドをDMドライバーが受け取り、基盤のSCSIドライバーへと受け渡します。

DM-MPIOドライバーは、`sg_persist`登録(登録解除)コマンドをLUNの一つの物理パスへのみ転送します。ここで求められるのは、DM-MPIOドライバーがこのコマンドをLUNのすべての物理パスへ受け渡すか、もしくは`sg_persist`コマンドがこれらのパスを検出してそれぞれに登録するという動作です。アクティブ/パッシブアレイにおいて、登録(または登録解除)コマンドはアクティブパスへのみ送信され、スタンバイパスへは送信されません。アクティブ/アクティブアレイであれば、このコマンドはDM-MPIOポリシーに基づいて動的に選択された単一パスへ転送されます。

こうしたDM-MPIOの動作は、現行の形式にてSCSI-3 PRがフェンシング方式として採用された場合、RHCSでは次のような結果になります。「静的パス選択」ベースの構成において、一度パスが選択されると、障害が発生するまでI/Oとして使用されます。ノードの登録は、初期に選択されたパスで実行されます。このパスで障害が発生すると、(登録されていない)残りのパスへのI/Oリトライは失敗してしまいます。「動的パス選択」ベースの構成では、無作為に選択されたパスを各I/Oで使用します。このパスは、登録されたパスと異なることがあります。ノードの登録は、無作為に選択されたパスで実行されます。各I/Oのパスを動的に選択する方法の場合、登録されたパスと未登録のパスの間で「フェイルオーバー、フェイルバックのピンポン」が発生します。「ピンポン」はすべての未登録パスが失敗と認識され、利用可能なパスとして登録パスが残されるまで続けられます。登録パスが失敗すると、他にI/Oが利用できるパスまたは登録パスが存在しない理由から、デバイスへはアクセスできなくなります。登録パスがいずれかの構成で失敗した場合、アプリケーションはデータを利用できなくなります。アプリケーションI/Oで障害が出ているときも、サービスはクラスターノードの別のノードへ移動されないまま、ノード上で実行し続けます。

したがって、登録パスでの障害の発生は、たとえデバイスへの他の正常なパスがあったとしても、デバイスへのアクセスを失うのに十分ということです。RHEL5.2以前のリリースでSCSI-3 PRフェンシングを使用すると、マルチパスのメリットはすべて享受できません。

注: すべてのアクティブ/パッシブアレイがマルチパス環境のSCSI-3 PRで動作するわけではないので、サポートしていません。

SCSI-3 Persistent Reservationを使用する場合は、マルチパス環境における上記の影響を制限する方法に関する、次のガイドラインを参照してください。

1. 「静的パス選択」ベースの構成によるアクティブ/アクティブアレイのみを使用します。これにより、登録パスと未登録パス間の「フェイルオーバー、フェイルバックのピンポン」を回避できます。RHEL5では、`/etc/multipath.conf`の*`path_grouping_policy`*パラメーターを「静的パス選択」のfailoverに設定します。
2. ストレージにアクセスできなくなったとき、qdisk構成を使用してノードをフェンシングします。これにより、すべてのクラスターサービスは別のクラスターメンバーへと移動され、データへアクセスできないままにノード上でクラスターサービスを実行することを回避できます。登録パスの障害時には、デバイスへのアクセスはすべて失われます。そのノード上のqdiskdは、qdisk内にある自身の領域に書き込めなくなります。そのノードのqdiskdはこれを検知し、CMANにノードをフェンシングするよう要求します。フェンシングされると、ノードは「自動リセット」方式の設定によってリセットされます。ノードは、リブートしてクラスターに参加します。このとき、パス障害も解決することがあります。

結論

SCSI-3 PRをフェンシング方式として採用する際にユーザーが留意すべき重要な結論を、次にまとめます。

1. HP iLOフェンシングと比較して、SCSI-3 PRフェンシングはより多くのデバイス数を抱えていたとしても、高速に処理できます。したがって、この方法であればアプリケーションのフェイルオーバー時間を大幅に短縮することができます。

2. SCSI-3 PRフェンシングは、SCSI-3 PRフェンシング要件が一致する場合に、Serviceguardをサポートするサードパーティシステムでもサポートすることができます。したがって、SCSI-3 PRフェンシングによって、オペレーターは一般的なフェンシング方式をHPシステムと非HPシステムの両方で使用することができるようになります。これに対して、HPシステムでHP iLOフェンシング方式を使用した場合、非HPシステムでは別のフェンシング方式を選択しなければなりません。

3. SCSI-3 PRをフェンシング方式として使用するには、すべての共有ストレージでLVM2クラスターボリュームを使用する必要があります。これは、デバイス検出を行う上で必要です。フェンシングスクリプトは、共有デバイスを検出する上でLVMコマンドに依存します。検出するには、ボリュームグループがクラスター対応である必要があります。移行する場合、SGLXにおけるすべてのボリュームグループの設定は、RHCSで使用する前にクラスターボリュームグループへ変換します。このほか、このボリューム内のすべてのデバイスはSPC-3への対応が必要です。

4. クラスター内のすべてのノードでは、ストレージの見え方に一貫性が必要です。つまり、クラスター内のすべてのノードは同じデバイスに登録する必要があるということです。これは、各ノードが別ノードの登録キーを、登録されているすべてのデバイスから削除できるようにするためです。これにより、フェンシング操作を実行しているノードは、他のノードが登録されているすべてのデバイスを認識できなければなりません。この要件を満たすには、デバイス名に一貫性を持たせる必要があります。

5. SCSI-3 PRフェンシング方式は、RHEL 5.3リリース以降のマルチパス構成でのみサポートされます。

6. RHCSクラスターでSGLX環境におけるさまざまな障害シナリオを扱えるよう、qdisk構成を定義します。

RHCSリソース、サービス、およびフェイルオーバードメイン

この項では、クラスターリソース、サービス、およびフェイルオーバードメインのコンセプトについて説明します。次の項で説明するパッケージの移行を理解する上で役に立ちます。

サービスとリソース

Oracleデータベースのようなアプリケーションは、RHCSサービスで構成することにより、高い可用性を実現します。クラスターサービスは、あるノードから別ノードへフェイルオーバーできるコンポーネントである、クラスターリソースで構成されます。多くのサービスに共通するクラスターリソースの一般例では、IPアドレス、ボリュームグループ、ext3ファイルシステムなどが挙げられます。クラスターサービスを構築すると、フェイルオーバー時にもアプリケーションに対して透過的なクライアントアクセスができます。ハードウェアまたはソフトウェアで障害が発生した場合、クラスターは障害ノードのクラスターサービスを稼動ノード上で自動的に再起動します。これによって、データ消失を防ぎ、ユーザーへの途絶も抑えることができます。

RHCSサービスは、SGLXパッケージに似ています。アプリケーションサービスはこのパッケージにまとめられており、障害時にはクラスター内の別ノードへと自動転送され、最小限の中断でサービスを提供し続けることができます。

フェイルオーバードメイン

クラスターサービスは、フェイルオーバードメインと関連付けられます。フェイルオーバードメインは、特定のクラスターサービスを実行するのに適切なクラスターノードのサブセットです。ただし、各クラスターサービスはデータの整合性を保つため、一度に1つのクラスターノード上でしか実行できません。フェイルオーバードメイン内のどのノードで実行するかは、優先順位を付けられます。クラスターサービスは、関連するフェイルオーバードメインのノード上でのみ実行するよう制限できます。

クラスターサービスを制限されたフェイルオーバードメインに割り当てることで、フェイルオーバー時にクラスターサービスを実行するノードを限定できます。また、(そのノードがアクティブである限り)特定のノードでクラスターサービスを実行するよう、フェイルオーバードメイン内のノードに優先順位を付けることもできます。

フェイルオーバードメインの特長は、次のとおりです。

- Unrestricted: メンバーのサブセットが優先されます。ただし、このドメインに割り当てられたクラスターサービスは、利用可能なクラスターメンバーのいずれでも実行することができます。
- Restricted: クラスターサービスは、メンバーのサブセットでのみ実行できます。
- Unordered: クラスターサービスを実行するメンバーは、優先順位が付けられていないフェイルオーバードメインメンバーの利用可能リストから選択します。
- Ordered: クラスターサービスを実行するドメインメンバーを、優先順位に従って選択できます。リストの一番上にあるメンバー(`/etc/cluster/cluster.conf`で指定)は優先順位が高く、降順で記述されます。

順序付けと制限フラグを使用すると、別のタイプのフェイルオーバードメインを構築できます。構築可能なドメインは、「ordered、restricted」、「unordered、restricted」、「ordered、unrestricted」、「unordered、unrestricted」です。

フェイルオーバードメインについて詳しくは、RedHat社のWebサイトhttp://sources.redhat.com/cluster/wiki/FailoverDomainsを参照してください。

リソースエージェント

リソースエージェント（RA）は、指定されたリソース（IPアドレス、ファイルシステム、LVMなど）の操作を扱うためのスクリプトまたは実行可能プログラムです。

RAは、Resource Managerのリクエストを受け、指定されたリソースインスタンスに対して次のアクションを実行できなければなりません。

1. *Start:* リソースインスタンスをオンラインにして、利用できるようにします。
2. *Stop:* リソースエージェントを停止します。
3. *Monitor:* リソースインスタンスの現在のステータスを確認し、返します。
4. *Meta-data:* stdout経由でリソースエージェントのメタデータを返します。

リソースエージェントの機能（start、stop、monitor、およびmeta-data）は、RHCSクラスター環境でアプリケーションが動作できるよう、OCFに対応している必要があります。リソースエージェントの動作について詳しくは、RedHat社のWebサイトhttp://sources.redhat.com/cluster/wiki/RGManagerのOCF RA API v1.0を参照してください。

サポートされるリソースエージェントは、Red Hat Cluster Suite管理者ガイドを参照してください。RHCSでは、現在Red Hat Cluster Suiteで提供されていないリソースを扱うための*script* RAも提供しています。

SGLXパッケージの移行

前の項では、2つの「使い慣れた」RHCSクラスター構成を使用してSGLXクラスターを移行する方法について説明しました。この項では、SGLXパッケージをRHCSクラスターサービスへ移行する方法について説明します。

