Nagios redundancy material

Nagios **冗長化検証資料**

Version 1.0

保証免責
本書は記載事項またはそれに関わる事項について、明示的あるいは黙示的な保証はいたしておりません。
したがいまして、これらを原因として発生した損失や損害についての責任を負いません。

表記について

本書では以下の書体を使用しています。

- イタリック文字

  本文中でのコマンド、ファイル名、変数など可変なパラメータ値を表します。

- 等幅文字

  ファイルの内容やコマンドの入出力例に使います。入力の場合にはボールドで表します。

```
$ cd /usr/src/sys/i386/conf
$ ls
GENERIC         Makefile      OLDCARD       SMP
GENERIC.hints   NOTES         PAE           gethints.awk
$
```

- 省略文字

  ファイルの内容やコマンドの入出力例を省略する場合に'...'を使います。

```
$ vi /etc/rc.conf
  ...
sshd_enable="YES"
named_enable="YES"
  ...
$
```

- プロンプト

  一般または、管理権限を持った実行環境をそれぞれ、'$'(ドル)、'#'(シャープ)のプロンプトで表します。

```
$ su
Password: root's passwd
#
```

目次

1. はじめに

1.1. 本書について

本書では、Nagios を核にした監視システム冗長化による高可用性を実現するための手法、および議論について記述しています。

Nagios スイートと関連アドオンの組合せによって、監視機能を冗長化する機能は提供されており、これらについても議論の対象としています。また、冗長化を行なうための手順、設定方法も記述してあります。

本書は、これらの冗長化手法が適切なソリューションであるかを議論するためのたたき台として利用してください。

1.2. 前提知識

Unix、または Windows の基本的なユーザオペレーションと、管理オペレーションが可能であることを想定しています。また、一般的な Internet プロトコルや、それに基づいて実装されたアプリケーションなどについて知っている必要があります。

本書では、ソフトウェア上の設定に関して、*parameter = value* といった実際の設定情報についてのみ記述します。これらの設定情報についての詳細は関連マニュアルを参照するべきでしょう。以下に挙げるドキュメントを参照しておくことを推奨します。

文献

文献	著者	リンク
monitoring environment using Nagios and PerfParse	Isidore	
Nagios® Version 1.x Documentation	Ethan Galstad	http://nagios.sourceforge.net/docs/1_0/
PerfParse Installation Guide	Garry Cook Ben Clewett Yves Mettier Flo Gleixner	http://perfparse.sourceforge.net/docs.php
apache	The Apache Software Foundation	http://httpd.apache.org/docs/2.0/
MySQL リファレンスマニュアル MySQL のバックアップ・リカバリ	MySQL AB ソニーグローバルソリューションズ(株)	http://dev.mysql.com/doc/mysql/ja/index.html http://www.mysql.gr.jp/frame/modules/bwiki/index.php?plugin=attach&refer=OSC2005%2F%BB%F1%CE%C1&openfile=20050326-MySQL.pdf

2. 高可用性の実現

2.1. 出発点

高可用性の実現にはいろいろな方法があると思いますが、そのすべてを書き出すことは本書では不可能です。しかし、いくつか候補に挙げることはできるはずです。
まずは Nagios の公式ページで提案されている冗長化手法について評価を行い、この手法を出発点として思考実験を行ってみましょう。

Nagios の開発者である Ethan Galstad は、冗長化についてのテーマとして http://nagios.sourceforge.net/docs/1_0/redundancy.html を挙げています。ここに記述されている内容を簡単にまとめると、以下のようになるでしょう。

