高信頼HAクラスタに向けた障害通知フレームワークの実装

NTT コムウェア株式会社 大塚憲司 佐々木博正
VA Linux Systems Japan 株式会社 黒澤 崇宏

1 はじめに

通信事業者向けのシステムや金融システムといった、大規模システムへ Linux を適用することに対する要求が強くなっている。こういった分野に Linux を適用するためには、可用性の向上が重要である。このため、HA クラスタソフトなどを用い、障害の検出を行い、障害が発生したサーバの業務をほかのサーバへ切り替えることで、可用性の向上を図っている。

このとき、サーバの負荷が高くなると応答性が落ち、HA クラスタソフトが障害と誤検出し、必要のない切り替え処理を実施することとなり、結果として可用性をそこなうことになる。

負荷が高くなることによる応答性の低下は、メモリ確保の遅延やスケジューリングの遅延に起因しており、HA クラスタソフトなどのユーザ空間のプロセスからの制御には限界がある。また、カーネルでなければ検出できない事象も存在する。

そこで、サーバの負荷が高い状態でも安定した障害検知を可能にし、それを統一的な形式でユーザ空間のプロセスに対して提供する障害通知フレームワークを設計、実装した。障害通知フレームワークは、統合されたインタフェースを提供することで、プラグインにより障害検出の機能追加が容易に行えると共に、ユーザ空間のプロセスは各種の障害検出を統一的に扱うことも可能にする。

本論文では、障害検知機能として、クラスタシステムを構成する相手ノードの状態を監視するハートビートプラグインを実装し、障害通知フレームワークの有効性を検証した。

2 障害通知フレームワークの設計

障害通知フレームワークは、障害発生などのイベントをカーネル内で捉え、それを統一的な形式でユーザ空間のプロセスに対して提供するカーネルモジュールである。

カーネル内で障害発生を検出することによって、負荷が高くなることによる応答性の低下を避けることができる。また、デバイスドライバのような下位のレイヤで発生したエラーを検出することができる。

障害通知フレームワークを実装することで、監視プロセスは各種の障害検出を統一的に扱うことが可能になる。また、障害を検出する機能は、フレームワークのプラグインとして実装されるため、障害検出の機能追加を容易に行うことが可能である。

2.1 コンポーネント構成

障害通知フレームワークを用いたシステム[1]は、以下のコンポーネントから構成されている。

- 障害通知フレームワーク
- 障害検知プラグイン
- 監視プロセス（実装範囲外）

監視プロセスとは、障害通知フレームワークからイベント（障害通知）を受け取り、そのイベントに応じてリソースの切り替えなどを行うユーザ空間のプロセスである。代表的な例として、クラスタ管理ソフトウェアが挙げられる。

これらのコンポーネントの構成と、コンポーネント間のインタフェースの概略を図 1 に示す。

障害通知フレームワークは、各種障害検知プラグインから報告されるイベントを監視プロセスへ統一的に通知するのが主な役割である。通知の際には、監視プロセスから指定されたイベントのみを通知する。

また、イベントが障害通知フレームワークに報告されたときに、アクション（挙動）を起こすことができ

[1] システムとしては HA クラスタシステムなどが考えられる。

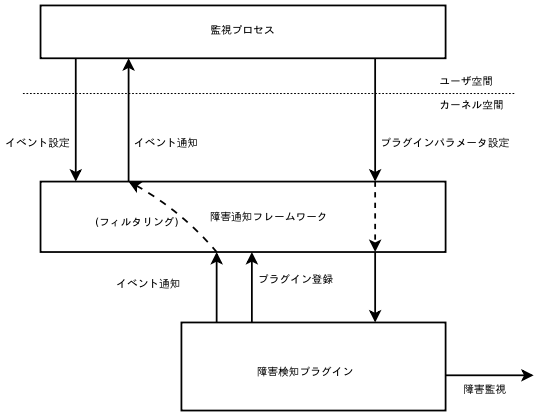

図 1: コンポーネント構成

る。挙動を起こす際にも、監視プロセスから指定された挙動を発生させる。本論文では、発生させることができる挙動として、カーネルパニックを実装している。

障害検知プラグインは、監視する対象の種別ごとに存在する。代表的なものとしては、他ノード監視のためのハートビートプラグイン、ディスク障害検知プラグインなどが挙げられ、各プラグインはカーネルモジュールとして実装される。障害検知プラグインは、初期化時に自身を障害通知フレームワークに登録する。プラグインに固有なパラメータ設定が必要となる場合は、必要なパラメータを障害通知フレームワーク経由で監視プロセスに見せることになる。

