heartbeat と MySQL で実現した データベースクラスタリング環境の構築

岩田 浩 (IWATA, Hiroshi) iwatahrs@nttdata.co.jp
俊成 清博 (TOSHINARI, Kiyohiro) toshinarik@nttdata.co.jp
渡部 広志 (WATABE, Hiroshi) watabeh@nttdata.co.jp

株式会社 NTT データ 第四公共ビジネスユニット

概 要

昨今データベースサーバやアプリケーションサーバなど信頼性が重要視される部分に Linux が用いられるケースが増えてきている。それに伴い、商用 UNIX 上で提供されていたクラスタリング製品の移植を中心に、Linux におけるクラスタリングソリューションも充実してきている。このように商用 UNIX の守備範囲であった分野に Linux が進出する一方で、元来より Linux が得意としていた小規模分野においても、より高い信頼性が求められてきている。本論文では、High-Availability Linux Project の成果物の一部である heartbeat を用いた、小規模分野におけるデータベースクラスタリング環境構築の実例と合わせて、その際に実装した追加機能について紹介する。

1. はじめに

コンピュータシステムの普及・高度化に伴い、Linux を用いた中・小規模システムにおいてもハードウェア障害あるいはソフトウェア障害時におけるサービス継続性を目的とした HA (High Availability：高信頼性) システムが求められている。

従来 HA システムといえば、機能・信頼性・ベンダによる保証などを考慮し Solaris や HP-UX といった商用 UNIX に商用 HA プロダクトを組み合わせて構成される場合が多かったが、Linux においても VERITAS Cluster Server といった著名な商用 HA プロダクトや、Oracle RAC といった商用 DBMS 製品における HA 構成がサポートされるにあたり、急速に HA 構成が広まってきている。

このような状況において、筆者が所属する部署では主に中・小規模システムのシステム構築に取り組んでいることもあり、商用製品よりもオープンソースプロダクトを中心に、システムインテグレータの立場 (＝オープンソースの利用者の立場) でオープンソース HA プロダクトに注目し、その実用性について検証を重ねてきた。

本論文では、High-Availability Linux Project[1] の成果物の一部である heartbeat に対しプロセス監視機能を追加し、DBMS である MySQL と組み合わせて実現したフェールオーバー型のデータベースクラスタリング構成について紹介する。

2. HA クラスタリングとは

クラスタリングを方式別に分類すると、HPC (High Performance Computing) クラスタと、HA (High Availability) クラスタに分類される。HPC クラスタはワークステーションおよび PC を束ねて高速な並列計算を目的とした並列・分散コンピューティング環境で、Linux で動作する代表的なプロダクトとしては Beowulf[2] や Score[3] がある。

HA クラスタは、複数のサービスでグループを組み、外部的に単一のサービスと見せることで、可用性や応答性の向上を実現する構成である。さらに方式により「負荷分散型クラスタ」と「フェールオーバー型クラスタ」に分類される。

以下に本論文の対象である HA クラスタについて簡単に説明する。

2.1. 負荷分散型クラスタ

負荷分散型クラスタは主に負荷分散を目的とした HA クラスタである。アクセスを分散させるリダイレクタと実際のサービスを提供するノード (ノードグループ) で構成される (図1)。リダイレクタはノード側で実行されるサービスの状態を監視し、必要に応じてノードに対するアクセスを制御 (負荷分散やグループへのノード追加/削除) することにより、グループ全体で可用性や応答性を向上させる。主にノード間でのデータ同期の必要が低い構成 (静的コンテンツ Web サー

バ、メールサーバ、DNS サーバ等）で利用される。Linux で動作する代表的なプロダクトとしては LVS (Linux Virtual Server Project[4]) が挙げられる。

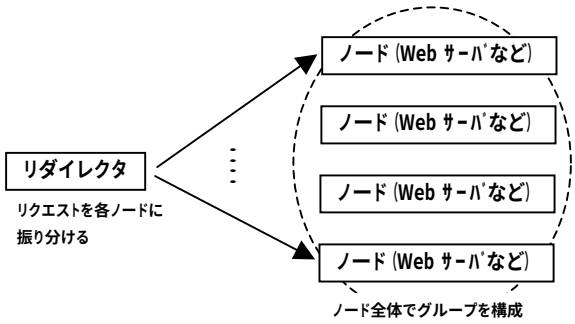

図 1 負荷分散型クラスタ

2.2. フェールオーバー型クラスタ

負荷分散型クラスタが障害の検知やノードの追加/削除といった制御部分と、サービスを提供するノードが分離した構成であることに対し、フェールオーバー型クラスタはサービスとクラスタリング制御部分を同一のノード上に配置し、障害発生ノード上のサービスをほかのサーバで引き継ぐこと（フェールオーバー）することにより高可用性を実現する（図2）。