Ethan Galstad 式

HA 手法	説明
2(多)重化 マスタ - スレーブ式 アクティブ - アクティブ	ポーリングは、多重化されたポーラすべてから行なわれます。スレーブは enable_notifications パラメータを 0 に設定させるため、マスタだけがインシデントの検出に伴ってアラートを発生します。 スレーブ側からハートビートを検査し、ハートビートエラーが発生するとイベントハンドラ経由で ENABLE_NOTIFICATIONS コマンドを nagios.cmd パイプファイルへ送信します。これにより、スレーブがアラートを発生するようになります。 また、マスタが復帰すると、スレーブは DISABLE_NOTIFICATIONS コマンドを nagios.cmd パイプファイルへ送信します。これにより、スレーブがアラートを発生しなくなります。
フェイルオーバ化 マスタ - スレーブ式 アクティブ - スタンバイ	上記の手法とよく似ていますが、ポーリングはマスタだけが行い、スレーブは何もしない、という点が異なります。 スレーブがハートビートエラーを検出すると、START_EXECUTING_SVC_CHECKS コマンドと ENABEL_NOTIFICATIONS コマンドを nagios.cmd パイプファイルへ送信します。これにより、スレーブがポーリングを行ない、インシデントを検出するとアラートを発生するようになります。 また、マスタが復帰すると、スレーブは STOP_EXECUTING_SVC_CHECKS コマンドと DISABLE_NOTIFICATIONS コマンドを nagios.cmd パイプファイルへ送信します。これにより、スレーブは何も行なわないようになります。

2.2. 要件開発

Ethan Galstad のアイデアはシンプルで、信頼性が高いように思われます。特にフェイルオーバやフェイルバックが自動的に行なわれるので非常に好ましい提案です。
さらに、Ethan Galstad はインシデントの状態をスレーブに引き継がせる、いわゆるステートフルなフェイルオーバについても触れています。

しかし、われわれがNagiosを監視基盤として利用する上で、高可用性について、いくつか解決しなければならない問題があります。つまり、Ethan Galstad のアイデアを拡張し、いろいろな問題を解決するための要件を開発しなければなりません。

① サービスチェックのステータス

Ethan Galstad は、ステートフルフェイルオーバの実現方法として NSCA の利用を示唆しました。これは、本来は監視の分散化に用いるアドオンで、パッシブチェックを行なうためのスイートです。

確かに、NSCA を用いることでサービスチェックの状態をスレーブに引き継がせることが可能です。しかし、NSCA の導入や、それに合わせたコンフィグレーションが必要になってきます。著者は、ルーチン化している監視サーバの構築方法をできるだけそのままの状態で利用したいのです。そしてすばやくサービスデリバリを行いたいと考えています。また、NSCA を導入しても、著者が認識している問題のすべてが解決するわけではありません。

② PerfParse のトレンドデータ

監視を行なう上でそのステータスを計数化して、顧客へ改善案を提案できたとしたらそれは素晴らしいことでしょう。ですから、PerfParse の持つ情報もフェイルオーバできなければ冗長化の恩恵が半減することでしょう。

③ Nagios のイベントログ

Nagios が出力するイベントログは意外と重要です。なぜなら Nagios コンソールの Reporting で示されるサブリンクは CGI プログラムが動作するように構成されていますが、CGI プログラムはイベントログを解析してレポート出力を行なうからです。
これらの機能は重宝しますから、イベントログもフェイルオーバするべきでしょう。

④ ハートビートエラーの検出

フェイルオーバの条件として、マスタが監視サービスを継続できないことを可能な限り早く、正確に検出しなければならないことでしょう。

これは、あらゆる監視情報はマスタからスレーブに引き継がなければなりませんが、いったんスレーブに監視制御が遷移した場合には、マスタの監視情報はもう必要ないはずです。つまり、監視制御が遷移した後は、スレーブがマスタとなるわけですから監視情報は元スレーブ自身が生成する必要があるはずです。このふるまいを決定するトリガとしてハートビートエラーの検出精度は重要になってきます。

⑤ フェイルバックの容易さ

マスタが故障して、スレーブへ切り替わることで高可用性を提供するわけですから、可能な限り迅速に元マスタを再構築しなければなりません。このことにより、現マスタが故障した場合の高可用性を維持できるからです。