監視プロセスは、障害通知フレームワークからのイベント通知を受け、クラスタ資源のフェイルオーバなどを決定することとなる。監視プロセスは、ユーザ空間のプロセスとして実装される。

2.2 フレームワークのインタフェース

障害通知フレームワークは、ユーザ空間でイベントを受ける監視プロセスに対し、以下のインタフェースを提供する。これらのインタフェースは、proc ファイルシステムを用いて実装した。

- 各イベントに対して通知･非通知の要求や、イベント発生時の挙動設定のためのインタフェース（イベント設定インタフェース）
- 監視プロセスに対し、イベント通知を行うためのインタフェース（イベント通知インタフェース）
- 各障害検知プラグインのパラメータ設定を橋渡しするためのインタフェース（プラグインパラメータ設定インタフェース）

障害検知プラグインに対しては、以下のインタフェースを提供する。これらのインタフェースは、カーネル内の関数として実装した。

- プラグイン登録・登録解除インタフェース
- プラグインからのイベント通知を受け付けるインタフェース（イベント受信インタフェース）

障害検知プラグインは、障害通知フレームワークに対して障害イベントの通知を行う前に、前もってプラグイン登録インタフェースを使用して自身を障害通知フレームワークに登録しておかなければならない。また、プラグインの終了処理の際には、登録解除インタフェースを使用してフレームワークから自身の登録を解除する必要がある。

イベント受信インタフェースは、障害検知プラグインが障害を検知した際などに、そのイベントをフレームワークに対し通知するのに使用される。

3 プラグインの実装例

本論文では、プラグインの実装例として他ノードの状態を検出するハートビートプラグインのプロトタイプを実装した。

3.1 概要

一般にハートビートによるノード監視はユーザプロセスとして実装されることが多いが、その場合、I/O 高負荷などによりシステムメモリが不足すると、メモリの回収処理による遅延が発生し、誤ってノード異常と判断してしまうケースがある。

そこで、ハートビートパケットの送受信に必要なメモリを固定的に確保して動作するカーネル内ハートビートモジュールを、障害通知フレームワークのプラグインとして開発した。

ハートビートプラグインは、他ノードの状態をハートビートパケットの送受信により相互監視するためのプラグインであり、相互監視している他ノードの状態

が変化した場合にはフレームワークに対しイベント通知を行う。クラスタ管理ソフトウェアなどのユーザプロセスは、ハートビートプラグインからのイベントを障害通知フレームワーク経由で受け取ることで、クラスタノードの状態を把握可能となる。

ハートビートパケットは自身の存在のアナウンスとして送信される。つまり、ハートビートパケットの送信に対し、他ノードからのレスポンスが返されるわけではない。

3.2 設計

3.2.1 コンポーネント構成

ハートビートプラグインのコンポーネント構成図を、図 2 に示す。

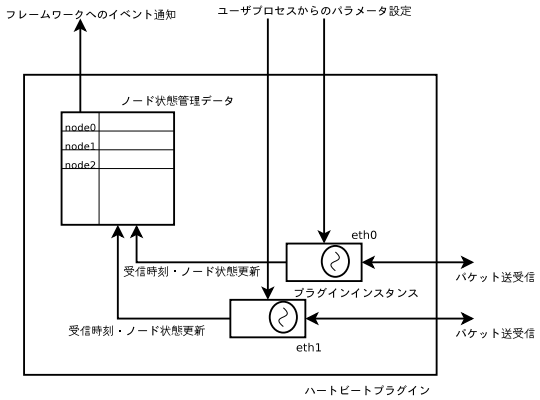

図 2: コンポーネント構成

ハートビートプラグインの基本的な動作は、イーサネットのネットワークインタフェースからハートビートパケットをマルチキャストにより一定間隔ごとに送信し、一方で他ノードから送信されたハートビートパケットを受信するといった形式となる。他ノードからのハートビートパケットの受信状況から、他ノードの状態を管理し、状態に変化があったら障害通知フレームワークに通知を行うこととなる。