Servicegaurdパッケージ

次の表に、このWhite Paperで移行方法について説明するパッケージのタイプを示します。

表1：RHCSクラスターへのSGLXパッケージの移行

パッケージのプロフィール	概要
カスタマー定義スクリプト付きのレガシーパッケージ	制御スクリプトのカスタマー定義用の領域に、別のスクリプトを起動するコードや、Serviceguard制御スクリプトテンプレートにはない機能を持つレガシーフェイルオーバーパッケージ。
カスタマー定義スクリプト付きのモジュラーパッケージ	制御スクリプトのカスタマー定義用の領域に、別のスクリプトを起動するコードや、Serviceguard制御スクリプトテンプレートにはない機能を持つモジュラーフェイルオーバーパッケージ。
制御スクリプトおよびユーザー定義変数付きのレガシーパッケージ	パッケージ制御スクリプトに、Serviceguardのパラメーターとは異なるユーザー定義の環境変数を持つレガシーフェイルオーバーパッケージ。
制御スクリプトおよびユーザー定義変数付きのモジュラーパッケージ	パッケージ制御スクリプトに、Serviceguardのパラメーターとは異なるユーザー定義の環境変数を持つモジュラーフェイルオーバーパッケージ。

OracleやSambaなど、さまざまなSGLXツールキットの移行方法については、後に別のWhite Paperで説明します。

パッケージのタイプ

SGLXフェイルオーバータイプパッケージは、一度に1つのノード上で稼動し、障害時には別のノードへ切り替わります。Red Hat Cluster Suiteでは、このタイプのみをサポートしています。SGLXマルチノードパッケージは、同時に複数のノード上で稼動し、個々のノード上で個別に起動したり停止したりすることができます。これは、Red Hat Cluster Suiteではサポートされていません。同様に、SGLXシステムマルチノードパッケージは、同時にすべてのクラスターノード上で稼動し、個々のノード上で個別に起動したり停止したりすることができません。これも、Red Hat Cluster Suiteではサポートされていません。

Red Hat Cluster Suiteはシステムマルチノードまたはマルチノードタイプのパッケージをサポートしていない一方で、RedHat社のWebサイトhttp://sources.redhat.com/cluster/wiki/MultipleInstanceServicesで「マルチインスタンス」サービスという回避策を提供しています。これは、同一サービスを複数のノードで実行するためのものです。マルチノードまたはシステムマルチノードタイプのパッケージについて、全機能は提供していません。このような回避策でも、複数のノード上で稼動するサービスは、マルチノードまたはシステムマルチノードタイプのパッケージをクラスターレベルで管理するSGLXとは異なり、ノードごとに管理する必要があります。つまり、複数ノード上のサービスは、マルチノードまたはシステムマルチノードタイプのパッケージのように、1つのコマンドで一斉に開始または停止することができないということです。

注：このWhite Paperで、実装するウォッチドッグタイマー（WDT）コントローラーサービスは、「マルチインスタンスサービス」の回避策の例です。

再起動、フェイルオーバー、およびフェイルバック

Serviceguardでは、パッケージが正常に開始すると、パッケージマネージャープロセスが「パッケージサービス」のプロセスIDを監視します。サービスに障害が発生し、そのサービスの再起動パラメーターの値が0以上に設定されている場合、サービスはパッケージを停止させることなく、同じノード上で再起動されます。ただし、再起動の回数が上限を超えている場合、パッケージは現行のノード上で停止され、パッケージ切り替えフラグとフェイルオーバーポリシーに従って、別のノードで開始されます。RHCSクラスターでは、リソースが*status*チェック関数のコールに失敗した場合、リソースマネージャーは影響のあったサービスを停止させます。*recovery*パラメーター値に基づき、サービスは再起動、再配置、または無効化されます。パラメーターがrestartに設定されている場合、サービスは（フェイルオーバードメインポリシーに基づいて）最も利用しやすいオンラインノードへと再配置される前に、ローカルで再起動されます。どのノードも利用できない場合、またはすべてのメンバーでサービスを開始できない場合は、サービスが無効化されます。パラメーターが*relocate*に設定されている場合、サービスはローカルで再起動されることなく、別ノードへ再配置されます。パラメーターが*disable*に設定されている場合は、サービスの起動は無効化されます。

Serviceguardではサービスを複数回（無制限まで）再起動することができますが（*service_restart*パラメーターで定義）、RHCSでは一回しか再起動できません。したがって、SGLXパッケージを移行するときは、最大で一回のみサービスを再起動することができます。

Serviceguardにおいて、パッケージ構成パラメーターの*failover_policy*は、パッケージの再配置時にノードを選択する際に使用します。このパラメーターは、configured_nodeまたはmin_package_nodeのいずれかに設定できます。configured_nodeポリシーは、Serviceguardが*node_name*エントリーのリストから優先度の順番でノードを選択することを示しています。min_package_nodeで構成すると、パッケージの起動が必要なとき、Serviceguardは*node_name*エントリーのリストから最も少ないパッケージを稼動しているノードを選択します。

Red Hat Cluster Suiteでは、フェイルオーバードメインタイプを使用して、サービスを再配置するノードを選択します。「ordered、restricted」のフェイルオーバードメインでは、最も優先度の高いメンバーでオンラインになっているものを常に選択し、ドメイン上のサービスを実行します。たとえば、ドメインメンバーがノード{A、B、C}の場合、メンバー「A」がオンラインになっており、クォーラムを持っていれば、サービスは常にメンバー「A」で実行されます。つまり、メンバーAの優先順位がメンバーBよりも高い場合、メンバーBで実行中のサービスはメンバーAがオフラインからオンラインになると、メンバーAに移行するということです。すべてのドメインメンバーがオフラインの場合、サービスは実行されません。「ordered、restricted」タイプのフェイルオーバードメインは、configured_nodeフェイルオーバーポリシーと同様の動作を行います。

Serviceguardでは、*failback_policy*パラメーターを使用して、プライマリノードでパッケージが実行されておらず、プライマリノードが実行可能になったときのパッケージに対するアクションを決定します。manualに設定している場合、採用されたノードでパッケージが実行されていれば、パッケージがプライマリノードへ移動することはありません。automaticに設定されている場合、プライマリノードがパッケージを実行できるようになると同時に、Serviceguardはパッケージをプライマリノードへ移動させます。Red Hat Cluster Suiteでは、orderedタイプのフェイルオーバードメインの*nofailback*ポリシーを有効にすることで、「より適した」ノードがクラスターに復帰したとき、自動でフェイルバックしないようにします。

現在、Red Hat Cluster Suiteは、SGLX min_package_nodeフェイルオーバーポリシーと同等のフェイルオーバードメインをサポートしていません。このようなパッケージでは、Red Hat Cluster Suiteがサポートするフェイルオーバーポリシーに変更する必要があります。表2に、SGLXパッケージ構成と同等のRHCS構成を示します。

表2：SGLXパッケージ構成と同等のRed Hat Cluster Suite構成

SGLXパッケージ	Red Hat Cluster Suiteサービス
failover_policy を configured_node に設定 *failback_policy*をautomaticに設定 *service_restart > 0*	*ordered*を*1*に設定 ノードの優先順位をServiceguardパッケージ構成の*node_name*にリストされた順番で設定 *restricted*を1に設定 *nofailback*を0に設定 *recovery*を*restart*に設定（最大一回まで再起動可）
failover_policy を configured_node に設定 *failback_policy*をmanualに設定 *service_restart*を0に設定	*ordered*を*1*に設定 ノードの優先順位をServiceguardパッケージ構成のnode_nameにリストされた順番で設定 *restricted*を1に設定 *nofailback*を1に設定 *recovery*をrelocateに設定（サービス障害時は再起動を行わない）
*failover_policy*をmin_package_nodeに設定	サポートされていません

注：モジュラーパッケージで使用されるパラメーター名と値は小文字ですが、レガシーパッケージのデフォルトでは大文字になっています。ただし、指定されていない限り、このWhite Paperで説明するモジュラーパッケージのパラメーター名と値はレガシーパッケージでも有効とします。

自動起動機能

Serviceguardにおいて、*auto_run*パラメーターはクラスター起動時にパッケージを開始すべきかを決定したり、障害時に新規ノードでパッケージを自動的に再起動すべきかを決定したりします。同様に、Red Hat Cluster Suiteでは*autostart*をyesに設定すると、クラスターの起動と共にサービスが自動起動し、障害時には採用されたノードで起動するようになります。したがって、*autostart*オプションは*auto_run*オプションで構成されるServiceguardパッケージの移行で使用できます。

高速フェイルオプション

Serviceguardでは、*node_fail_fast_enabled*をyesに設定すると、パッケージの障害時にノードはリセットされます。さらに、Serviceguardでは*service_fail_fast_enabled*もサポートしており、yesに設定するとサービス障害時にノードをリセットします。

RHCSでは、*hardrecovery*オプションをyesに設定すると、ノード上でのサービスの停止に失敗した場合、メンバーはリブートされます。

*node_fail_fast_enabled*または*service_fail_fast_enabled*とまったく同じというわけではありませんが、サービス停止に失敗した場合は、*hardrecovery*オプションを使用してノードをリセットできます。サービス停止が失敗したことを考えれば、サービスを再配置するよりも単純にノードをリブートした方が安全です。

パッケージの依存関係

Serviceguardにおいて、パッケージは別のパッケージと依存関係を持つことができます。これは、主に*dependency_condition*および*dependency_location*のパラメーターを使って記述します。*dependency_condition*パラメーター（upまたはdown）は、依存関係を満たす上で必要なパッケージの状態を表します。upに設定すると、このパッケージはpackage_nameで識別されたパッケージがアップである事が必要という意味になります。同様に、downに設定すると、このパッケージは*package_name*で識別されたパッケージがダウンである事が必要になるという意味になります。*dependency_location*は、same_node、any_node、およびdifferent_nodeのどの条件を満たすべきかを示すものです。

