Pacemaker 設定の手引き
日本語版

著者: Andrew Beekhof
日本語訳: **久保 元治**

目次

図目次

第 1 章

はじめに

1.1 本書で説明すること

本書の目的は、Pacemaker の設定に必要な概念を詳しく説明することです。この目的を達成するために、CIB で使われている XML 構文を重点的に説明します。

XML アレルギーの方向けに、クラスタシェルと Python ベースの GUI も用意してあります。しかし、これらのツールは XML を隠蔽するので、本書では取り扱いません[*1]

なお、本書は、特定のクラスタ構築シナリオにもとづいてステップごとに設定方法を解説する、というスタイルにはなっていません。代わりに、あらゆるタイプの Pacemaker クラスタを構築できるようにするために、Pacemaker の構成要素を理解していただくことが本書の目的です。

1.2 Pacemaker とは

Pacemaker はクラスタシステムのリソースを管理するプログラムです。その目的は、ノードやサービス (リソース) の障害を検出したり修復して、クラスタシステムの可用性を最大に高めることです。この目的を達成するために、OpenAIS または Heartbeat が提供するメッセージングやメンバ管理機能を利用します。

Pacemaker は以下のような基本的な機能を備えています。

- ノードあるいはサービスレベルの障害の検出
- ストレージに関する前提要件を持たない。共有ストレージすら不可欠な構成要素としない。
- リソースに関する前提要件を持たない。スクリプト化できるならなんでもクラスタ化できる。
- 小規模から大規模までさまざまなクラスタを実現できる。
- STONITH を活用してデータ整合性を保証できる。
- クラスタ全体にまたがるサービスの順序づけ、複数サービスの相互依存性や排他性を管理できる。
- 複数のノードでアクティブにならなければならないサービスもサポートする。
- 複数の動作モード (たとえばマスター/スレーブやプライマリ/セカンダリ) を持つサービスもサポートする。
- さまざまなクラスタ管理ニーズを統合したスクリプト化可能なシェルの提供。

[*1] しかし、本書に説明した概念を理解しておくことによって、これらのツールが提供する機能の理解も早まります。

1.3 Pacemaker で実現できるクラスタのタイプ

Pacemaker は環境要件を持ちません。このため、アクティブ/アクティブ、アクティブ/パッシブ、N+1、N+M、N 対 1、N 対 N といったさまざまな形式の冗長化をサポートします。

1.3.1 2 ノードアクティブ/パッシブクラスタ

Pacemaker と DRBD を組み合わせたこの構成は、多くの高可用性ニーズに低コストで対応できるシェアードナッシングソリューションになります。

1.3.2 N 対 1 クラスタ

複数ノードを組み合わせてその 1 台をバックアップノードにすれば、複数のアクティブ/パッシブクラスタセットを構成するのと比べてハードウェアコストを大きく節約できます。

1.3.3 N 対 N クラスタ

共有ストレージが利用できるなら、Pacemaker クラスタのすべてのノードは互いにフェールオーバ先として設定できます。サービスの特性によっては、Pacemaker を負荷分散の基盤としても利用できます。

1.4 Pacemaker のアーキテクチャ

クラスタは 3 つの要素で構成されます。

- ノード間のメッセージングとメンバ管理を受け持つ基盤層
- クラスタの状態に関係なく動作するコンポーネント群。この層には、起動/停止などの動作を実行するスクリプト群と、数タイプあるスクリプト実装方法の違いを吸収するデーモンが含まれる。
- クラスタの頭脳に相当する層。クラスタリソースマネージャは、ノードがクラスタに参加したりクラスタから切り離されたこと、リソースの監視結果、管理者による設定の変更などのイベントをキャッチして処理する。イベントにもとづいて、クラスタ全体の最適な状態を維持するための遷移パスを計算して実行を指示する。遷移には、リソースの移動、ノードの停止、ときには電源スイッチを制御してノードを強制的にオフラインにすること、などが含まれる。

[cluster-stack-heartbeat.png not found]