ハートビートプラグインでは、ハートビートパケットを送受信するネットワークインタフェースごとにカーネルスレッドを起動する。フレームワークからは、このネットワークインタフェース単位 (カーネルスレッド単位) でプラグインインスタンスが割り当てられているように見えるものとする。これにより、ネットワークインタフェース単位のパラメータ設定を行うことで、監視に用いる通信路ごとのパラメータ設定が可能となる。

3.3 障害通知フレームワークへのイベント通知

ハートビートプラグインは、相互監視しているノードの状態に変化があった場合に、フレームワークに対しイベント通知を行う。通知することになるイベントの種類について以下に示す。

- ノードダウンイベント (nodedown): 全てのハートビートプラグインのインスタンスにおいて、`deadtime` 以上の時間を経過しても対象とするノードからのハートビートパケットが到着しなかった場合に通知される。イベントに付随する情報としては、ダウンしたノードの名前を通知する。なお、ダウン検出されたノードの状態はダウンとし、ダウン状態でさらに `deadtime` 以上の間ハートビートパケットが到着しない場合にはイベント通知は行わない。
- ノードアップイベント (nodeup): アップ以外の状態にあるノードから、ハートビートパケットが到着した際に通知される。イベントに付随する情報としては、アップ状態になったノードの名前を通知する。ノードダウンイベントの場合と同様、ノード状態が変化しない場合にはイベント通知は行わない。
- リンクダウンイベント (linkdown): あるハートビートプラグインのインスタンスが監視対象のどのノードからも `deadtime` 以上の時間ハートビートパケットを受信しなかった場合に通知する。イベントに付随する情報としては、そのインスタンスが担当するネットワークインタフェース名が通知される。以上で説明したイベントと同様、リンク状態が変化しない場合にはイベント通知は行わない。
- リンクアップイベント (linkup): あるハートビートプラグインのインスタンスが、担当するリンクの状態がアップ状態以外の状態で、監視対象のいずれかのノードからハートビートパケットを受信

した場合に通知する。イベントに付随する情報としては、そのインスタンスが担当するネットワークインタフェース名が通知される。以上で説明したイベントと同様、リンク状態が変化しない場合にはイベント通知は行わない。

- 送受信処理異常発生イベント (sendfail): 送受信処理で異常が発生した場合に通知する。ハートビートメッセージ送信用ソケットバッファが deadtime 経過しても利用可能とならなかった場合などがこれに該当する。

なお、2 ノード構成のクラスタの場合、一方のノードがダウンすると他方のノードでは全インスタンスからリンクダウンイベントが通知されることとなる。ハートビートプラグインが通知するリンクダウンイベントは、クラスタノード間の通信路としてのリンクがダウンしたというイベントであり、実際のハードウェアのリンクがダウンしたということを必ずしも示さないという点に注意する必要がある。

3.4 ページ回収処理によるパケット送受信処理遅延の回避

ユーザプロセスからハートビートパケットを送信する場合、パケット送信時にパケット管理用のオブジェクトであるソケットバッファ(struct sk_buff) および送信データを格納するためのデータ領域が確保され、ネットワークデバイスドライバより送信が完了した時点で解放される。これらのデータオブジェクトはスラブキャッシュとして管理されており、割り当てるオブジェクトがなくなった時点で新たにスラブキャッシュ用のページが確保されるが、I/O 高負荷などによりシステム上のメモリが不足してくると、ページの回収処理が実行される。ページの回収処理では、利用頻度の低いページを解放していくが、この時ファイルデータのキャッシュなどに利用されているページはディスクへ書き戻す必要があり、ディスク I/O 処理が発生するため、パケットの送信処理も遅延することになる。

そこで、ハートビートプラグインからハートビートパケットを送信する際には、ハートビートプラグインであらかじめ確保しておいたソケットバッファ中にハートビートパケットを構成し、IP レイヤに渡すようにする。さらに、ソケットバッファの参照カウンタを 1 つ余分に加算しておくことで、パケット送信完了時にネットワークデバイスドライバでの解放を防ぎ、再利用が可能となるため、メモリ不足時における遅延を回避することが可能となる。