Red Hat Cluster Suiteでは、サービスは別のサービスに対して依存関係を持つことができます。サービス依存オプションの*depend*は、依存するサービス名を指定するときに使用できます。この依存関係は、サービス依存モードと呼ばれる*depend_mode*パラメーターを基に制限されます。モードをhardに設定すると、サービスは依存するサービスがアップであるときのみ、アップします。この依存関係は、初期サービス起動時のほか、サービス実行中にも満たす必要があります。依存するサービスが停止すると、そのサービスも停止します。*depend_mode*がsoftに設定されると、依存関係は起動時にのみ満たされる必要があります。依存するサービスがアップしたときのみ、そのサービスはアップします。ただし、一度起動すると、依存するサービスが停止しても、サービスは停止しません。

イベントスクリプティングは、クラスター操作時に発生するさまざまな現象を基にサービスの移行を実行する方法の1つです。ここでのイベントスクリプティングは、RHCSでは容易に利用できないSGLX依存関係機能を実装する際に使用します。これは、RHCSには同一のものがないSGLXオプションを、スクリプトを使用してプログラムで簡単に実装できることを示した例です。イベントスクリプティングについて詳しくは、RedHat社のWebサイトhttp://sources.redhat.com/cluster/wiki/RGManagerを参照してください。

次に、RHCSでは（*depend*および*depend_mode*経由で）容易にサポートされていないSGLXの依存関係について説明します。これは、イベントスクリプティングで実装可能です。

1) *dependency_condition*がdownに設定されたSGLX構成すべて

2) *dependency_location*がsame_nodeまたはdifferent_nodeに設定されたSGLX構成すべて。*same_node*がサポートされていないことから、Serviceguardの*priority*パラメーターは意味がなくなります。

Red Hat Cluster Suiteでサポートされている唯一のSGLX構成のタイプは、次のとおりです。

- *dependency_condition*がupに設定され、*dependency_location*がany_nodeに設定されている構成。Serviceguardでは、依存関係の条件の確認は起動時に実行され、その後も必要に応じて実施されます。したがって、Red Hatと同様の動作を実行するには、*depend_mode*をsoftに設定する必要があります。

クライアントネットワークの監視

Serviceguardは、パッケージ向けにサブネットを監視するよう設定できます。このサブネットは、パッケージ構成ファイル内の*monitored_subnets*パラメーター（レガシーパッケージでは*SUBNET*パラメーター）で定義できます。定義されたサブネットが停止すると、パッケージは自身向けに構成され、定義されたすべてのサブネットを持つ別ノードへ再配置されます。

RHCSでも、IPアドレスのリソースが属するサブネットを監視できますが、その他のサブネットは監視できません。IPアドレスリソースの*monitor_link*パラメーターを*yes*に設定すると、IPアドレスがバウンドされたNICのリンクが存在しない場合、*status*関数は失敗します。失敗すると、リソースマネージャーがクラスター内の別ノードにサービスを再配置します。

*monitored_subnets*で構成されたSGLXパッケージを移行する場合、RHCSクラスターではIPアドレスが所属するサブネットしか監視できません。その他のサブネットでSGLXでは監視対象となっていたものは、RHCSでは監視されません。SGLX監視ではこのように設定ができますが、一般的ではありません。

アプリケーションの起動、シャットダウン、および監視

RHCSでは現在、OracleやSambaなどの標準アプリケーションを完全サポートするリソースエージェントを提供しています。既存のSGLXのお客様は、Red Hatが提供するリソースエージェント対応のアプリケーション向けに、カスタマー定義の領域（レガシースクリプト）か外部スクリプト（モジュラーパッケージ）を実装している可能性があります。既存のSGLXクラスターにおいて、アプリケーションはレガシーパッケージのカスタマー定義領域またはモジュラーパッケージの外部スクリプトのいずれかを使用して、高い可用性を確保できます。モジュラーパッケージの場合、*external_script*がアプリケーションの起動と開始を行うスクリプトを認識します。こうしたケースでは、カスタマー定義領域または外部スクリプトを新規Red Hatリソースエージェントへ移植するよりも、提供されるリソースエージェントを使用することを推奨します。

ただし、多くの場合、カスタマー定義の領域や外部スクリプトと同等のリソースエージェントは存在しないかもしれません。このようなときは、カスタマー定義の領域または外部スクリプトをRed Hatの*script*リソースとして移植する必要があります。RHCSの*script*リソースタイプは、Linux Standard Base（LSB）対応スクリプトをクラスターサービスとして統合する事ができます。これにより、アプリケーションは起動、停止、監視できるようになります。移植には、「リソースエージェント」の項で説明する必須アクションの*start*、*stop*、*monitor*、および*meta-data*の実装が必要になります。

レガシーパッケージでは、制御スクリプトのカスタマー定義領域は「*#START CUSTOMER DEFINED FUNCTIONS*」と「*#END CUSTOMER DEFINED FUNCTIONS*」のコードの間となります。*customer_defined_run_cmds*関数の基盤にあるロジックは、*script*リソースの*start*関数として移植する必要があります。モジュラーパッケージの場合、*external_script*がアプリケーションの起動と開始を行うスクリプトを認識します。*external_script*内の*start*関数は、*script*リソースの*start*関数として移植する必要があります。

同様に、*customer_defined_halt_cmds*関数の基盤にあるロジックを*script* RAリソースの*stop*関数として移植する必要があります。モジュラーパッケージにおいて、*external_script*内の*stop*関数は、*script*リソースの*stop*関数として移植する必要があります。

RHCSでは、「サービス」という用語はリソースのコレクションを意味し、SGLXのパッケージと同等です。しかし、Serviceguardでは、「サービス」はパッケージがアップのときに監視されるデーモンを指します。Serviceguardはパッケージサービスを使用して、パッケージがアップかダウンかを判断します。Serviceguardにおいて、パッケージサービスはパッケージのアップまたはダウンの状態を判断するメカニズムを提供します。Red Hatでも、各リソースエージェントによって実装される*monitor*関数を経由して同様の機能を確保できます。この関数はサービスのヘルスを確認するため、リソースグループマネージャーから定期的に呼び出されます。その期間は、/usr/share/cluster/script.shファイル内にある*monitor*アクション（デフォルトは30秒）で指定した*interval*パラメーターで定義されます。

定期的に監視するためのデフォルト設定を記述した/usr/share/cluster/script.shファイルの一部は、次のとおりです。

```
<action name="monitor" interval="30s" timeout="0"/>
```

移行の一環として、*SERVICE_CMD*スクリプトのロジックは*script* RAリソースの*monitor*関数に移植する必要があります。

レガシーパッケージの場合、カスタマー定義領域で使用されているレガシーパッケージ制御スクリプト内に非Serviceguard変数が定義されている可能性があります。また、カスタマー定義領域をスクリプトとして実装した場合、スクリプトには追加の変数が定義されている可能性があります。これら両方の変数のセットは、新たに実装するscript RAへ移植する必要があります。同様に、モジュラーパッケージにも、パッケージ構成と、同様に外部スクリプトファイルにも非Serviceguard変数が定義されている可能性があります。これらは、新たに実装する*script* RAへ移植する必要があります。

Script RAリソースは、LSB仕様に準拠する必要があります。詳しくは、Free Standards GroupのWebサイトhttp://refspecs.freestandards.org/LSB_2.0.1/LSB-Core/LSB-Core/iniscrptact.htmlを参照してください。準拠しない場合、特に*start*、*stop*、および*monitor*関数のリターンコードが準拠していない場合、リソースマネージャーの動作に一貫性がなくなることがあります。

「移行手順」の項では、カスタマー定義スクリプトまたは外部スクリプトをRed Hatの*script*リソースへ移植する例について説明します。付録には、移植された*script*リソースのサンプルリストがあります。

パッケージの起動およびシャットダウン/サービスの階層構造

RHCSにおいて、サービスとは高可用性を実現するために、管理(起動、停止、または再配置)対象となる1つのエンティティに構成されたクラスターリソースのコレクションを指します。サービスは、各リソース、その属性、およびその他リソースとの関係を指定するリソースツリーで表します。関係には、親、子、または孫などがあります。サービスは1つのエンティティに見えますが、サービス内で起動および停止する各リソースの順番はリソースの階層が決定します。

親子関係の場合、起動またはシャットダウンは単純です。すべての親は子の前に起動し、子は親が停止する前にすべて速やかに停止していなければなりません。リソースのヘルスが良好であると判断するには、すべての子のヘルスが良好である必要があります。

タイプ属性を持つ子のリソースにおいて、子のリソースの「タイプ」属性は各リソースタイプの開始順序と停止順序を定義します。*service.sh*サービスリソースエージェントに記述された開始値および停止値は、次のとおりです。

```
<special tag="rgmanager">
    <attributes root="1" maxinstances="1"/>
    <child type="lvm" start="1" stop="9"/>
    <child type="fs" start="2" stop="8"/>
    <child type="clusterfs" start="3" stop="7"/>
    <child type="netfs" start="4" stop="6"/>
    <child type="nfsexport" start="5" stop="5"/>
    <child type="nfsclient" start="6" stop="4"/>
    <child type="ip" start="7" stop="2"/>
    <child type="smb" start="8" stop="3"/>
    <child type="script" start="9" stop="1"/>
</special>
```

このサービスリソースでは、すべての子のLVMが最初に開始され、次に子のファイルシステムが開始されると、続けてすべての子のスクリプトが開始されます。タイプ属性を持たない子のリソースについては、追加で検討が必要です。

タイプ属性を持たない子のリソースの場合、その開始順序や停止順序はサービスリソースで明確に指定されていません。開始順序と停止順序は、/etc/cluster/cluster.conf内の子リソースの順番に応じて決定します。さらに、タイプ属性を持たない子のリソースはすべてのタイプ属性を持つ子リソースが開始した後で開始し、すべてのタイプ属性を持つ子リソースよりも前に停止します。リソースタイプ内の順序は、クラスター構成ファイルの/etc/cluster/cluster.conf内に保存されます。

Servicegurdパッケージの移行では、タイプ属性を持つ子のリソースのみを使用します。したがって、リソースはSGLXパッケージと同じ順番で開始(または停止)します。