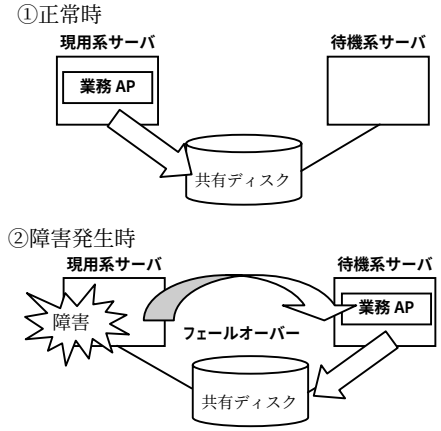

図 2 フェールオーバー型クラスタ

サービスのフェールオーバーを実現させるためには、ノード間でデータを引き継ぐ必要があるが、引き継ぐデータをノード間で共有するディスク上に置く方式を共有ディスク型、ノード間でデータを複製する方式をレプリケーション型と本論文では区別する。

共有ディスク型はデータの同期を取る必要が無いため、大量のデータを扱うシステムに適している。その反面、共有ディスクを構成するためには複数チャネル接続に対応した SCSI ストレージや FibreChannel 接続ストレージを必要とし、高価なハードウェア構成となる。

レプリケーション型はそれぞれのノード間でディスク内容をレプリケーションすることによりデータの整合性を保つ構成である。データベースの場合 DBMS が有するレプリケーション機能を利用し、書き込み要求の度にデータの同期を行う。また通常のファイルシステム上のデータの場合は rsync コマンド等を使用して一定間隔でデータの同期を行う。いずれにせよ、データ同期の度にノード間で通信が発生するため、データの更新量が大きく更新頻度が高いシステムには適さないが、共有ディスクである必要が無いため、比較的低価格なハードウェア構成で実現可能である。

3. Linux における HA クラスタリング

3.1. 商用プロダクト

Linux における商用クラスタリングプロダクトは、商用 UNIX から移植された製品を中心に存在する。以下に代表的な商用プロダクトを示す。いずれのプロダクトもフェールオーバー型クラスタをサポートしている。

- VERITAS Cluster Server （米 Veritas）
- CLUSTERPRO （NEC）
- DNCWARE Cluster Perfect （東芝）
- LifeKeeper （SteelEye）
- Matrix HA （PolyServe）
- PRIMECLUSTER （Fujitsu Siemens）
- RoseHA （Rose）

また、商用プロダクトにおける負荷分散型クラスタの主流はソフトウェアからハードウェア機器に移行している。

3.2. オープンソースプロダクト

Linux におけるオープンソースな HA ソリューションは High-Availability Linux Project において、ノード生死の監視および仮想 IP アドレスの引継ぎを実現する heartbeat コマンドの開発が続けられている。

さらに負荷分散エンジンである LVS と heartbeat を組み合わせたプロジェクトとして、UltraMonkey[5] がある。

また、LVS と VRRP (Virtual Router Redundancy Protocol[6]) に加えて独自のヘルスチェック機構を組み合わせプロジェクトとして KEEPALIVED[7] がある。

UltraMonkey、KEEPALIVED のいずれも冗長化された負荷分散型クラスタの実現を目的としている。

4. データベースクラスタリング環境の構築

筆者が手がけた案件において、どのようにしてオープンソースプロダクトを使用したクラスタリング構成を実現したか、その過程と実現内容を紹介する。

4.1. システム構成

今回、筆者は以下のシステム要件を持つシステム基盤の設計･構築を担当した。

(1) Web アプリケーションの実行基盤
(2) インターネットとは接続しない独立したネットワークを構成
(3) クライアント台数は50台程度
(4) 24時間365日サービスを提供
(5) ネットワークを含めて SPOF[1]なし
(6) 可能な限りオープンソースを使用
(7) 業務 AP は Perl (CGI) で記述
(8) ハードウェアは IA サーバを使用
(9) できる限り低価格