2.3. 開発要件に対する解決案

前節でいろいろな要件開発を行ないましたので、これらに対する IT レベルでの実装策定を行ないましょう。

実装策定

問題	策定案
①サービスチェックのステータス ②PerfParse のトレンドデータ	これらのデータは、MySQL データベースによって管理されています。 マスタの持つ監視情報は、MySQL のレプリケーション機能によってスレーブに複製することで、フェイルオーバ時に備えることができます。 PerfParse のデータは、時間の経過とともに増大する傾向がありますので、レプリケーションは差分トランザクションログを基に複製します。 ただし、ハートビートエラーを検出した場合は、レプリケーションを停止します。
③Nagios のイベントログ	イベントログは単純なテキストファイルとして管理されています。 マスタのイベントログを周期的にスレーブにバックアップするために rsync を利用します。 イベントログは時間の経過とともに増大する傾向がありますので差分バックアップを行なう rsync は適切でしょう。 ただし、ハートビートエラーを検出した場合は、バックアップを停止します。
④ハートビートエラーの検出	これまでの著者の経験から言えば、欲しい機能は自分で作らなくとも既に誰かが作っていたことが多い(VCS や Sun Cluster などのプロダクトは除きますが)のですが、この機能について言えば、簡単なツールを作成する必要がありそうです。オープンソースでは、このようなものがあるようです。 http://ultramonkey.jp/2.0.1/heartbeat.html 実際は、ハートビートエラーだけを検出する精度の高いソフトウェアが欲しいのですが、このプロジェクトでは IP アドレスの引継ぎを主たる目的に作成されているようです。IP アドレスの切替え時に発生する ARP 問題を解決するために Linux カーネルに依存した作りになっているため、今回はこのソフトウェアの評価は見送りました。
⑤フェイルバックの容易さ	フェイルバックが簡単に行なえるのであれば、障害のためだけではなく、マスタ

	のメンテナンスに非常に役立つことでしょう。 しかし、今回の検証では次の 2 つの理由により、手動による運用が必要になります。 1 つは、MySQL のレプリケーション機能は現在のところ一方向にしか動作しないということです。A -> B へのレプリケーションを逆転するためには、コンフィグレーションを変更し、双方の MySQL を再起動しなければなりません。また、rsync でバックアップするデータもハートビート状態によってバックアップ方法が適切に変更できなければなりません。 もう 1 つの理由は、ハートビートエラー検出の精度が低いと思わぬデータベースの整合性に関する副作用が生じるだろうという危惧です。このため、自動フェイルバックは検証対象から除外します。

2.4. 運用時の前提

今回フェイルオーバの検証を行なうにあたり、以下の前提で行なうことを明示しておきます。

① 冗長構成

マスタ - スレーブ方式を採用し、マスタがアクティブでスレーブがスタンバイとします。
マスタは常にアクティブ状態であり、アクティブとは監視サービスを提供していることを意味します。

② マスタとスレーブ

マスタに障害が発生すると自動的にスレーブがマスタへ昇格します。
概念的に障害が発生したマスタはスレーブへと降格します。

③ フェイルオーバとフェイルバック

フェイルオーバは自動的に行なわれますが、今回の検証内容では概念的にフェイルバックは存在しません。フェイルオーバが発生するたびにマスタ - スレーブ間の昇格･降格が循環するように構成するからです。
たとえば yasu がマスタ、nabira がスレーブで構成されていて、yasu に障害が発生してフェイルオーバが発生すると nabira がマスタ、yasu がスレーブということになります。
ここで、yasu が復旧した場合、nabira のスレーブと位置づけられます。
nabira に障害が発生してフェイルオーバが発生すると、yasu がマスタに昇格します。
元のマスタ - スレーブ構成に一回りする現象をフェイルバックと呼ぶことも可能かもしれません。

④ 自動化部分

スレーブがマスタに昇格する部分のみが自動化されます。スレーブのセットアップと Nagios の web コンソール切り替えは手動です。

⑤ スレーブのセットアップ

MySQL のレプリケーション作業、Nagios イベントログのコピー、ハートビートエラー検出スクリプトの構成を行ないます。

2.5. **検証環境の用意**

検証する上で、確認用に使用した環境は以下の通りです。

監視サーバ環境

ホスト名	プラットフォーム	OS	冗長化の役割
yasu	Sun Microsystems Ultra10 CPU: UltraSparc-IIi 333.00 MHz メモリ: 256MB ディスク: 8GB NIC: hme0(100baseTX)	FreeBSD 5.3-RELEASE-p15	マスタ(アクティブ) 初期状態。
nabira	Sun Microsystems Ultra5 CPU: UltraSparc-IIi 333.00 MHz メモリ: 256MB ディスク: 8GB NIC: hme0 のみ使用(100baseTX)	FreeBSD 5.3-RELEASE-p15	スレーブ(スタンバイ) 初期状態。