Red Hatでは、多くのサービス間で共有できるよう、共通のリソースブロック内でリソースの設定を行えます。こうした再利用は、必要な属性をサービスブロック内にある親リソースから継承できることが条件です。これは、複数のサービスが互いに競合しないようにするためです。こうした利用は、NFS構成では可能かつ有益です。その他リソース内のアプリケーションでは、再利用できません。たとえばファイルシステムのリソースが複数箇所にあると、2つのノードに1つのファイルシステムをマウントすることになり、破損を引き起こします。同様に、これを*IPアドレス*および*ボリュームグループ*のリソースで使用することは適切ではありません。

RHCSにおいて、サービスのリソースで障害が発生すると、サービスも障害が発生したと見なされます。このような場合、期待されるアクションはすべてのリソース、障害が発生したサービス、そして障害が発生していない残りのサービスを含め、すべてのサービスを再起動することです。状況によっては、通常の復旧を実行する前にサービスの一部のみを再起動するので十分なことがあります。これは、independent_subtree属性を使用して実施できます。(クラスター環境でアプリケーションを開始、停止、および監視するときに使用する)アプリケーションの*script*リソースは、個別のツリーとして作成できます。これにより、障害が発生した場合でも、LVM、IP、およびFSなどその他リソースではなく、スクリプトのみを再起動することができます。起動時の問題は、他すべてのリソースを不要に再起動させることなく、アプリケーションの再起動のみにすることで解決できる場合があります。

サービスIPアドレス

パッケージでは、指定されたサブネット上の1つ以上の再配置可能なIPアドレスを定義できます。同様に、Red Hat Cluster SuiteでもIPアドレスリソースを定義して、再配置可能なIPアドレスを構成できます。パッケージ構成ファイルでの*ip_address*エントリーごとに、それぞれ別のIPリソースを作成する必要があります。

SGLXパッケージの移行では、パッケージ構成の*ip_address*エントリーごとに別のIPリソースを作成します。

注: Red Hat Cluster Suiteは、クラスターサービスで使用できるよう、IPv4アドレスとIPv6アドレスの両方をサポートします。クラスター通信でサポートするのは、IPv4のみです。

ボリュームグループ

パッケージ構成ファイルの*vg*パラメーターは、ファイルシステムをマウントするLVMボリュームグループを指定するものです。パッケージに対して、*vg*は複数定義できます。同様に、RHCSでもLVMリソースをアプリケーションのボリュームグループとして構成できます。

SGLXパッケージの移行では、パッケージ構成の*vg*エントリーごとに別のLVMリソースを作成します。

注: RHCSにおいて、LVMリソースエージェントが「管理」するのは、「単一システム」タイプのボリュームグループのみです。Cluster LVMデーモン(CLVMD)が管理する「クラスター対応」タイプのボリュームグループでは、リソースエージェントを必要としません。「SCSI-3 PRフェンシングに基づく「使い慣れた」構成」を使用する場合、パッケージの*vg*エントリーに対してLVMリソースを作成しないでください。これは、SCSI-3 PRフェンシングの場合、パッケージ構成ファイルの「単一システム」のボリュームグループを「クラスター対応」のボリュームグループに変換しなければならないからです。

ファイルシステム

開始時において、SGLXパッケージは(ファイルシステムに関連する)1つ以上の論理ボリュームをアクティブ化し、ファイルシステムをマウントします。停止時には、パッケージはファイルシステムをアンマウントします。レガシーパッケージの場合、パッケージ制御スクリプトにマウント、アンマウント、およびfsck操作を実行するのに必要なファイルシステムコマンドが含まれています。モジュラーパッケージでは、これらのコマンドはパッケージ構成ファイル内に含まれています。RHCSの場合、ファイルシステムに対する1つ以上の論理ボリュームはサービスの開始時にアクティブ化され、ファイルシステムがマウントされます。停止時には、サービスがファイルシステムをアンマウントします。Red Hat Cluster Suiteの場合、ファイルシステムリソースエージェントに必要なコマンドやロジックが含まれています。

Serviceguardの場合、*fs_directory*、*fs_type*、*fs_mount_opt*、*fs_umount_opt*、および*fs_fsck_opt*と組み合わせた*fs_name*パラメーターにより、パッケージがマウントするファイルシステムを指定します。`fs_name`(レガシーパッケージでは*LV*パラメーター)は、論理ボリュームのブロックデバイスファイルを指定し、*fs_directory*パラメーターは*fs_name*が指定するファイルシステムのルートとなります。

パッケージを起動すると、パッケージ用に定義された各ファイルシステムはマウントされる前にファイルシステムチェック(`fsck`操作)を受けます。*fs_fsck_opt*は、操作で使用する`fsck`オプションを指定するものです。マウントポイントが使用中で*fs_mount_retry_count*がゼロより大きければ、起動スクリプトは使用中の原因であるユーザープロセスを停止しようと試みます。その後、再度ファイルシステムのマウントを実行します。この処理は、*fs_mount_retry_count*で指定されている回数だけ実行されます。*fs_mount_retry_count*で指定された回数後もマウントに失敗すると、パッケージは起動されません。マウント操作で使用するオプションは、*fs_mount_opt*で指定します。

パッケージのシャットダウン時は、*fs_name*で指定された各ファイルシステムはアンマウントされます。マウントポイントが使用中で*fs_unmount_retry_count*（レガシーパッケージのパラメーターは*FS_UMOUNT_COUNT*）がゼロより大きければ、シャットダウンスクリプトは使用中の原因であるユーザープロセスを停止しようと試みます。その後、再度ファイルシステムのアンマウントを実行します。この処理は、*fs_unmount_retry_count*で指定されている回数だけ実行されます。*fs_unmount_retry_count*で指定された回数後もアンマウントに失敗すると、パッケージはシャットダウンされません。*fs_umount_opt*は、アンマウント操作で使用するオプションを指定します。

RHCSでアプリケーションのファイルシステムを定義するには、ファイスシステムのリソースを定義する必要があります。マウントポイント、ファイルシステムタイプ、デバイスといったファイルシステムのリソースパラメーターは、リソースの*start*機能の一部としてマウントされるファイルシステムを指定します。パラメーターの*force_fsck*を設定すると、マウント前に、ファイルシステム上でファイルシステムチェックが実行されます。このオプションは、ext2などの非Journalファイルシステムでは無視されます。リソースのシャットダウン時には、「*stop*」機能jの一環でファイルシステムはアンマウントされます。マウントポイントが使用中であれば、*force_unmount*オプションはマウントポイントを使用するすべてのプロセスを停止し、ファイルシステムのアンマウントを強制実行します。パラメーターの*self_fence*を設定すると、ファイルシステムのアンマウントに失敗した場合、ノードをリブートします。

注：ファイルシステムのext2、ext3、およびgfsは、RHCSとSGLXの両方でサポートされます。

Red Hatのファイルシステムリソースと比較して、SGLXはパッケージに対して追加のファイルシステム機能を提供します。Red Hatファイルシステムリソースでサポートされていない、SGLXファイルシステムの機能一覧は次のとおりです。

1. *concurrent_fsck_operations*：パッケージ起動時にマウントされるファイルシステムで許されるfsckの同時実行数。

2. *concurrent_mount_and_umount_operations*：パッケージの起動時またはシャットダウン時に許されるマウントおよびアンマウントの同時実行数。

3. *fs_mount_retry_count*：各ファイルシステムに対するマウントのリトライ数。

4. *fs_umount_retry_count*：各ファイルシステムに対するアンマウントのリトライ数。

SGLXパッケージをRHCSへ移行する際の推奨事項は、次のとおりです。

- SGLXパッケージ内の各*fs_nameエントリー*に対して、対応するRHCSファイルシステムリソースを作成する必要があります。SGLXパッケージ構成ファイル内で定義されたfs_nameの順番で、リソースが/etc/cluster/cluster.confに定義されていることを確認してください。

- Red Hatファイルシステムリソースには*force_fsck*オプションを設定し、ファイルシステムをマウントする前にシステムチェックをファイルシステム上で実行するようにします。

- *self_fence*と組み合わせて*force_unmount*オプションを使用することで、アンマウントが失敗した場合はノードがリセットされるようにします。

移行手順

ここでは、Apache Webサーバーパッケージを例に移行手順について説明します。

移行するSGLXクラスター

この項では、RHCSクラスターに移行する既存のServiceguardクラスターについて説明します。設定は、2ノードのSGLX 11.19クラスターで、x86_64システムのノード1とノード2ではRHEL 5.3 Linuxオペレーティングシステムが稼動している構成です。クライアントアクセスにはNICのボンディングされたペア（Linuxチャネルボンディングを使用）を、クラスター通信にはスイッチに接続された単一NICを使用します。ネットワークパーティション時の調停（アービトレーション）には、クォーラムサーバーを使用します。

クラスターは、Apacheサーバーに高可用性を提供するモジュラーパッケージのapache_modで構成します。論理ボリュームの1vol01にはext2タイプのファイルシステムを作成します。これは、ディスクLUN（SCSI-3 PRを使用しているのでパーティションではない）を含むボリュームグループのvgap向けに作成されたもので、クラスター内のすべてのノードが参照できます。クラスター内の各ノードは、マウントポイントディレクトリの/u01を持っています。lvol01に作成されたファイルシステムは、apache_modパッケージが実行中のノードにマウントされます。

Apache Webサーバーは、Apacheパッケージを実行するよう構成されたすべてのクラスターノード（node1とnode2）上にインストールされます。*DocumentRoot*ディレクトリおよび*ServerRoot*ディレクトリは、（共有ファイルシステムをマウントする）/u01ディレクトリ内にあります。これにより、情報はどのクラスターメンバーからもアクセスすることができます。Apache構成ファイルのhttpd.confは、/u01/apache/httpd.confにあります。