上記の要件でまず決まったことは、Web サーバ、AP サーバおよび DB サーバの構成である。業務アプリケーション開発グループの要望により、Web アプリケーション記述言語に Perl を選択したため、Web サーバと AP サーバを同一サーバで構成することとした。さらに業務分析の結果、処理データ量や更新頻度、レスポンスタイムなどの要件から見て、Web/AP サーバと DB サーバを同一のサーバで構成してもパフォーマンス上、問題が無いことが判明した。

次に、要件 (4) および (5) を考慮し、Web/AP/DB サーバの冗長化構成について検討を行った結果、Active/Standby 構成の HA クラスタリング構成とした。

またセキュリティを考慮し、クライアントセグメントとサーバセグメントの間に冗長化したファイアウォールを設置することとした。

4.1.1. DBMS の選定

本構成では、Web/DB サーバを二台でクラスタリング構成とする。サーバ二台でのクラスタリング構成には、共有ディスク方式とレプリケーション方式があるが、データ更新量と更新頻度からレプリケーション方式で問題ないと判断した。DBMS は、オープンソースプロダクトでレプリケーション構成に標準で対応している MySQL[8]を使用した。

4.1.2. Web サーバの選定

業務 AP を Perl で記述すること、および筆者が所属する部署におけるノウハウの蓄積量が多い事から、Web サーバには Apache を採用した。

4.1.3. クラスタリングソフトウェアの選定

クラスタリングソフトウェアは、カスタマイズ自由度、自力でのトラブル解析の容易から、商用プロダクトではなく、オープンソースプロダクトを選定した。

3.2 章で紹介したプロダクトの中より、今回は heartbeat を使用してデータベースクラスタリングの構成を試みた。

4.1.4. ネットワーク構成

heartbeat はそれ自身では NIC 冗長化の仕組みを持たないため、本構成では Linux の持つ NIC Bonding driver[9] を用いて NIC の冗長化を行った。bonding されたインターフェイスに対して、heartbeat が仮想 IP アドレスを共有リソースとして構成することにより、NIC の完全冗長化構成が実現される。図 3 にソフトウェアを含めた、本構成のシステム構成図を示す。