ホストグループ

グループ A	www1
グループ B	cha1, cha2, masaki1, masaki2, masami, mx, tadashi, wataru
グループ C	bluenose, pamir, passat,

3. MySQL 概説

3.1. 概要

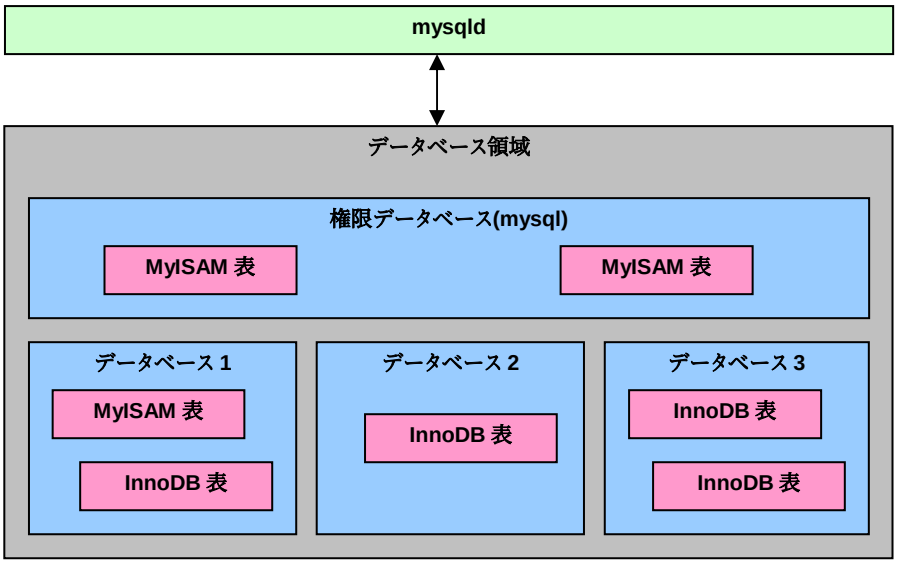
スレーブのセットアップを行なう前に MySQL の基本機能、およびバックアップ/レプリケーションについて簡単にまとめておきました。この章は、**MySQL のバックアップ・リカバリ**(OSC2005: Sony Global Solutions Inc.)の文書から引用させていただいています。

3.2. MySQL の機能概要

特徴

観点	特徴
動作環境	Linux、Solaris、FreeBSD、HP-UX、AIX、Windows 他。
実装言語	C/C++ マルチスレッド。
SQL 文	SQL92 の大半と SQL99 Core の一部、また便利な独自構文を提供。 サブクエリは 4.1 から、ストアードプロシージャは 5.0 から。
テーブル定義	InnoDB、MyISAM 等の型付けがあるストレージエンジン。 B-Tree 索引、整合性制約、シーケンスをサポート(ヴューは 5.0 から)。
トランザクション	ロックなしでの読取り一貫性、行ロック、4 段階全ての分離レベルなどをサポート(InnoDB のみ)。
バックアップ	ロックなしで一貫性のあるオンラインバックアップが可能(InnoDB のみ)。 増分バックアップ相当の機能を提供。
リカバリ	ロールフォワードリカバリ(過去の一時点も含む)が可能。 クラッシュリカバリが可能(InnoDB のみ)。
日本語処理	列単位でキャラクタセットの指定ができる(4.1 から)。 ベンダ定義文字のコード変換は未サポート(文字化けする)。
セキュリティ	認証手段は、パスワード認証と SSL クライアント証明書認証。 暗号化手段は、SSL と VPN(SSH ポートフォワーディングなど)。