DocumentRoot、*ServerRoot*、および*Listen*を示したApache構成の一部は、次のとおりです。

```
ServerRoot “/u01/apache/httpd.conf”
DocumentRoot "/u01/apache/doc"
Listen 80
```

apache_modパッケージのパッケージ構成パラメーターは、次のとおりです。

```
package_name                        Apache_mod
package_type                        failover
node_name                           node1
node_name                           node2
auto_run                            yes
node_fail_fast_enabled              no
run_script_timeout                  no_timeout
halt_script_timeout                 no_timeout
successor_halt_timeout              no_timeout
script_log_file                     /usr/local/cmcluster/apache/$SG_PACKAGE.log
operation_sequence                  $SGCONF/scripts/sg/volume_group.sh
operation_sequence                  $SGCONF/scripts/sg/filesystem.sh
operation_sequence                  $SGCONF/scripts/sg/package_ip.sh
operation_sequence                  $SGCONF/scripts/sg/external.sh
operation_sequence                  $SGCONF/scripts/sg/service.sh
#log_level
failover_policy                     configured_node
failback_policy                     manual
priority                            no_priority
service_name                Apache
service_cmd                 "/usr/local/cmcluster/apache/mon.sh"
service_restart                     3
service_fail_fast_enabled           no
service_halt_timeout                300
concurrent_fsck_operations          1
concurrent_mount_and_umount_operations  1
fs_mount_retry_count                0
fs_umount_retry_count               1
vgchange_cmd                        "vgchange -a y"
vg                                  vgap
```

```
external_script                    /usr/local/cmcluster/apache/extapache.sh
```

apache_modパッケージでは、ノード上のWebサーバーを開始および停止するための外部スクリプトファイルを使用できるよう構成されています。外部スクリプトのテンプレートに変更を加えると、次のようになります。

```
APACHE_CONF_FILE=/u01/apache/httpd.conf
APACHE_PID_FILE=/var/run/httpd.pid
APACHE_SERVER=/usr/sbin/httpd

function validate_command
{
    # Output messages will only be displayed in STDOUT, if there is
    # any error condition while executing the master control script
    # with a "validate" parameter from cmcheckconf/cmapplyconf command

    sg_log 5 "validate_command"
    # ADD your package validation steps here
       [ -x $APACHE_SERVER ] || return 1
       [ -f $APACHE_CONF_FILE ] || return 1
    return 0
}

start()
{
    sg_log 5 "start_command"
    # ADD your package start steps here
    $APACHE_SERVER -f $APACHE_CONF_FILE -c "PidFile $APACHE_PID_FILE";
    echo "Started the apache process \n";
    ps -ef | grep http| grep –v grep;
    return $?
}

stop()
{
    sg_log 5 "stop_command"
    # ADD your package halt steps here
    kill `cat $APACHE_PID_FILE`
    echo "Stopped the apache process\n"
    ps -ef | grep http| grep –v grep
    ret=$?
    if [ $ret -eq 0 ]; then
     return 1;
    else return 0;
     fi
}
```

計画フェーズ

このフェーズでは、移行への準備を行います。このフェーズ中、Serviceguardクラスターとそのパッケージはアップのまま実行されています。

追加要件

移行における追加のハードウェア要件

- *qdisk*構成として使用する1つの共有ディスク(LUN)

移行におけるソフトウェア要件

- Red Hat Cluster Suite
- ウォッチドッグrpm（RHEL 5.3にインストールされているrpmは`watchdog-5.3.1-7.el5.x86_64.rpm`）

SGLXクラスター構成の移行

ここでは、ノード名や障害検知タイムアウトといったServiceguardの「クラスターレベル」構成を、付録に付属する空白のクラスターワークシートに記録します。このワークシートは、RHCSクラスターを作成する際に使用します。

SGLXクラスターで使用する最新のクラスター構成値は、「`cmgetconf -c`」コマンドで検索します。システム上のクラスター構成ファイル（`/usr/local/cmcluster/sgcluster.conf`）を使用しないでください。構成ファイルがクラスターで実際に使用している値と同期していない可能性があるからです。この値を、クラスターワークシート内のSGLXパラメーターに反映させます。そして、次に示すように、対応するRHCSエントリーをワークシートに反映させます。

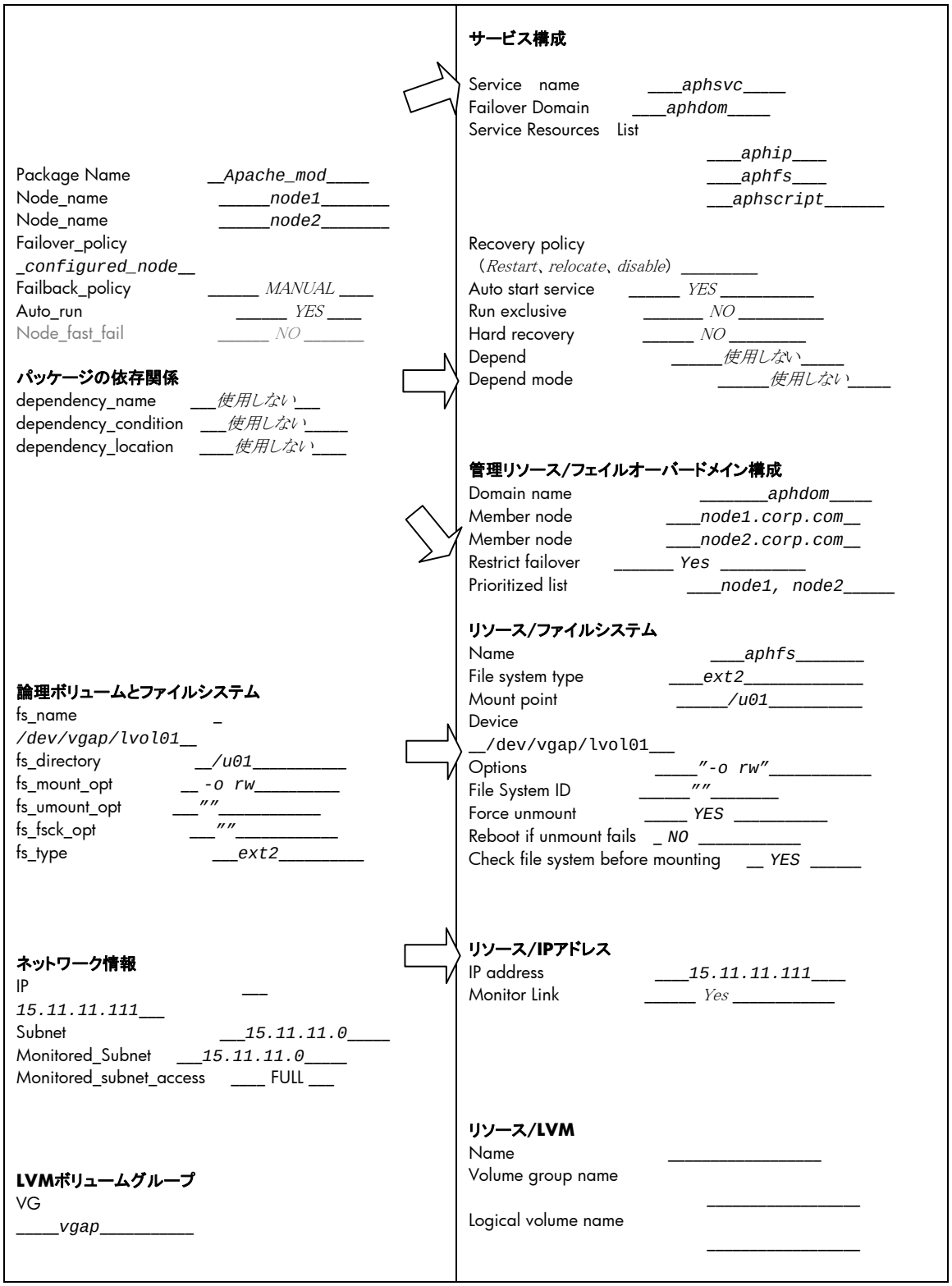
SGLXクラスター構成パラメーター	RHCS構成パラメーター
Cluster name ___*sgcluster*_____ Node name _____*node1*_____ Node name _____*node2*_____	Cluster name ___*rhcluster*_____ Node name ___*node1.corp.com*___ ___*node2.corp.com*___
	メンバー構成 Node name ___*node1.corp.com*___ Fence type * ___*fence_scsi*_____ Node name ___*node2.corp.com*____ Fence type * ____*fence_scsi*_____
SGLX 11.19 Member Timeout _____14秒___	Totem token timeout ____*20秒*____
	フェンスデバイス*構成 Fence method ____*fence_scsi*___ Fence parameters ____なし_____
	Qdisk*構成 * - これは、SGLXクラスターの移行で推奨される2つの「使い慣れた」RHCS構成をベースにしたものです

パッケージ構成の移行

パッケージに関連する構成はすべて、付録に付属する空白のパッケージワークシートに記録します。移行するパッケージごとに別のワークシートを使用してください。この情報は、RHCSでクラスターリソースとサービスを作成するときに使用します。

「cmgetconf –p apache_mod」コマンドを使って、パッケージで使用されている最新のパッケージ構成を検索します。パッケージ構成ファイルは、パッケージの実際の値と同期していない可能性があるので、使用しないでください。検索した値をパッケージワークシート内のSGLXパラメーターに反映させます。そして、次に示すように、対応するRHCSエントリーをワークシートに反映させます。

SGLXパッケージ構成パラメーター　　RHCSサービスおよびリソース構成パラメーター



Package Name __Apache_mod____
Node_name _____node1_____
Node_name _____node2_____
Failover_policy _configured_node__
Failback_policy _____ MANUAL ____
Auto_run _____ YES ____
Node_fast_fail _____ NO _____

パッケージの依存関係
dependency_name ___使用しない___
dependency_condition ___使用しない____
dependency_location ____使用しない____

論理ボリュームとファイルシステム
fs_name _ /dev/vgap/lvol01__
fs_directory __/u01_____
fs_mount_opt __ -o rw_____
fs_umount_opt ___""_____
fs_fsck_opt ___""_____
fs_type ___ext2_____

ネットワーク情報
IP ___ 15.11.11.111___
Subnet ___15.11.11.0____
Monitored_Subnet ___15.11.11.0____
Monitored_subnet_access ____ FULL ___