図 1.1 クラスタスタックの構成

OpenAIS と Pacemaker を組み合わせると、オープンソースのクラスタファイルシステムを活用できます。クラスタファイルシステムによる最近の標準化によって、共通の分散ロックマネージャが利用可能になりました。OpenAIS によるメッセージング、メンバシップ管理、フェンシングにこれを活用できます。

[cluster-stack-heartbeat.png not found]

図 1.2　クラスタスタックの構成 (OpenAIS)

1.5　Pacemaker の内部コンポーネント

Pacemaker は 4 つのキーコンポーネントで構成されています。

- CIB (クラスタインフォメーションベース)
- CRM デーモン (クラスタリソースマネージャ)
- PEngine (ポリシーエンジン)
- STONITH デーモン

CIB は XML 形式のデータ構造を使ってクラスタ全体の構成と現在の状態を記録します。CIB の内容は、クラスタの全ノードに自動的に配布されて同期がはかられます。PEngine はクラスタの最適な状態とそこに至るための必要な処理を計算します。

クラスタ全体を統一的に管理するために、Pacemaker はどれか 1 つの CRMd を選出して DC (Designated Coordinater) にします。PEngine の計算結果は DC に引き渡されます。なお、DC になった CRMd が停止したり DC ノードがダウンすると、新しい DC がただちに選出されます。

DC は PEngine から受け取った指令を順序どおりにローカルノードの LRMd (ローカルリソースマネージャ) もしくは他ノードの CRMd に伝達します。この伝達にはクラスタのメッセージング基盤が使われます。そして他ノードでは CRMd から LRMd に指令が伝えられます。

他ノードは指令の処理結果を DC に報告します。また、期待される結果と実際の結果の比較にもとづいて、以前の処理結果が完了するために必要なアクションを起こすか、処理を中断します。中断した場合は、予期しなかった結果を踏まえた新しい戦略立案を PEngine に依頼しま。

共有データの保護やリソースのリカバリのためにどれかのノードの電源を OFF にするしかない場合もありえます。このような場合、Pacemaker は STONITHd を活用します。STONITH とは Shoot-The-Other-Node-In-The-Head の略語で「相手を頭越しに叩く」というイメージを表しています。通常これは、リモートマシンの電源スイッチを制御するハードウェア等を使って実現します。監視により障害が見つかった場合に STONITH 機能を利用しやすくするために、Pacemaker は STONITH デバイスもリソースとして取り扱い、したがって CIB で管理します。ただし STONITHd はクラスタ全体の STONITH 機構のトポロジーを理解している独立したサーバプログラムで、そのクライアントである Pacemaker は、防御のために必要な依頼を出すだけですみます。残りの細かい制御は STONITHd が引き受けます。

第 2 章

設定の基礎

2.1 設定の構成要素

クラスタの詳細な構成は XML 形式で表記され、設定と状態の 2 つのセクションに分かれています。

status セクションはリソースごとの全ノードの履歴を記録しています。子のデータにもとづいて、クラスタは現在の状態を構成できます。status セクションの信頼できる情報源はローカルリソースマネージャ (lrmd) で、クラスタはセクション全体をときどき再構成します。このような理由により、これはディスクに書き出されることはなく、したがって管理者はどのような方法であってもこれを修正しようとすべきではありません。

configuration セクションはクラスタのオプション、リソースのリスト、リソースをどこに置くべきかといった情報を記録しています。configuration セクションの説明が本書のおもな目的です。

configuration セクションはさらに 4 つのパートに分かれています。

- 設定オプション (crm_config と呼びます)
- クラスタを構成するノード
- リソース定義
- リソース間の関係 (constraints、制約)

```
<cib generated="true" admin_epoch="0" epoch="0" num_updates="0" have-quorum="false">
 <configuration>
  <crm_config/>
  <nodes/>
  <resources/>
  <constraints/>
 </configuration>
 <status/>
</cib>
```