3.3. データベースの構成



3.4. ディレクトリ/ファイルの種類

種類

名前	例
ベースディレクトリ	/var/db/mysql
データディレクトリ データベースディレクトリ	/var/db/mysql mysql、test、nagios、perfparse
表定義ファイル	*.frm
InnoDB データファイル InnoDB ログファイル MyISAM 表ファイル MyISAM 索引ファイル	ibdata1 ib_logfile0、ib_logfile1 *.MYD *.MYI
バイナリログファイル	yasu-bin.000001 nabira-relay-bin.000001
エラーログファイル ソケットファイル PID ファイル	yasu.err /tmp/mysql.sock yasu.pid

3.5. テーブルとファイルの関係

表 A～C: InnoDB 型。

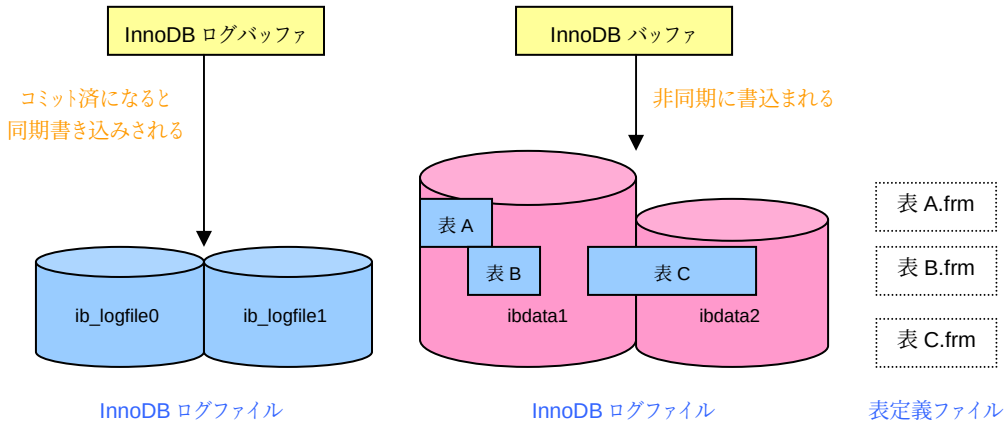


表 D～F: MyISAM 型。

3.6. バイナリログについて

① バイナリログとは

・MySQL のトランザクションログ(バイナリファイル)。

・SQL 文を時系列に蓄積したもの。

・コミット後に書込まれる。

・mysqlbinlog ユーティリティでテキスト形式の SQL 文に変換可能。

② バイナリログの取得方法

・デフォルトでは取得されない。

・デフォルトではコミット時、同期書き込みではない。

・ファイル名の拡張子は、6 桁の連番。

・log-bin を指定するとバイナリログが有効になる。

・innodb_safe_binlog を指定するとコミット時、同期書き込みが有効になる。

```
yasu# cd /var/db/mysql
yasu# cat my.cnt
  ...
[mysqld]
log-bin
innodb_safe_binlog
  ...
```

③ バイナリログのローテーション

・新しくファイルを作成して、書き込み先を切り替える(例: .000001 から .000002 へ)。

・切り替え後のファイルには書き込みは発生しない(.000001 はバックアップ可能)。

④ バイナリログのローテート時期

・SQL 文で FLUSH LOGS を発行する。

・mysqladmin ユーティリティにオプション--flush-logs を与えて実行する。

・mysqld を再起動した場合。

・ファイルが一定サイズを超過した場合(max_binlog_size 変数。デフォルトでは 1GB)。

4. 冗長構成のセットアップ

4.1. 概要

マスタが既に動作中である場合を仮定して、冗長構成としてスレーブをセットアップします。これは、マスタとスレーブをほぼ同時期にセットアップする場合でも手順は変わりません。本章では、マスタを yasu、スレーブを nabira が担うことを前提にしています。

また本章では、監視サーバを構築する手順をすべて記述しているわけではないことに注意してください。詳細な構築手順は 1.2. 前提知識 節の参考資料を参照することを勧めます。

本章全体をよく読み、手順を深く理解してください。

4.2. Nagios、PerfParse のセットアップ

マスタ、スレーブの Nagios と PerfParse は通常通りにセットアップしてください。
監視設定の内容は、マスタ、スレーブともに同じコンフィグレーションを用意しておきます。

スレーブの Nagios は停止しておきます(rc.conf では有効にします)。スレーブをリブートした場合に備えて、自動起動しないように nagios.sh を nagios.sh.disable といった名前に変更しておいてください。