LVMボリュームグループ
VG ____vgap_____

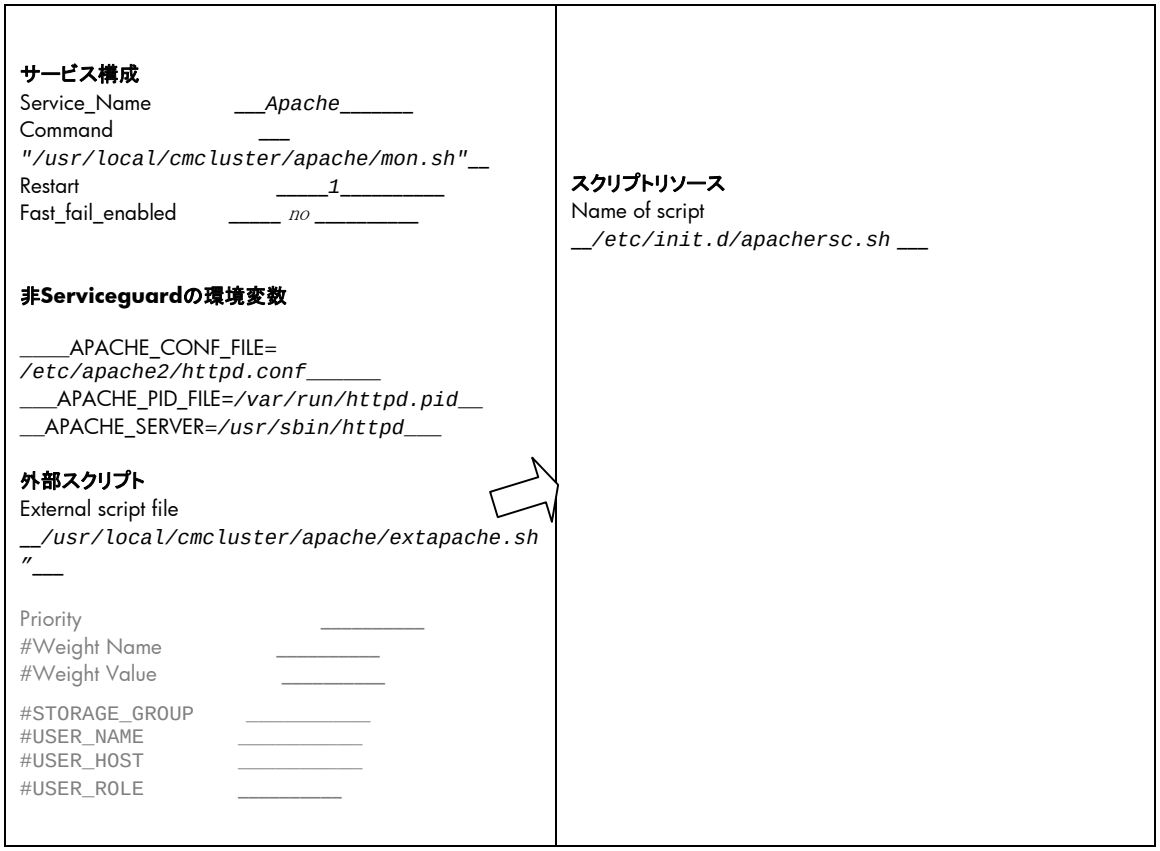
サービス構成
Service name ____aphsvc____
Failover Domain ____aphdom____
Service Resources List
____aphip____
____aphfs____
___aphscript_____
Recovery policy （Restart、relocate、disable） _____
Auto start service _____ YES _____
Run exclusive _____ NO _____
Hard recovery _____ NO _____
Depend _____使用しない____
Depend mode _____使用しない____

管理リソース/フェイルオーバードメイン構成
Domain name _____aphdom____
Member node ____node1.corp.com__
Member node ____node2.corp.com__
Restrict failover _____ Yes _____
Prioritized list ____node1, node2_____

リソース/ファイルシステム
Name ____aphfs_____
File system type ____ext2_____
Mount point _____/u01_____
Device __/dev/vgap/lvol01___
Options ____"-o rw"_____
File System ID _____""_____
Force unmount _____ YES _____
Reboot if unmount fails _ NO _____
Check file system before mounting __ YES _____

リソース/IPアドレス
IP address ____15.11.11.111___
Monitor Link _____ Yes _____

リソース/LVM
Name _____
Volume group name _____
Logical volume name _____



顧客定義領域および外部スクリプト変数の保存

ここでは、Apache Webサーバーの HAを有効にする、SGLXで使用した/usr/local/cmcluster/apache/extapache.sh外部スクリプトをRHCS「スクリプト」リソースとして移植します。

注: この例では、すでにApache Webサーバー向けに構成された「apache」リソースが提供されているので、外部スクリプトを「スクリプト」リソースとして移植しなくてもかまいません。scriptリソースにexternal_scriptを移植する理由は、同等のリソースエージェントを持たないアプリケーションをRHCSに移行する手順を示すためです。

まず、/usr/local/cmcluster/apache/extapache.shファイル内の*start*関数および*stop*関数を、「スクリプト」リソースファイルの/etc/init.d/apachersc.sh内にある*start*関数と*stop*関数へ移植します。両関数を移植したら、次にServiceguardの*SERVICE_CMD*スクリプトのロジックを*script*リソースファイルの*monitor*関数へ移植します。

その次に、external_scriptファイル内にある非Serviceguardの変数のほか、SGLXの*SERVICE_CMD*スクリプト内にある変数をすべて/etc/init.d/apachersc.shファイルへ移動します。

注：外部スクリプトファイルが使用するパッケージ構成ファイル内の非Serviceguard変数であっても、「スクリプト」リソースファイルへ移植する必要があります。

移植プロセス中は、LSB仕様に準拠してください。

注：（LSB仕様ベースの）*script*リソースには外部スクリプトの*validate_command*関数と同等のものがないので、*validate_command*関数はスクリプトリソースファイルに移植しません。

次に、/etc/init.d/apachersc.shのリストを示します。

```
# Resource that starts, stops, and monitors the apache web server

# Variable required the agent
APACHE_CONF_FILE=/u01/apache/httpd.conf
APACHE_PID_FILE=/var/run/httpd.pid
APACHE_SERVER=/usr/sbin/httpd

# starts the apache web server
start()
{
    echo -n "Starting the apache script resource: "
    $APACHE_SERVER -f $APACHE_CONF_FILE -c "PidFile $APACHE_PID_FILE";
    echo "Started the apache process \n";
    ps -ef | grep http| grep –v grep;
    return $?
}

# stops the apache web server
stop()
{
    echo -n "Stopping the apache script resource: "
    kill `cat $APACHE_PID_FILE`
    echo "Stopped the apache process\n"
    ps -ef | grep http| grep –v grep
    return $?
}

# Monitors the apache web server
monitor()
{
    echo -n "Monitoring the apache script resource: "
    if [ ! -f $APACHE_PID_FILE ]; then
        return 1
    else
        PARENT_PID=`cat $APACHE_PIDFILE`
        grep httpd /proc/$PARENT_PID/stat >/dev/null 2>&1
    fi
    if [ $? != 1 ]; then
        return 0
    else
        return 1
    fi
}

# main
  … …
  … …
```

アプリケーションと構成データのバックアップ

RHCSに移行する前に、アプリケーションデータの最新コピーをバックアップすることをおすすめします。これは、移行時の操作エラーでデータ消失を防ぐためです。データのバックアップには、RHEL5対応のバックアップソフトウェアを使用します。

移行フェーズ

このフェーズでは、「SCSI-3 PRフェンシングに基づく「使い慣れた」構成」を設定し、パッケージをRHCSに移行します。始める前に、クラスターアプリケーションの可用性の影響をなくすため、Serviceguardクラスターとすべてのパッケージをシャットダウンします。このフェーズが完了するまで、クライアントはアプリケーションにアクセスできません。

Serviceguardクラスターとパッケージの停止

Red Hat Cluster Suiteをインストールする前に、Serviceguardクラスターとアプリケーションパッケージを停止させます。次のコマンドで、アプリケーションパッケージとクラスターを停止させてください。

```
$> cmhaltpkg apache_mod
$> cmhaltcl
```

RHCS RPMのインストール

Red Hat Cluster Suiteのrpmをすべてのクラスターノード上にインストールします。

注： クラスターノードにインストールされたApache Webサーバーを再インストールしないでください。SGLXクラスターのアプリケーション構成は、RHCSクラスターでも継続して使用します。

ボリュームグループをCLVMに変換

*/etc/lvm/lvm.conf*を編集して、*locking_type*のパラメーターを3に設定します。クラスター化されていないSGLXボリュームグループvgapをクラスター化されたボリュームグループへ変換するには、「vgchange -c y vgap」コマンドを実行します。

RHCSメンバー、SCSI-3 PRフェンシング、およびqdiskの構成

この項では、Serviceguardクラスター構成から構成を取得し、RHCSクラスターを構成する方法について説明します。

注： 設定には、Red Hat対応の構成ユーティリティを使用してください。詳しくは、RedHat社のWebサイトhttp://www.redhat.com/docs/manuals/csgfsの「Configuring and Managing a Red Hat Cluster」ドキュメントを参照してください。

「*fdisk -l*」コマンドを使って構成ディスクのリストを表示し、未使用の共有ディスクをqdiskとして選択します。

次に、共有ディスク（ここでは/dev/sdc1）を次のようにqdiskとして初期化します。

```
$> mkqdisk -c /dev/sdc1 -l qdisksgrh
```

Red Hatドキュメントの「Configuring and Managing an RHCS cluster」にあるとおり、SCSI-3 PRをフェンシング方式として使ってnode1とnode2の2ノードクラスターを設定します。

クラスター構成の際には*qdisk*オプションを選択し、次のように構成します。

```
<totem token="20000"/>

<quorumd interval="2" label="qdisksgrh" tko="5" votes="1">
   <heuristic program="/usr/local/sbin/checkpvtlink.sh eth1"
      score="1" interval="2" tko="3"/>
</quorumd>
```