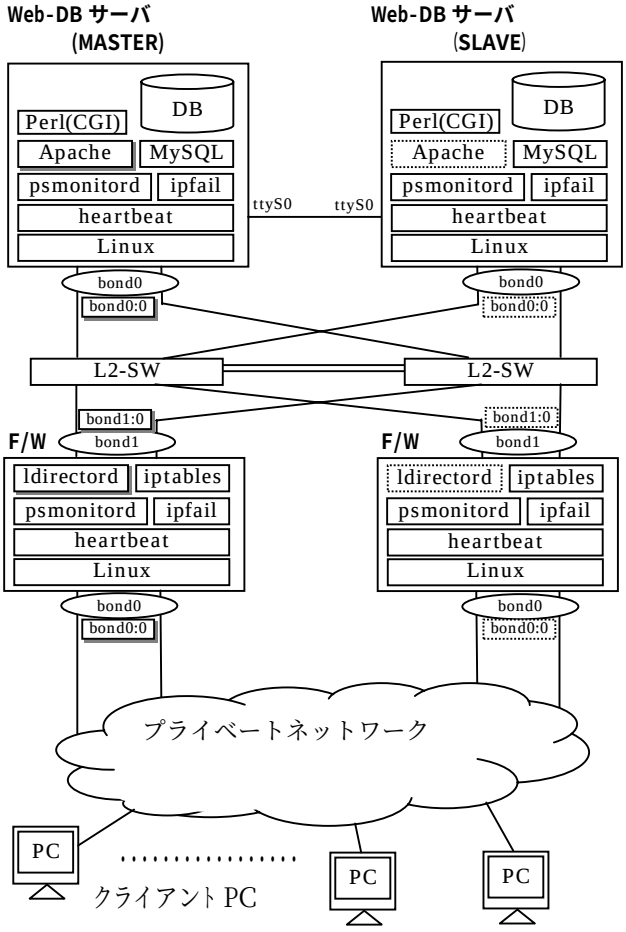

図 3 システム構成図

[1] Single Point Of Failure の略

4.2. heartbeat

4.2.1. heartbeat の構成

heartbeat の構成を図 4 に示す。heartbeat はクラスタ構成として2台構成のみサポートしている。

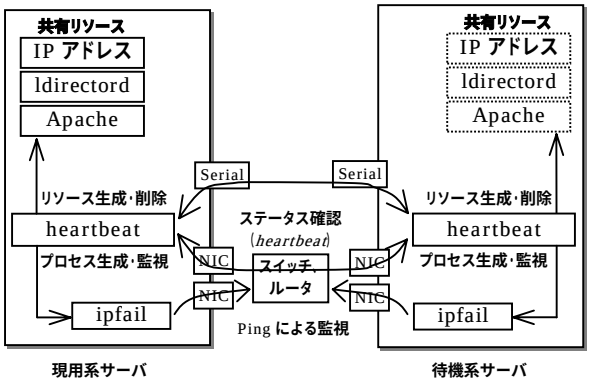

図 4 heartbeat の構成

プロセスとしての「heartbeat」と、クラスタ間でステータス確認の通信を意味する語である「*heartbeat*：心臓の鼓動」が混在するため、今後は前者を heartbeat、後者を *heartbeat* と記述する。

(注) heartbeat はさらに “master control process”、“FIFO reader”および”*heartbeat* を行うプロセス”とに分けられるが、図 4 では同一に記述している。

heartbeat はお互いに常時通信を行い、現在のステータスの確認、自身または相手側に障害が発生した場合の共有リソースの引継ぎ（フェールオーバー）を制御する、クラスタリングシステムにおいて中心となるプログラムである。そして、heartbeat 単体プログラムで全ての機能を実現するのではなく、いくつかの外部プログラムと連携してクラスタリング機能を提供する構成となっている（図 4）。

heartbeat は *heartbeat* が途絶えた場合を除き、自身ではネットワーク障害を検知することができない。また *heartbeat* 障害を検知しても、冗長化されている *heartbeat* 経路のいずれかで通信が確保されている限りフェールオーバーは発生しない。クラスタリング構成では通常、スプリットブレイン状態を回避するために *heartbeat* 経路を冗長化するため、heartbeat のみではサーバ NIC や接続している SW-HUB 等の障害発生を検知し、自動的に待機系にフェールオーバーすることができないため、ipfail という heartbeat と連携してネットワークを監視するするプロセスを用いて、サーバ NIC などのネットワーク障害を検知する。

ipfail は障害監視対象である NIC の先に存在するノード（通常はインテリジェント SW-HUB やルータを指定）に対して ping 応答をチェックすることにより NIC 障害、ネットワーク障害を検知し、フェールオーバーを実現する。よって、通常は heartbeat と ipfail を組み合わせて構成する。

4.2.2. heartbeat の問題点

ここまで説明してきたように、heartbeat（+ Bonding + ipfail）ではノード障害やネットワーク障害には対応可能であるが、プロセス障害、ディスク障害などハードウェア関連の障害は検知できない。

そこで本事例では「プロセス障害」に対して、独自にプロセス監視スクリプト「psmonitord」を実装することにより、heartbeat に機能を追加した。

4.3. プロセス監視の実装

4.3.1. psmonitord の構成

heartbeat の構成（図 4）にあるプロセスの中で、ipfail は heartbeat がプロセスの存在を監視しているため障害発生時に自動復旧される（フェールオーバーの原因にはならない）。これに対して、heartbeat は heartbeat 自身や Apache、 MySQL といったプロセスに対する監視を行わないため、プロセス監視を行うスクリプト「psmonitord」を Perl で実装した（図 5）。

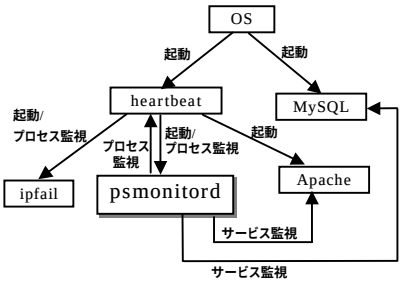

図 5 psmonitord の構成

4.3.2. heartbeat の監視

heartbeat は master control process、FIFO reader、write、read の各プロセスで構成されている[2]。各プロセスの障害時動作を表 1 に示す。

No.1 の場合、タイミングにより共有リソースを開放する場合と開放しない場合が混在する。開放しない場合であっても待機側では共有リソースが起動するため、リソースの競合といった重大障害を引き起こす。また、master control process 以外のプロセスが残存するため、heartbeat を通常の手段で停止/再起動することができない重大な障害となる。