また、ユーザプロセスにおいて相手ノードからのハートビートパケットを監視するために、通常 select() または poll() などが使用される。select() または poll() では呼ばれる毎に監視テーブル用メモリの確保・解放処理が行なわれ、メモリ確保時にシステムメモリが不足していると、ページ回収処理が実行され処理が遅延することになる[2]。この場合、相手ノードからハートビートパケットが送信されていないのか、ページ回収処理により遅延しているのかユーザプロセス側で判断することができないため、相手ノードがダウンしたと誤検出してしまう場合がある。

これに対してハートビートプラグインでは、直接 sock 構造体の sk_sleep メンバでパケットを待ちあわせるため、メモリ不足による遅延を回避することができる。

ただし、受信パケット用のソケットバッファはネットワークデバイスドライバで確保されるため、これを固定的に確保することはできない。システムメモリ不足によりソケットバッファが確保できない場合は、パケット送信時と違いページ回収処理が実行されることはなく[3]、単にパケット受信が行なわれないことになる。

3.5 検証

障害通知フレームワークとハートビートプラグインによって、メモリ回収処理による遅延が発生しないことを検証した。

一般的に知られる HA クラスタソフトでは、I/O 高負荷状態によってハートビートが遅延しシステムストールを誤検出する問題を持つものがあるが、今回の検証の結果、I/O 高負荷状態でもハートビートが遅延することなく、誤検出をすることもなかった。

[2] これらの問題は epoll() を使用することで回避するこが可能である。

[3] パケットの受信を通知する割り込み処理でソケットバッファの確保およびパケットの受信が行なわれるが、その場合、メモリ確保を待つためにスリープすることが許されないため、ページ回収処理は実行せずにエラーとなる

3.5.1 検証環境

検証に使用したサーバマシンを、表 1 に示す。

表 1: 検証に使用したサーバマシン

項目	概要
モデル	HP ProLiant DL360 RD4 x 2 台
CPU	Xeon 3.40GHz x 2
メモリ	2GBytes
ディスク	SCSI Disk 42.8GB x 2 Compaq RAID Controller Smart Array 64xx 論理ドライブ 1 72GBytes (RAID-1)
NIC	eth0 Broadcom BCM5704 Gigabit Ethernet eth1 Broadcom BCM5704 Gigabit Ethernet

Linux カーネルは、2.6.10 を使用した。
ハートビートの設定は、表 2 の通りとする。

表 2: ハートビートの設定

項目	概要	設定値
interval	ハートビートの送信間隔	750ms
deadtime	監視対象をノードダウンと判定するまでの時間	4500ms

3.5.2 検証方法

I/O 高負荷状態は、`dd` と `sync` を使用して発生させる。
具体的には以下のような処理となる。

- バックグラウンドで `sync` コマンドを 1 秒間隔で連続的に実行させる。
- `dd` コマンドを同時に 35 プロセス起動する。1 つの `dd` コマンドは、512MBytes のファイルを作成する。(35 プロセスの合計 17.5GBytes)
- 全 `dd` コマンドの完了を待ち合わせ後に、上記の処理を繰り返す。

このような処理を 2 時間行い、検証を行った。

3.5.3 検証結果

カーネルログや障害通知フレームワークのイベントを監視することにより、I/O 高負荷時に相手ノードのダウンが誤検出されるといったことはなかった。また、`tcpdump` により、I/O 高負荷時でも interval で設定した値である 750ms 間隔でハートビートが送信されていることを確認した。

4 課題と今後の展開

4.1 障害通知フレームワークの機能強化

今回はプロトタイプ的に必要最小限の機能しか実装しておらず、実用化を進める上では機能強化が必要となる。主な機能強予定項目を以下に列挙する。

シグナルによるイベント通知
プラグインが障害を検出したことを知らせる手段として、シグナルによる通知機能を追加する。ユーザプロセスはイベント通知を非同期で知ることができるようになるため、利便性が向上する。