Red Hatで推奨されているとおり、totem tokenは少なくとも*qdiskd*メンバーシップのタイムアウト値に対して2倍の20秒に設定します。

スクリプトの/usr/local/sbin/checkpvtlink.shは、次のように実装します。

```
#!/bin/sh

#
# Check for link state
#
ethtool $1 | grep -q "Link detected.*yes"
exit $?
```

構成が完了したら、/etc/cluster/cluster.confファイルは次のようになります。

```
<cluster alias="rhcluster" config_version="1" name="rhcluster">

   <fence_daemon clean_start="0" post_fail_delay="0"
        post_join_delay="3"/>
   <clusternodes>
      <clusternode name="node1.corp.com" nodeid="1" votes="1">
      <fence>
            <method name="1">
            <device name="scsi-pr" node="node1.corp.com"/>
            </method>
       </fence>
       </clusternode>

      <clusternode name="node2.corp.com" nodeid="2" votes="1">
       <fence>
            <method name="1">
            <device name="scsi-pr" node="node2.corp.com"/>
            </method>
        </fence>
        </clusternode>
   </clusternodes>
   <cman expected_votes="2" two_node="0"/>

   <fencedevices>
      <fencedevice agent="fence_scsi" name="scsi-pr"/>
   </fencedevices>
  <totem token="20000"/>
  <quorumd interval="2" label="qdisksgrh" tko="5" votes="1">
    <heuristic program="/usr/local/sbin/checkpvtlink.sh eth1"
      score="1" interval="2" tko="3"/>
  </quorumd>

</cluster>
```

自動再起動方式の構成

この項では、すでに*fence_scsi*によってフェンシングされたノードを自動で再起動する方法を設定します。詳しくは、「SCSI-3 PRフェンシングに基づく「使い慣れた」構成」の項を参照してください。これを実行するため、各ノードにウォッチドッグデーモン（watchdogd）を設定し、クラスター内の各ノードにあるRHCSサービスとして管理します。

node1とnode2の両方に対してwatchdogd構成ファイルの/etc/watchdog.confを編集し、パラメーターのみを次のとおり設定します。

```
file                = /var/log/messages
test-binary          = /root/RHCS-WDT/check_reservation.sh
watchdog-device = /dev/watchdog
realtime             = yes
priority             = 1
```

クラスター内のノードごとに*restricted*フェイルオーバードメインを作成します。node1にはnode1-domainを、node2にはnode2-domainを作成してください。

ノードのウォッチドッグデーモンを管理(開始、停止、および監視)する共有スクリプトリソースの*wdtcscript*を作成します。次に、共有リソースのwdtscriptを持つwdtcsrv-node1サービスを、node1-domainのフェイルオーバードメインで作成します。これにより、サービスはnode1でのみ実行されるようになります。同様に、wdtcscript共有リソースを持つwdtcsrv-node2サービスを、node2-domainのフェイルオーバードメインで作成します。

構成が完了したら、/etc/cluster/cluster.confファイルには次のようなXML定義が記述されます。

```
<cluster alias="rhcluster" config_version="1" name="rhcluster">
   . . .
   . . .
<rm>
   <failoverdomains>
     <failoverdomain name="node1" restricted="1">
       <failoverdomainnode name="node1-domain"/>
     </failoverdomain>
     <failoverdomain name="node2" restricted="1">
       <failoverdomainnode name="node2-domain"/>
     </failoverdomain>
   </failoverdomains>
   <resources>
     <script name="wdtcscript" file="/etc/init.d/wdtcscript.sh" />
   </resources>
   <service name="wdtcsrv-gf1" domain="node1-domain" >
     <script ref="wdtcscript" />
   </service>
   <service name="wdtcsrv-gf2" domain="node2-domain" >
     <script ref="wdtcscript" />
   </service>
 </rm>
</cluster>
```

アプリケーションのフェイルオーバードメイン、リソース、およびサービスの設定

Serviceguardパッケージのapache_modの移行に必要なフェイルオーバードメインを設定します。ドメインとプロパティのリストは、計画フェーズで用意したパッケージ構成ワークシートを参照してください。

パッケージ構成ワークシートでは、aphdomがapache_modパッケージのフェイルオーバードメインとして認識されています。ドメインメンバーはnode1とnode2で、タイプは「restricted、ordered」です。RHCS構成ユーティリティを使って、aphdomドメインを作成します。

次に、クラスターサービスのaphsvcと、Serviceguardパッケージのapache_modの移行で必要となるリソースを設定します。サービスの詳細や子リソースについては、サービスの計画フェーズで用意したパッケージ構成ワークシートを参照してください。

フェイルオーバードメイン、リソース、およびApacheのサービスを構成したら、次のXML定義が/etc/cluster/cluster.confファイル内の<rm>タグの下にあるか確認します。

```
<rm>
  <failoverdomains>
   <failoverdomain name="aphdom" nofailback="1" ordered="1" restricted="1">
      <failoverdomainnode name="node1.corp.com" priority="1"/>
      <failoverdomainnode name="node2.corp.com" priority="2"/>
   </failoverdomain>
  </failoverdomains>

  <service autostart="1" domain="aphdom" exclusive="0"
       name="aphsvc" recovery="restart">
   <fs device="/dev/vgap/lvol01" force_fsck="1" force_unmount="1" fsid="10453"
       fstype="ext2" mountpoint="/u01" name="aphfs" options="-o rw"
       self_fence="0"/>
   <ip address="15.11.11.111" monitor_link="0"/>
   <script file="/etc/init.d/apachersc.sh" name="aphscript"/>
  </service>
</rm>
```

クラスターおよびサービスの開始

システムのブート時に、cman、clvmd、qdiskd、scsi_reserve、およびrgmanagerのサービスの開始を有効にします。システムのブート時にwatchdogdサービスが起動しないよう無効化します。

```
$> chkconfig cman on
$> chkconfig clvmd on
$> chkconfig qdiskd on
$> chkconfig scsi_reserve on
$> chkconfig rgmanager on
$> chkconfig watchdogd off
```

ウォッチドッグ・コントローラー・クラスター・サービスであるnode1上のwdtscrv-node、およびnode2上のwdtcsrv-node2を有効にします。

```
On node1 execute the command as follows:
$> clusvcadm –e wdtcsrv-node1

On node2 execute the command as follows:
$> clusvcadm –e wdtcsrv-node2
```

node1とnode2のApacheサービス、aphsvcを有効にします。

```
On node1 and node2 execute the following commands:
$> clusvcadm –e aphsvc
```

次のクラスターサービスを開始します。

```
$> service cman start
$> service clvmd start
$> service qdiskd start
$> service scsi_reserve start
$> service rgmanager start
```

*clustat*コマンドを使って、aphsvcとwdtcsrv-node1のサービスがnode1でアップされているか、またwdtcsrv-node2がnode2でアップされているかを確認します。

用語

ASIC
: Application Specific Integrated Circuit。iLOで使用されるリモート管理用のハードウェア

APC
: American Power Conversion Corp

CMAN
: Cluster Manager。クラスターのクォーラムを管理し、クラスターメンバーシップを保持

CLVMD
: Cluster LVM Daemon。クラスター環境でLVMメタデータを同期

DLM
: Distributed Lock Manager。ロック管理義務をクラスター内のすべてのノードへ配信

DRAC
: Dell Remote Access Controller。Dellサーバーに帯域外管理を提供するインターフェイスカード

DM-MPIO
: Device MapperMultipathドライバー

FC
: ファイバーチャネル

GFS
: Red Hat Global File System。ノード間で共有されるブロックデバイスへクラスターノードが同時アクセスできるようにするクラスターファイルシステム

HA
: 高可用性

iLO
: HP Integrated Lights-Out

LVM
: Linuxの論理ボリューム管理機能

LVM2
: Linuxで論理ボリューム管理機能を提供する、新規のユーザー・スペース・ツール・セット

LAN
: ローカルエリアネットワーク

LSB
: Linux Standard Base

NIC
: ネットワークインターフェイスカード

NMIウォッチドッグ
: 非マスク割り込みウォッチドッグ。特定のサーバーハードウェアに依存し、システムハングを検知する

OCF
: オープンクラスターフレームワーク。RHCSのリソースエージェント統合のためのインターフェイス

PR
Persistent Reservation。SCSI-3コマンドのセットで、複数のシステムによる共有デバイスへのアクセスを許可しながら、その他からのアクセスを制御

Qdisk
RHCSクラスター向けのディスクベースのクォーラム

Qdiskd
クラスター・クォーラム・ディスク・デーモン。クラスター環境におけるノードの適合具合の判定方法を提供

RA
リソースエージェント（RA）。指定されたリソースへの操作を扱うスクリプトまたは実行可能プログラム

RHCS
Red Hat Cluster Suite。導入することで、パフォーマンス、高可用性、負荷分散、拡張性、ファイル共有、経済性などのニーズを満たすことができる、ソフトウェアコンポーネントの統合セット

RIBCL
リモートInsightボードコマンド言語。XMLベースの独自言語で、iLOの管理で使用

RHEL5
Red Hat Enterprise Linux 5

RSA
リモート管理アダプター。IBMシステムにおける包括的なサーバー管理を提供

SAN
ストレージエリアネットワーク

SCSI
Small Computer System Interface。コンピューターシステムと周辺機器間のデータ転送のための標準規格

WTI
Western Telematic Inc.