[2] heartbeat ver. 1.2.0 における構成

表 1 プロセス障害時の動作

No.	プロセス障害対象	障害時の動作	共有リソースの扱い
1	master control process	該当以外のプロセスは残存	タイミングによって動作が異なる
2	FIFO reader	該当以外のプロセスは残存	フェールオーバー
3	write および read	master control process 以外のプロセスは一時的に消去され、その後再生成される	プロセス消去時にフェールオーバー※

※auto_failback=on の場合、プロセス再生成時にもフェールオーバーされる。

No.2 の場合、共有リソースはフェールオーバーするため、サービス自体は継続されるが、プロセスが残存するため heartbeat を正常な手段で停止/再起動することができない。

No.3 の場合、理想的な動作であり、特段の対応は不要といえる。

このため、No.1 および No.2 の場合に対応するため、psmonitord は master control process と FIFO reader を監視し、存在しない場合には heartbeat 関連のプロセスをすべて kill し、その後に共有リソースをフェールオーバーさせる。

また heartbeat プロセス障害検地時の際に psmonitord に任意のコマンド(例:shutdown)を実行させることも可能な実装とした。ただし、No.3 の場合では一時的に FIFO process が消去されるため、プロセス障害であるという誤検出を避けるためにタイムアウト時間とリトライ回数を指定できるように実装した。

4.3.3. ミドルウェアの監視

一般的にプロセス障害の検知には、ps コマンドの結果を元にプロセスの存在を確認するが、プロセスが存在していてもサービス機能に障害が発生している場合まで含めて障害検知を行いたい場合がある。そのため psmonitord ではプロセスの存在確認に加えて、指定ポートに対する反応の有無および、任意のコマンド実行結果によっても、障害検知が可能なように実装した。図 4 のとおり、本構成では Apache と MySQL を監視対象とする。

4.3.3.1. Apache の監視

Perl の Socket モジュールを使用し、指定ホストの指定ポート(デフォルトでは localhost, 80 番)に対して接続を行い、"HEAD / HTTP/1.0¥r¥n¥r¥n"に対して 指定した文字列(デフォルトでは"200 OK") を受け取る事で正常とみなす。

4.3.3.2. MySQL の監視

Perl の DBI モジュールを使用し、指定ホストの指定ポート(デフォルトでは localhost, 3306 番)に対して接続し、確認を行う。さらに本構成では、マスターサーバに対して"SHOW MASTER STATUS"およびスレーブサーバに対して"SHOW SLAVE STATUS"を実行することでレプリケーション状態を確認する。

4.3.4. heartbeat との連携

psmonitord はプロセスを監視し障害を検知するが、実際に共有リソースをフェールオーバーするのは heartbeat である。heartbeat は外部とのインターフェイスとして C 言語の heartbeat API と hb_standby コマンドの二種類を用意している。psmonitord は Perl で記述されていることから、hb_standby コマンドを使用して heartbeat と連携を行う。

hb_standby を実行すると、heartbeat が保有している共有リソースを強制的に開放し、その結果、他方のノードが共有リソースを保持する。

4.3.5. 障害検知時の動作

psmonitord は障害を検知すると heartbeat に対して hb_standby を実行し、保有している共有リソースを強制的に開放するよう指示し、heartbeat は standby 状態に入る。heartbeat の設定で auto_failback=on に設定している場合には、障害復帰のタイミングでフェールバックするため、本構成では auto_failback=off に設定した。

なお障害検知に際して、Apache, MySQL のいずれにおいても、指定回数のリトライを行い、リトライ回数以内で復帰した場合には、リトライ回数を0に戻す。これにより高負荷状態における誤検出を抑止する。

4.3.6. psmonitord 自身の監視

次に psmonitord 自身の監視を考える。本構成では、heartbeat から respawn 指定で psmonitord を起動させる。これにより psmonitord プロセス障害時には、heartbeat 自身がプロセス再生成を行う。

以下に heartbeat 設定ファイルの該当部分を示す。

psmonitord 起動部分(ha.cf)

```
respawn   root   /usr/lib/heartbeat/psmonitord
```

4.3.7. psmonitord.conf の書式

psmonitord の機能について紹介してきたが、以下に psmonitord の設定ファイルの書式を示す。

基本設定（psmonitord.conf）

```
INIT_WAIT="10"
TIMEOUT="10"
INTERVAL="5"
RETRY="2"
SYSLOG="local0.info"
HALT_COMMAND="/usr/bin/shutdown -h now"
```