図 2.1　設定前の CIB

2.2 クラスタの現在の状態

クラスタの構築を始める前に、クラスタの現在の状態を知る方法を知っておくことは有益です。この目的のために、crm_mon ユーティリティが用意されています。このユーティリティは、クラスタの状態をノード別にまたはリソース別に表示し、1 回だけ表示させるか連続更新表示モードで表示することができます。さらに、実行された操作のリストをノードまたはリソース別に表示して、失敗した操作に関する情報を表示するというモードもあります。

crm_mon を使うと、クラスタの異常状態を確認したり、実際の障害やシミュレートした障害に対してクラスタがどう振る舞うかを調べられます。

指定できるすべてのオプションは crm_mon --help コマンドで確認できます。

```
============
Last updated: Fri Nov 23 15:26:13 2007
Current DC: sles-3 (2298606a-6a8c-499a-9d25-76242f7006ec)
3 Nodes configured.
5 Resources configured.
============
Node: sles-1 (1186dc9a-324d-425a-966e-d757e693dc86): online
Node: sles-2 (02fb99a8-e30e-482f-b3ad-0fb3ce27d088): standby
Node: sles-3 (2298606a-6a8c-499a-9d25-76242f7006ec): online
Resource Group: group-1
   192.168.100.181       (heartbeat::ocf:IPaddr):      Started sles-1
   192.168.100.182       (heartbeat:IPaddr):           Started sles-1
   192.168.100.183       (heartbeat::ocf:IPaddr):      Started sles-1
rsc_sles-1 (heartbeat::ocf:IPaddr):     Started sles-1
rsc_sles-2 (heartbeat::ocf:IPaddr):     Started sles-3
rsc_sles-3 (heartbeat::ocf:IPaddr):     Started sles-3
Clone Set: DoFencing
   child_DoFencing:0     (stonith:external/vmware):    Started sles-3
   child_DoFencing:1     (stonith:external/vmware):    Stopped
   child_DoFencing:2     (stonith:external/vmware):    Started sles-1
```

図 2.2 crm_mon の標準的な出力

```
============
Last updated: Fri Nov 23 15:26:14 2007
Current DC: sles-3 (2298606a-6a8c-499a-9d25-76242f7006ec)
3 Nodes configured.
5 Resources configured.
============
Node: sles-1 (1186dc9a-324d-425a-966e-d757e693dc86): online
   192.168.100.181       (heartbeat::ocf:IPaddr):    Started sles-1
   192.168.100.182       (heartbeat:IPaddr):         Started sles-1
   192.168.100.183       (heartbeat::ocf:IPaddr):    Started sles-1
   rsc_sles-1     (heartbeat::ocf:IPaddr):    Started sles-1
   child_DoFencing:2     (stonith:external/vmware):  Started sles-1
Node: sles-2 (02fb99a8-e30e-482f-b3ad-0fb3ce27d088): standby
Node: sles-3 (2298606a-6a8c-499a-9d25-76242f7006ec): online
   rsc_sles-2     (heartbeat::ocf:IPaddr):    Started sles-3
   rsc_sles-3     (heartbeat::ocf:IPaddr):    Started sles-3
   child_DoFencing:0     (stonith:external/vmware):  Started sles-3
```

図 2.3 crm_mon -n の標準的な出力

DC (Designated Controller) ノードはクラスタ全体に対する意思決定を行うノードです。これがダウンしたら、残っているクラスタノードから新しい DC が選出されます。DC ノードのログは有用であるという点を除いて、どのノードが DC かということは、管理者にとって重要ではありません。

2.3 クラスタの設定を変更するには

クラスタの設定を変更するための基本的なルールは 3 つです。

- ルール 1 - cib.xml ファイルを書き換えてはならない。
- ルール 2 - ルール 1 を再確認する。
- ルール 3 - もしもルール 1 と 2 を破ったら、クラスタはそのことを注意して新しい設定の適用を拒否する。

このことを深く心に留めていただいた上で、代わりにどうすればいいかを説明します。

クラスタの設定を修正するためのもっとも強力なツールは `cibadmin` コマンドです。このコマンドは実行中のクラスタに働きかけます。`cibadmin` を使って、クラスタ状態の問い合わせ、設定の一部の追加、削除、更新、置換を行えます。そしてすべての変更はただちに有効になります。このため、再読み込みといった処理は必要ありません。