パラメータ参照機能
現状、設定したパラメータの参照機能が実装されていない。パラメータに設定されている内容を確認する手段として必要であり、参照機能を提供する。

4.2 ハートビートプラグインの機能強化

今回は、プラグインの実装例としての意味あいが強く、障害通知フレームワーク同様、実用化を進める上では機能強化が必要となる。主な機能強予定項目を以下に列挙する。

リンクダウン検出機能の強化
2 ノード構成とした場合、片方のネットワークインタフェースに障害が発生すると、両方のノードに対して、ノードダウンイベントおよびリンクダウンイベントが通知さることになるため、どちらのノードで障害が発生したのかを判断することができない。そのため、別プラグインとしてリンクダウン検出プラグインを提供するか、ハートビートプラグインのリンクダウン検出機能を強化する必要がある。

ハートビート統計情報の提供

ノードダウンを検出した際に、送信側と受信側のどちらのノードに問題があるのかを切り分けたり、プラグインパラメータである interval, deadtime を最適な値に調整するために、以下のような統計情報をユーザプロセスに提供することで、利便性が向上する。

- パケット最終送信時刻
- パケット最終受信時刻
- 最大・最小・平均 パケット送信間隔
- 最大・最小・平均 パケット受信間隔
- ノードダウン回数、ノードアップ回数など

deadtime のレベル設定

ソフトリミット、ハードリミット的な利用を考慮し、レベル毎に deadtime を設定する機能を提供する。ソフトリミットを超えた場合は警告を出力するのみとして interval の調整に利用し、ハードリミットを超えた場合にフェイルオーバーさせるなどの利用が考えられる。

ノード単位毎の deadtime 設定

現状 deadtime は、ネットワークインタフェース毎にしか設定できないため、監視対象の中で最もスペックの低いノードにあわせて設定することになる。要件によっては、ノードのハードスペックや重要度に応じて、監視レベルの精度を変えたい場合が考えられる。そのため、ノード毎に deadtime を設定する機能を提供する。

4.3 その他のプラグインの充実

この障害通知フレームワークを成功に導くためにはプラグインの充実が不可欠である。今後、実装を予定しているプラグイン例を以下に列挙する。

I/O エラー検出プラグイン

ファイルシステムレイヤとデバイスドライバの間で動作し、デバイスドライバで発生した I/O エラーを検出するプラグインを提供する。I/O エラーを検出した時点で即ファイルオーバーを行うことができるため、より高信頼な HA クラスタシステムを構築することが可能となる。

メモリ負荷状態監視プラグイン

メモリ使用状況をイベントとして通知するプラグインを提供する。システム負荷状況をユーザプロセス側でリアルタイムに監視することができ、その情報をもとにトラフィックの制限などを行うことが可能となる。また、メモリに関するチューニングを行なう上で必要な情報を整理し、ユーザプロセスへ提供する。

- kswapd（ページ回収デーモン）の動作頻度
- kswapd の動作時間
- 単位時間あたりの平均ページ回収量など

5 まとめ

大規模システムへの Linux の適用に対する要求から、Linux の可用性の向上が求められている。そこで、サーバの負荷が高い状態でも安定した障害検知を可能にし、それを統一的な形式でユーザ空間のプロセスに対して提供する障害通知フレームワークの設計、実装を行った。これにより、カーネル内でのみ検出できる障害などをユーザ空間のプロセスに伝えることができ、障害検出の機能追加が容易に行えるようになった。また、ハートビートプラグインのプロトタイプを実装し、その有効性を検証した。

今後は、障害通知フレームワークやハートビートプラグインの機能強化を図る。また、障害検知プラグインの充実を図る。このように実用化を進めることで、高信頼 HA クラスタの構築を目指す。

6 謝辞

本開発は、NTT コムウェア株式会社と VA Linux Systems Japan 株式会社との共同プロジェクトとして実施した。

本開発を行うにあたって、高橋 浩和氏をはじめ VA Linux Systems Japan 株式会社の皆様には有益なアドバイスを多く頂き大変感謝いたします。また、
若山　弘和氏をはじめ NTT コムウェア株式会社オープンソースソフトウェア推進部 OSS 基盤部門カーネルグループの皆様にも、多くのコメントを頂き大変感謝いたします。

参考文献

[1] Linux-HA Project Web Site
http://www.linux-ha.org/