WDT
ウォッチドッグタイマー。システムハングを検知および管理

XML
拡張マークアップ言語。カスタムのマークアップ言語を作成するための汎用な仕様

モジュラーパッケージ
Serviceguard A.11.18で登場した単一のパッケージ構成ファイル。パッケージ構成情報はパッケージ構成（ASCII）ファイルにのみ含まれている。11.18より前のパッケージ構成情報は、パッケージASCIIファイルとパッケージ制御スクリプトの両方に含まれている

レガシーパッケージ
11.18より前のパッケージ構成

詳細情報

- HP Serviceguard for Linux製品のドキュメント：HPのWebサイトhttp://docs.hp.com（英語）
- Red Hat Cluster Suite for RHEL5および構成ガイド：RedHat社のWebサイトhttp://www.redhat.com/docs/manuals/csgfs
- サーバー、ストレージ、およびソフトウェアの対応バージョンを示すHP Serviceguard for Linux認定マトリックス：HPのWebサイトhttp://www.hp.com/info/sglx（英語）
- RHCSの追加情報については、RedHat社のWebサイトhttp://sources.redhat.com/cluster/wikiを参照してください。ただし、アップストリームにある機能は完全に正式リリースへ実装されるわけではないので、wikiの情報を参照する際はその点にご留意ください。

付録

ウォッチドッグ・コントローラー・サービス・スクリプトのサンプル

```
#! /bin/sh
#
#  resource that start, stop, and monitors watchdog service
#
#
# Global variables

watchdog_dev_file="/dev/watchdog"
wdog_dev_maj_no="10"
wdog_dev_min_no="130"
wdog_init_script="/etc/init.d/watchdog"
cluster_details_cache_file="/tmp/cluster_cache"

# Function to retrieve the cluster id
# from cman
cache_cluster_id()
{
   cluster_id=$( cman_tool status | grep -i "Cluster ID" \
           | awk -F": " '{ print $2 }' )

   if [ -z "$cluster_id" ] ; then
      echo -e "ERROR: Unable to determine cluster id"
      return 1
   else
      echo "ClusterID:$cluster_id" > $cluster_details_cache_file
   fi
}

# Function to get the node id from cman
cache_node_id()
{
   node_id=$( cman_tool status | grep -i "Node ID" \
         | awk -F": " '{ print $2 }' )
   if [ -z "$node_id" ] ; then
      echo -e "ERROR: unable to determine node id"
      return 1
   else
      echo "NodeID:$node_id" >> $cluster_details_cache_file
   fi
}

# Function to start watchdog service
start()
{
    typeset -i retval=0

    # Store cluster id in cache file
    cache_cluster_id

    # Store node id in cache file
    cache_node_id

    # Check if the watchdog device file exists
    if [ ! -c $watchdog_dev_file ]; then
       echo -e "Creating watchdog device file \"$watchdog_dev_file\""
       mknod /dev/watchdog c $wdog_dev_maj_no $wdog_dev_min_no
       if (( $? != 0 )); then
          echo   -e   "ERROR   :   Failed   to   create   watchdog   device   file
\"$watchdog_dev_file\""
          return 1
       fi
    fi
```

```
   sleep 5

   # Start the watchdog timer service
   echo -n "Starting watchdog service: "
   if [ -f "$wdog_init_script" ]; then
      # Start the watchdog service
      $wdog_init_script "start"
      if (( $? != 0 )); then
         echo -e "ERROR : Failed to start the watchdog service"
         return 1
      fi
   fi
   return 0
}

# Function to stop watchdog service
stop()
{
   typeset -i retval=0

   # Start the watchdog timer service
   echo -n "Stopping watchdog service: "
   if [ -f "$wdog_init_script" ]; then
      # Stop the watchdog service
      $wdog_init_script "stop"
      if (( $? != 0 )); then
         echo -e "ERROR : Failed to stop the watchdog service"
         retval=1
      fi
   fi
   return $retval
}

# Function to restart watchdog service
restart() {
   typeset -i retval=0
   # Start the watchdog timer service
   echo -n "Restarting watchdog service: "
   if [ -f "$wdog_init_script" ]; then
      # Restart the watchdog service
      $wdog_init_script "restart"
      if (( $? != 0 )); then
         echo -e "ERROR : Failed to restart the watchdog service"
         retval=1
      fi
   fi

   return $retval
}

# Function to get status of watchdog service
status()
{
   typeset wdog_status=""
   typeset -i retval=0

   wdog_status=$($wdog_init_script status)
   echo $wdog_status | grep -i -q "watchdog.*is.*running"
   if (( $? == 0 )); then
      retval=0
   else
      retval=1
   fi

   return $retval
}
```

```
typeset -i ret_val=0
case $1 in
  start )
       start
       ret_val=$?
       ;;
  stop )
       stop
       ret_val=$?
       ;;
  restart )
       restart
       ret_val=$?
       ;;
  status )
       status
       ret_val=$?
       ;;
  * )
       echo "Usage: $0 {start|stop|retsart|status}"
       exit 1
       ;;
esac

exit $ret_val
```

ウォッチドッグのコールバックフェンシング検証スクリプトのサンプル

```
#!/bin/bash

# This script is used by the watchdog timer to check for SCSI reservations
# If reservation of the current node are not found on any of the shared LUN's
# the watchdog timer reboots the node thinking that it has been fenced.

# Global Variables

typeset scsi_devices=""
typeset cluster_id=""
typeset node_id=""
typeset key=""
typeset cluster_cache_file="/tmp/cluster_cache"

# Function to get the get the physical
# volumes from the cluster logical volumes
get_scsi_devices()
{
   scsi_devices=$( vgs --config 'global { locking_type = 0 }' \
                  --noheadings -o vg_attr,pv_name 2> /dev/null \
            | awk ' $1 ~ /.*c$/ { print $2 } ' )

   if [ -z "$scsi_devices" ] ; then
      logger -t watchdog \
        "[error] did not find devices in cluster volumes"
      return 1
   fi
}

# Function to retrieve the cluster id
# from cluster cache file. If the file is
# not present then the cluster id is retrieved
# from "cman" and cached to the file
get_cluster_id()
{
   if [ -f "$cluster_cache_file" ]; then
      cluster_id=$( cat $cluster_cache_file | grep -i "ClusterID" \
                | awk -F":" '{ print $2 }' )
   else
      cluster_id=$( cman_tool status | grep -i "Cluster ID" \
                | awk -F": " '{ print $2 }' )

      # Store the contents in the cluster_cache file
      echo "ClusterID:$cluster_id" > $cluster_cache_file
   fi

   if [ -z "$cluster_id" ] ; then
      logger -s -t watchdog \
        "[error] unable to determine cluster id"
      return 1
   fi
}

# Function to get the node id from cman
get_node_id()
{
   if [ -f "$cluster_cache_file" ]; then
      node_id=$( cat $cluster_cache_file | grep -i "NodeID" \
             | awk -F":" '{ print $2 }' )
   else
      node_id=$( cman_tool status | grep -i "Node ID" \
             | awk -F": " '{ print $2 }' )

      # Store the contents in the cluster_cache file
```

```
      echo "NodeID:$node_id" >> $cluster_cache_file
   fi
   if [ -z "$node_id" ] ; then
      logger -t watchdog \
        "[error] unable to determine node id"
      return 1
   fi
}

# Function to generate the unique reservation key
# for the current node.
generate_unique_res_key()
{
   key=$( printf "%x%.4x" $cluster_id $node_id )

   if [ -z "$key" ] ; then
      logger -t watchdog \
        "[error] unable to generate key"
      return 1
   fi
}

# Function to check whether the current node has reserved
# any device. (shared physical volume)
check_scsi_res()
{
   typeset -i error=0

   for dev in $scsi_devices; do
      if sg_persist -n -d $dev -i -k | grep -qiE "$key" ; then
         echo "Found Key \"$key\" on device \"$dev\""
         devices[${#devices[@]}]=$dev
      fi
   done
   if [ -z "$devices" ] ; then
      echo "No registered devices found."
      return 1
   else
      return 0
   fi
}

typeset exit_val=0

# Script main
get_scsi_devices
if (( $? == 0 )); then
   get_cluster_id
   if (( $? == 0 )); then
      get_node_id
      if (( $? == 0 )); then
         generate_unique_res_key
         if (( $? == 0 )); then
            check_scsi_res
            if (( $? != 0 )); then
               exit_val=1
            else
               exit_val=0
            fi
         fi
      fi
   fi
fi
exit $exit_val
```

移行計画ワークシート

クラスター構成ワークシート

SGLXクラスター構成パラメーター	RHCS構成パラメーター
Cluster name _____ Node name _____ Node name _____ Serviceguard 11.19 Member Timeout _____	Cluster name _____ Node name _____ _____ **メンバー構成** Node name _____ Fence type * ____ _____ Node name _____ Fence type * ____ _____ Totem token timeout _____ **フェンスデバイス*構成** Fence method _____ Fence parameters _____ **Qdisk*構成** * - これは、SGLXクラスターの移行で推奨される2つの「使い慣れた」RHCS構成をベースにしたものです

SGLXパッケージ構成ワークシート

SGLXパッケージ構成	RHCSクラスターサービス、リソース、およびフェイルオーバードメイン構成
Package Name _____ Node_name _____ Node_name _____ Failover_policy *_configured_node__* Failback_policy _____ Auto_run _____ Node_fast_fail ___使用しない__ **パッケージの依存関係** dependency_name ___使用しない___ dependency_condition ___使用しない_____ dependency_location ____使用しない____ **論理ボリュームとファイルシステム** fs_name _____ fs_directory _____ fs_mount_opt _____ fs_umount_opt _____ fs_fsck_opt _____ fs_type _____ **ネットワーク情報** IP _____ Subnet _____ Monitored_Subnet _____ Monitored_subnet_access _____ LVMボリュームグループ VG _____	**サービス構成** Service name _____ Failover Domain _____ Service Resources List _____ _____ _____ Recovery policy （Restart、relocate、disable） _____ Auto start service _____ Run exclusive _____NO_____ Hard recovery _____NO_____ Depend _____ Depend mode _____ **管理リソース/フェイルオーバードメイン構成** Domain name _____ Member node _____ Member node _____ Restrict failover _____ Prioritized list _____ **リソース/ファイルシステム** Name _____ File system type _____ Mount point _____ Device _____ Options _____ File System ID _____ Force unmount _____ Reboot if unmount fails _____ Check file system before mounting _____ **リソース/IPアドレス** IP address _____ Monitor Link _____ **リソース/LVM** Name _____ Volume group name _____ Logical volume name _____

サービス構成
Service_Name _____
Command _____
Restart _____
Fast_fail_enabled _____

非Serviceguardの環境変数
_____
_____
_____
_____

外部スクリプト
External script file _____

Priority _____
#Weight Name _____
#Weight Value _____

#STORAGE_GROUP _____
#USER_NAME _____
#USER_HOST _____
#USER_ROLE _____

スクリプトリソース
Name of script _____

4AA1-xxxxENN、2009年5月