`cibadmin` のもっとも簡単な使い方は、現在の設定を一時ファイルに保存し、それを任意の XML エディタで編集してアップロードする、という方法です。

```
cibadmin --query > tmp.xml
vi tmp.xml
cibadmin --replace --xml-file tmp.xml
```

優秀な XML エディタを使えば、Relax NG スキーマを使って XML 表現が正しいことを確認できます。ほとんどのシステムで、設定を記述するためのスキーマは `/usr/lib/heartbeat/pacemaker.rng` にあります。

`resources` セクションだけを編集したいのであれば、

```
cibadmin --query --obj_type resources > tmp.xml
vi tmp.xml
cibadmin --replace --obj_type resources --xml-file tmp.xml
```

というコマンドを実行します。設定の他のパートを間違って編集してしまうことが避けられます。

2.4 設定の一部をまとめて削除するには

最初に削除したいオブジェクトを検索します。

```
sles-1:~ # cibadmin -Q | grep stonith
       <nvpair id="cib-bootstrap-options-stonith-action" name="stonith-action" value=" ↩
           reboot"/>
       <nvpair id="cib-bootstrap-options-stonith-enabled" name="stonith-enabled" value ↩
           ="1"/>
      <primitive id="child_DoFencing" class="stonith" type="external/vmware">
       <lrm_resource id="child_DoFencing:0" type="external/vmware" class="stonith">
       <lrm_resource id="child_DoFencing:0" type="external/vmware" class="stonith">
       <lrm_resource id="child_DoFencing:1" type="external/vmware" class="stonith">
       <lrm_resource id="child_DoFencing:0" type="external/vmware" class="stonith">
       <lrm_resource id="child_DoFencing:2" type="external/vmware" class="stonith">
       <lrm_resource id="child_DoFencing:0" type="external/vmware" class="stonith">
       <lrm_resource id="child_DoFencing:3" type="external/vmware" class="stonith">
```

次に、リソースのタグおよび ID を決定します (たとえば `primitive` および `child_DoFencing`)。そして次のコマンドを実行します。

```
cibadmin --delete --crm_xml ‘<primitive id=”child_DoFencing”/>’
```

2.5 XML を使わずに設定を変更するには

いくつかの一般的な処理では、XML を読み込んで編集する代わりに、高レベルのツールを使えます。

STONITH を有効にするには、次のコマンドを実行します。

```
crm_attribute --attr-name stonith-enabled --attr-value true
```

`somenode` 上でリソースを実行できるかどうかをチェックするには、次のコマンドを実行します。

```
crm_standby --get-value --node-uname somenode
```

`my-test-rsc` リソースがどのノードで実行されているかを調べるには、次のコマンドを実行します。

```
crm_resource --locate --resource my-test-rsc
```

2.6 作業領域で設定を変更する

実際の設定を更新する前に、一連の変更作業の効果を確認しておきたいことがあります。このために、crm_shadow コマンドが用意されています。このコマンドは設定のシャドウコピーを作成して、コピーに対してすべてのコマンドラインツールの利用を可能にします。

最初に、作成したいシャドウコピーの名前[*1]を指定した crm_shadow コマンドを実行して、現在の設定をコピーします。以後は画面に表示される説明にしたがってください。なおこの操作を間違えると、アクティブなクラスタ設定を更新してしまうことになります。

```
c001n01:~ # crm_shadow --create test
Setting up shadow instance
Type Ctrl-D to exit the crm_shadow shell
shadow[test]:
```

図 2.4 新しい作業領域の作成

```
shadow[test] # crm_shadow --which
test
```

図 2.5 現在のシャドウコピー名の確認

続けて実行するコマンドは、現在のアクティブな設定に対してではなく、シャドウコピーに対して実行されます。

```
shadow[test] # crm_failcount -G -r rsc_c001n01
 name=fail-count-rsc_c001n01 value=0
shadow[test] # crm_standby -v on -n c001n02
shadow[test] # crm_standby -G -n c001n02
name=c001n02 scope=nodes value=on
shadow[test] # cibadmin --erase --force
shadow[test] # cibadmin --query
 <cib cib_feature_revision="1" validate-with="pacemaker-1.0" admin_epoch="0"  ↩
     crm_feature_set="3.0" have-
    quorum="1" epoch="112" dc-uuid="c001n01" num_updates="1" cib-last-written="Fri Jun ↩
         27 12:17:10 2008">
  <configuration>
   <crm_config/>
   <nodes/>
   <resources/>
   <constraints/>
  </configuration>
 <status/>
</cib
```