Apache 監視用の設定（psmonitord_apache.conf）

```
PS="/usr/local/apache/bin/httpd"
HOST="localhost"
PORT="80"
METHOD="HEAD"
URL="/"
MATCH="401 Authorization Required"
INTERVAL="3"
RETRY="3"
FAILOVER="/usr/lib/heartbeat/hb_standby"
```

MySQL 監視用の設定（psmonitord_mysql.conf）

```
PS="/usr/local/mysql/libexec/mysqld"
MASTER_HOST="scwd01"
MASTER_PORT="3306"
MASTER_USER="admin"
MASTER_PASSWD=""
SLAVE_HOST="scwd02"
SLAVE_PORT="3306"
SLAVE_USER="admin"
SLAVE_PASSWD=""
INTERVAL="3"
RETRY="3"
FAILOVER="/usr/lib/heartbeat/hb_standby"
```

4.4. MySQL レプリケーション

heartbeat との組み合わせの前に、MySQL のレプリケーション機能の特徴について簡単に説明する。

1. MASTER→SLAVE の片方向レプリケーションのみ対応
2. SLAVE 側が MASTER 側に対して差分をチェックし、同期を行う
3. 同期が崩れた場合には、手動で DB 内容の同期を行わなければ再度レプリケーション構成にすることができない

MySQL のレプリケーション状態を確認するには、”SHOW MASTER STATUS” および ”SHOW SLAVE STATUS” を実行する。また ”change master” および ”slave start” コマンドを実行することによりレプリケーションの制御を行うことが可能である。

4.4.1. heartbeat と MySQL との組み合わせ

MySQL でレプリケーションを構成する場合、MySQL はそれぞれのサーバ上で常に起動されている必要がある。そのため MySQL 自体は heartbeat で管理する共有リソースにはならない。

クライアントからの通信は、共有リソースである仮想 IP アドレス宛に行われ、仮想 IP アドレスに対する通信を Apache が受け、CGI 経由で MySQL にアクセスが行われる。

この様な構成では、仮想 IP アドレスと同様に Apache も共有リソースとして管理し、フェールオーバーの際には両者を同時に SLAVE 側のサーバに引き継ぐ必要がある。 MySQL は片方向レプリケーションしかサポートしないため、共有リソースのフェールオーバーと同時に、MASTER 側の MySQL を ”SLAVE” 状態にし、SLAVE 側の MySQL を ”MASTER” 状態にする必要がある。このことを SLAVE 側の Apache の起動スクリプトに以下の修正を加えることで実現した。

Apache 起動スクリプト（/etc/ha.d/resource.d/apache）の追記内容

```
# Start MySQL Replication
/usr/local/mysql/bin/mysql -u admin -e "SLAVE STOP;"
/usr/local/mysql/bin/mysql -u admin -e "RESET MASTER;"
/usr/local/mysql/bin/mysql -u admin -e "RESET SLAVE;"
/usr/local/mysql/bin/mysql -u admin -e "CHANGE MASTER TO
MASTER_HOST='', MASTER_PORT=3306, MASTER_USER='';"
```