図 2.6 シャドウコピーに対する設定の変更

[*1] 複数のシャドウコピーを作成でき、それぞれは名前で識別されます。

実験が終ったら、変更をコミットするか、または破棄します。破棄方法は次に示すとおりです。ここでも画面表示にしたがってください。

```
shadow[test] # crm_shadow --delete test --force
Now type Ctrl-D to exit the crm_shadow shell
shadow[test] # exit
c001n01:~ # crm_shadow --which
No shadow instance provided
c001n01:~ # cibadmin -Q
 <cib cib_feature_revision="1" validate-with="pacemaker-1.0" admin_epoch="0"  ↩
     crm_feature_set="3.0" have-
    quorum="1" epoch="110" dc-uuid="c001n01" num_updates="551">
  <configuration>
   <crm_config>
     <cluster_property_set id="cib-bootstrap-options">
        <nvpair id="cib-bootstrap-1" name="stonith-enabled" value="1"/>
        <nvpair id="cib-bootstrap-2" name="pe-input-series-max" value="30000"/>
```

図 2.7 シャドウコピーを破棄して実際の設定が変更されていないことを確認

2.7 変更した設定のテスト

クラスタ設定のシャドウコピーに対してさまざまな変更を施す方法を説明しました。これらの変更を実際のクラスタにロード (`crm_shadow --commit mytest --force`) する前に、ptest ユーティリティで効果をシミュレートするとよいでしょう。

```
ptest --live-check -VVVVV --save-graph tmp.graph --save-dotfile tmp.dot
```

`ptest` は実際のクラスタ (live と呼びます) と同じライブラリを使って、シャドウコピーの内容が live にどのように作用するかを示してくれます。`ptest` は、大量のログ表示に加えて、`tmp.graph` と `tmp.dot` という 2 つのファイルを作成します。これらのファイルは、どちらも変更箇所に関するクラスタの応答を記録しています。`.graph` ファイルには、すべてのアクション、パラメータ、前提条件のリストを含めて、完全な状態遷移が記録されます。このファイルは非常に可読性が低いため、同じ内容を Graphviz 用の dot 形式のファイルで同時に作っておくといいでしょう。

[graphviz-1.png not found]

図 2.8 Graphviz で作成した遷移グラフの例

Graphviz の出力から次のことが読み取れます。

- 実線の矢印は遷移の順序関係を表します。
- 点線の矢印は実際の遷移が起こらない依存関係を表します。
- 点線の枠で囲まれたアクションは実際の遷移に関与しないアクションを表します。
- 緑の枠は遷移するアクションを表します。
- 赤の枠はクラスタが実行したかったが実行できなかったアクションを表します。
- 青の枠はクラスタが実行する必要はないと判断したアクションを表します。

- 橙の名前はグラフを簡素にするためにクラスタが使う簡略化した名前です。
- 黒の名前は LRM に送られたアクションを表します。
- アクション名は”リソース名_アクション_実行間隔 ノード名”という形式になっています。
- 赤枠のアクションに依存するアクションは、すべて実行できません。
- もしもループが起こっていたら、これはきわめてまずいです。開発チームに連絡してください。

上に示した例では、次のことが読み取れます。ノード 2 が新たにオンラインになり、クラスタは rsc1、rsc2 および rsc3 がまだそこで実行していないことを確認します (*_monitor_0 がこのことを表します)。このことが確認されたので、クラスタは node1 上の rsc1 と rsc2 を停止して、これらを node2 に移動しようとします。しかしなんらかの問題があったため、停止アクションは実行できないか許可されませんでした。したがって、ノード 2 で開始することもできませんでした。別のなんらかの理由で、クラスタは rsc3 をどのノードでも実行しようとはしていません。

`ptest` に関するヘルプ情報は `ptest --help` で表示できます。

[graphviz-1.png not found]

図 2.9 別のより複雑な遷移グラフの例

2.8 全クラスタノードの設定を更新する必要はありません

クラスタに対するどのような設定変更であっても、それはすべてのアクティブなクラスタメンバーにただちに同期されます。

バンド幅を節約するために、更新された設定の差分のみがインクリメンタルに他のノードにアナウンスされます。また、各ノードが保持するコピーが確実に同じものになるよう、MD5 チェックサムを活用します。

第 3 章

クラスタオプション

3.1 特殊オプション

オプション等の構文解釈の都合により、以下のオプションは他のオプションとは違ってトップレベルのオプションと位置付けられています。これらのオプションは、どのような値を指定した場合でも、設定データベース自身によって利用されます。

3.1.1 採用する設定の決定に関与するオプション

ノードがクラスタに参加したとき、クラスタは最適な構成の設定を持っているノードを探します。このときに以下のオプションが使われます。クラスタは admin_epoch、epoch、num_updates の組み合わせの最大の値を持っているノードに対して、全ノードの設定を置き換えるよう指示します。これは、クラスタ全体の設定を正しく維持するために、とても重要です。

フィールド	説明
admin_epoch	クラスタはこの値を修正しない。このオプションは、アクティブでないノードを無効にするために使用する。 値をゼロにしてはならない。ゼロを指定すると、ディスク上にある”空の”設定と実際の設定の違いを見つけられなくなってしまう。
epoch	設定が更新されるたびに、通常は管理者によって 1 増やされる。
num_updates	設定または状態が更新されるたびに、通常はクラスタによって 1 増やされる。

3.1.2 他のフィールド

フィールド	説明
validate-with	設定に関する懸賞方法を指定する。none を指定すると、クラスタは更新内容の DTD への適合性をチェックせず、適合していない場合でも拒否しない。このオプションは、異なるバージョン間でのアップグレード中に有用である。

3.1.3 クラスタが管理するフィールド

フィールド	説明
crm-debug-origin	最後の更新の出所を記録する。情報としての記録のみが目的。
cib-last-written	設定を最後にディスクに書き込んだ時間。情報としての記録のみが目的。
dc-uuid	クラスタのリーダーであるノード。リソースの配置やいくつかのイベントの順序を決定するために、クラスタが使用する。
have-quorum	クラスタがクォーラムを持つかどうか。false なら、クラスタがリソースを開始できないか他のノードを防御できないことがある。後述の no-quorum-policy を参照。

管理者がこれらのフィールドに値を設定することも可能ですが、ほとんどの場合管理者が設定した値はクラスタによって「正しい」値に書き換えられます。admin_epoch を書き換えるには次のコマンドを実行します。

```
cibadmin --modify --crm_xml '<cib admin_epoch="42"/>'
```

フィールド全体は次のようになります。

```
<cib have-quorum="true" validate-with="pacemaker-1.0" admin_epoch="1" epoch="12"  ↩
    num_updates="65"
dc-uuid="ea7d39f4-3b94-4cfa-ba7a-952956daabee" >
```

図 3.1 cib オブジェクトのフィールド

3.2 クラスタオプション

クラスタオプションは、ある事態に出会ったときにクラスタがどう振る舞うかを指定するオプションです。

オプションはいくつかのセットにまとめられ、高度な構成の場合にはその数は 2 以上になります[*1]。ここでは、各オプションが 1 回だけ使われるシンプルなケースを取り上げます。

3.2.1 指定可能なクラスタオプション

オプション	デフォルト	説明
batch-limit	30	TE が同時に処理できるジョブの数を指定する。適切な値はネットワークとクラスタノードの処理速度および負荷に依存する。

[*1] このことは、ルールに関する後述のセクションで説明します。ここでは、たとえば平日の営業時間中 (当然ながらダウン時間を可能な限り短くする必要があります) と休日 (ユーザに迷惑をかけることなくリソースを適切なノードに移動できます) でオプションのセットを使い分ける例を取り上げます。