合わせて、以下に heartbeat, Apache, MySQL の連動イメージを示す。

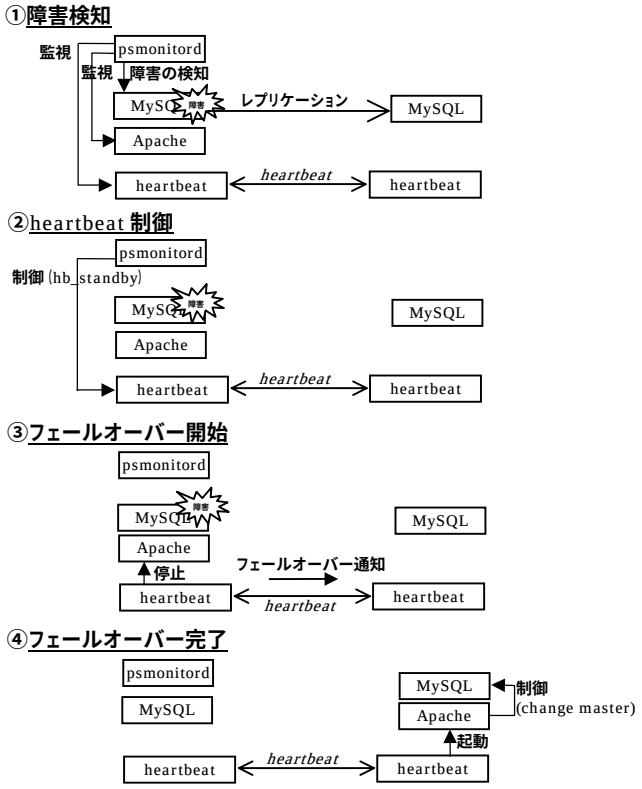

図 6 heartbeat/Apache/MySQL 連動イメージ
（MySQL 障害時の例）

5. 本構成で実現した基本機能

以下に本構成において実現した基本機能について、特に障害要因とそれに対応する動作の概要について簡単にまとめる。網掛け部分が新規に実装した範囲である。

表 2 本構成で実現した基本機能

障害要因	動作概要	実装分類
クラスタデーモン (heartbeat) 障害	フェールオーバー	psmonitord 新規実装 (一部)
IP 監視デーモン (ipfail) 障害	プロセス再生成 フェールオーバーせず	標準
プロセス監視デーモン (psmonitord) 障害	プロセス再生成 フェールオーバーせず	標準
NIC 片系障害	自動切換え フェールオーバーせず	NIC Bonding
NIC 両系障害	フェールオーバー	標準
ノード障害	フェールオーバー	標準
MySQL 障害	フェールオーバー	psmonitord 新規実装
Apache 障害	フェールオーバー	psmonitord 新規実装

5.1. 現状における制限事項

(1) 本構成ではフェールバック(切り戻し)には対応していない。これは MySQL のレプリケーション機能の制限により自動でフェールバック時にレプリケーションを再開できないためである。

(2) 本構成では psmonitord 間での情報の交換を行っていない。これは(1)の理由でフェールバックに対応できないため、あえて未実装としている。

(3) psmonitord は Apache, MySQL の監視に特化した構成となっている。これは本構成が特定の案件をターゲットとして構成しているためである。もちろんある程度の汎用化は考慮した実装としているが、Apache および MySQL 以外の監視に関してはテスト対象外としている。

6. 今後の課題

本構成では特定のお客様のシステム要件に対して必要な機能に実装対象を限定しため、以下の項目についても他のお客様の案件のシステム要件を勘案しつつ、実装の検討を進めて行きたいと考えている。

- 5.1 章で説明した制限事項の解消
- RAID コントローラ障害、ファン障害、温度異常といったハードウェア障害の検知と連携
- 共有ディスク構成での検証

7. 最後に

本論文では HA クラスタリングの概要を紹介し、オープンソース HA プロダクト(heartbeat)に対して、プロセス監視機能 psmonitord を追加実装したが、必要な機能や監視対象ミドルウェアを Apache と MySQL に限定することにより、最小限の追加コードで小規模 HA クラスタリングとして本事例に対する必要最低限に機能を実現できた。

psmonitord の実装アイデアは商用クラスタリング製品などを参考にしたため、今回の構成に技術的な新規性は少ないが、実際にお客様に対してシステム構築を進めていく過程で、クラスタリングソフトウェアを含め全てをオープンソースソフトウェアで構成することを目指していった応用事例として参考にしていただけたら僥倖である。

余談ではあるが、システム全体の提案・構築・運用を委託するシステムインテグレータの立場として、システム基盤の中核部分を支えるミドルウェアをオープンソースプロダクトで構成できるメリットは、試行錯誤を繰り返すだけの価値が十分にあったと考えている。

参考文献

[1] **High-Availability Linux Project**
http://linux-ha.org/

[2] **Beowulf.org - The Beowulf Cluster Site**
http://www.beowulf.org/

[3] **PC Cluster Consortium**
http://www.pccluster.org/

[4] **The Linux Virtual Server Project**
http://www.linuxvirtualserver.org/

[5] **ULTRA MONKEY**
High Availability and Load Balancing Solution for Linux
http://ultramonkey.org

[6] **Virtual Router Redundancy Protocol**
RFC 2338

[7] **KEEPALIVED**
HealthChecking for LVS & High-Availability
http://keepalived.sourceforge.net

[8] **MySQL**
http://www.mysql.com/

[9] **Linux Ethernet Bonding Driver mini-howto**
/usr/src/linux/Documentation/networking/bonding.txt