オプション	デフォルト	説明
no-quorum-policy	stop	クォーラムを持たない場合のクラスタの挙動を指定する。指定できる値は stop、freeze、ignore および suicide
symmetric-cluster	TRUE	すべてのリソースが任意のノードで実行できるか。
stonith-enabled	TRUE	フェイルしたノードや停止できないリソースを持つノードを撃つ (電源を落とす) か。貴重なデータを扱うクラスタでは、STONITH デバイスを設定してこのオプションを有効にする。 TRUE を指定した場合またはこのオプションを指定しない場合、STONITH リソースを少なくとも 1 つ以上指定していなければ、クラスタはリソースの開始を拒否する。
stonith-action	reboot	STONITH デバイスに送るアクション。reboot または poweroff を指定できる。
cluster-delay	60s	ネットワークのラウンドトリップ時間 (アクションの実行時間を除く)。適切な値はネットワークとクラスタノードの処理速度および負荷に依存する。
stop-orphan-resources	TRUE	取り除いたリソースは停止すべきか。
stop-orphan-actions	TRUE	取り除いたアクションは停止すべきか。
start-failure-is-fatal	TRUE	FALSE を指定すると、クラスタはリソースの失敗回数 resource-failure-stickness を代わりに用いる。
pe-error-series-max	-1	#####The number of PE inputs resulting in ERRORs to save. Used when reporting problems
pe-warn-series-max	-1	#####The number of PE inputs resulting in WARNINGs to save. Used when reporting problems.
pe-input-series-max	-1	#####The number of ”normal” PE inputs to save. Used when reporting problems.

クラスタオプションのデフォルト値と最新の値のリストは、pengine metadata コマンドを実行すれば得られます。

3.2.2 クラスタオプションの検索と設定

クラスタオプションを検索したり設定するには crm_attribute コマンドを使います。cluster-dalay オプションの現在の値を調べるには、次のコマンドを使います。

```
crm_attribute --attr-name cluster-delay --get-value
```

簡潔な指定方法も用意されています。

```
crm_attribute --get-value -n cluster-delay
```

値が設定されていたら、次のような出力が得られます。

```
sles-1:~ # crm_attribute --get-value -n cluster-delay
name=cluster-delay value=60s
```

値が設定されていなければ、コマンドはエラーを返します。

```
sles-1:~ # crm_attribute --get-value -n clusta-deway
name=clusta-deway value=(null)
Error performing operation: The object/attribute does not exist
```

値をたとえば 30s に変更するには、次のコマンドを実行します。

```
crm_attribute --attr-name cluster-delay --attr-value 30s
```

値をデフォルトに戻す場合、次のように値を削除します。

crm_attribute --attr-name cluster-delay --delete-attr

3.2.3 オプションが繰り返し表示される場合

コマンドの実行結果が次のようになる場合、オプションが複数回設定されています。

```
# crm_attribute --attr-name batch-limit --delete-attr
Multiple attributes match name=batch-limit in crm_config:
Value: 50 (set=cib-bootstrap-options, id=cib-bootstrap-options-batch-limit)
Value: 100 (set=custom, id=custom-batch-limit)
Please choose from one of the matches above and suppy the 'id' with --attr-id
#
```

図 3.2 複数回値が表示されたオプションの削除

このような場合は、表示される指示にしたがって処理を進めてください。現在どの値が使われているかを知る方法については、ルールに関するセクションを参照してください。

第 4 章

クラスタノード

4.1 クラスタノードの定義

クラスタを構成するノードの UUID、uname (ホスト名)、タイプは、次の形式で nodes セクションに登録されます。

```
<node id="1186dc9a-324d-425a-966e-d757e693dc86" uname="sles-1" type="normal"/>
```

図 4.1 クラスタノードの例

正常な環境の場合は、ノード間の通信とメンバシップデータによってクラスタ自身がこの情報を自動的に組み上げます。crm_uuid ツールを使ってクラスタ開始前に値を設定したり、設定された UUID を確認することもできます。

4.2 クラスタノードを詳細に定義する

ノードに関する基本的な情報に加えて、管理者は RAM サイズ、OS、カーネルバージョン、物理的な設置場所などの追加情報を定義することもできます。リソースをどこに配置するかをクラスタが決定するときに、これらの情報を使わせることもできます。ノード属性に関する詳細は「ルール」に関するセクションを参照してください。

ノード属性は、クラスタが開始して実行中であっても、crm_attribute コマンドで登録できます。

現在のノード (sles-1) にカーネルバージョン情報を追加するには、次のコマンドを実行します。

```
crm_attribute --type nodes --node-uname sles-1 --attr-name kernel --attr-value `uname -r`
```

登録後のノードの定義は次のようになります。

```
<node uname="sles-1" type="normal" id="1186dc9a-324d-425a-966e-d757e693dc86">
<instance_attributes id="nodes-1186dc9a-324d-425a-966e-d757e693dc86">
<nvpair id="kernel-1186dc9a-324d-425a-966e-d757e693dc86" name="kernel" value ↩
    ="2.6.16.46-0.4-default"/>
</instance_attributes>
</node>
```

図 4.2 crm_attribute で登録された属性を確認

ある属性の現在の値を調べるには、次のようなコマンドを実行します。

```
crm_attribute --type nodes --node-uname sles-1 --attr-name kernel --get-value
```

--type nodes は、クラスタの永続的な属性を表します。他に一時的な属性もあります。これは status セクションに記述され、ノードがクラスタに参加しなおすと失われます。クラスタはこのエリアにリソースのフェイル回数を記録します。管理者は--type status を指定してこの情報を確認できます。

4.3 クラスタノードの追加

4.3.1 OpenAIS の場合

新しいノードに OpenAIS と Pacemaker をインストールして、既存ノードから /etc/ais/openais.conf と /etc/ais/authkey をコピーします。追加するノードの IP アドレスに合わせて mcastaddr オプションを修正しなければならないことがあります。

OpenAIS のログに”invalid digest”が表示される場合は、ノード間のキーが一致していないことになります。

4.3.2 Heartbeat の場合

autojoin を指定してある場合は、Heartbeat と Pacemaker をインストールして既存ノードから /etc/ha.d/ha.cf と /etc/ha.d/authkeys をコピーするだけです。

そうでない場合は、ha.cf と authkeys を設定して、新しいノードを起動する前に hb_addnode コマンドを実行します。

4.4 クラスタノードの除去

4.4.1 OpenAIS の場合

####未記述

4.4.2 Heartbeat の場合

通信とメンバシップレイヤはクラスタノードに関する権威ある情報源であるため、単純に CIB からノードを削除することは信頼性がある方法とは言えません。まず heartbet にノード (以下の例では sles-1) を忘れさせる必要があります。このためには、まず sles-1 の heartbeat を停止し、残っているアクティブノードで次のコマンドを実行します。

```
hb_delnode sles-1
```

これを行った上で、次のコマンドを使って CIB からノードを除去します。

cibadmin --delete --obj_type nodes --crm_xml ‘ ¡node uname= ”sles-1 ”/¿ ’ cibadmin --delete --obj_type status --crm_xml ‘ ¡node_status uname= ” sles-1 ” /¿ ’

4.4.3 クラスタノードの置き換え

4.4.3.1 OpenAIS の場合

既存のノードを置き換えるには次の 5 つのステップを実行します。

1. 置き換え対象のノードを完全に停止させます。
2. 新しいサーバに古いサーバと同じホスト名と IP アドレスを割り振ります。
3. 新しいサーバにクラスタ関連ソフトウェアをインストールします。
4.
5. `/etc/ais/openais.conf` と `/etc/ais/authkey` を既存ノードからコピーします。
6. 新しいクラスタノードを開始します。

OpenAIS のログに”invalid digest”が表示される場合は、ノード間のキーが一致していないことになります。

4.4.3.2 Heartbeat の場合

第 5 章

第 6 章

第 7 章

第 8 章

第 9 章

第 10 章

第 11 章

第 12 章

第 13 章

第 14 章

第 15 章

第 16 章

第 17 章

第 18 章

第 19 章

第 20 章

第 21 章