4.3. HA 保守アカウントの用意

ハートビートスクリプトや Nagios イベントログのバックアップ用にマスタ、スレーブに保守アカウントを用意しておきます。例えば、以下のようなアカウントを用意してください。

```
# cat /etc/group
  ...
ha:*:3000:
# cat /etc/passwd
  ...
hb:*:3001:3000:infrastructure maintainer:/home/hb:/bin/sh
```

ssh-keygen コマンドでパスフレーズ無しの鍵ペアを生成し、相互に公開鍵認証による ssh 接続が可能な状態にしておきます。

また、Nagios グループアカウントに保守アカウントを追加しておきます。

```
# cat /etc/group
  ...
nagios:*:1001:www,hb
  ...
```

4.4. MySQL データベースのレプリケーション

Nagios と PerfParse のデータは MySQL で管理されています。マスタのデータベースをスレーブにレプリケーションしておけば、フェイルオーバに対する高可用性とともに整合性を保証することができるでしょう。

① マスタの my.cnf を以下のように変更します。その後、MySQL を再起動しましょう。
再起動後、yasu-bin.000001 や yasu-bin.index などのファイルが作成されることでしょう。

```
yasu# cd /var/db/mysql
yasu# vi my.cnf
  ...
log-bin
server-id = 1
  ...
yasu# /usr/local/etc/rc.d/mysql-server.sh restart
  ...
```

② マスタにレプリケーション可能なユーザを作成しておきます。
データベースアカウント repl の権限で、スレーブがマスタへレプリケーションを要求します。

```
yasu# mysql -u root -p mysql
mysql> grant replication slave on *.* to repl@'%' identified by 'passwd';
  ...
```

③ マスタのデータベースをバックアップします。

```
yasu# cd /var/db/mysql
yasu# mysqldump --opt --single-transaction --master-data --flush-logs ¥
--database nagios --database perfparse -u root -p > yasu_dump.sql
  ...
yasu# scp -i ~hb/.ssh/id_dsa yasu_dump.sql hb@nabira:
```

--flush-logs オプションを付加することでバイナリログがローテートされます。つまり、レプリケーション対象になるのは最新のバイナリログになります。

また、--master-data オプションの付加により、バックアップデータにレプリケーション対象となるマスタバイナリログファイル名とレプリケーション開始位置を設定する SQL 文が挿入されます。

④ スレーブの MySQL インスタンスを初期化します。

```
nabira# /usr/local/etc/rc.d/mysql-server.sh stop
  ...
nabira# cd /var/db/mysql
nabira# rm -fr ib_logfile* ibdata nabira-relay-bin.* master.info ¥
relay-log.info nagios/ perfparse/
  ...
```

スレーブの my.cnf を以下のように変更します。replicate-do-db パラメータにレプリケーションするデータベースを指定します。

```
nabira# cd /var/db/mysql
nabira# vi my.cnf
```

```
  ...
#log-bin
#server-id  = 1
  ...
server-id = 2
master-host = yasu
master-user = repl
master-password = passwd
replicate-do-db=nagios
replicate-do-db=perfparse
  ...
```

⑤ スレーブにバックアップデータをﾘｽﾄｱします。MySQL を起動する必要がありますが、以下の手順に従い、まだレプﾘケーｼｮﾝを開始しないようにしてください。

```
nabira# vi /etc/rc.conf
  ...
mysql_args="--skip-slave-start"
  ...
nabira# /usr/local/etc/rc.d/mysql-server.sh start
  ...
nabira# mysql -u root -p < ~hb/yasu.dump.sql
  ...
```

⑥ スレーブのレプﾘケーｼｮﾝを開始します。

```
nabira# vi /etc/rc.conf
  ...
#mysql_args="--skip-slave-start"
  ...
nabira# /usr/local/etc/rc.d/mysql-server.sh restart
  ...
```

rc.conf の mysql_args ﾊﾟﾗﾒｰﾀを無効化し、MySQL を再起動します。その際、スレーブのﾊﾞｲﾅﾘﾛｸﾞはﾛｰﾃｰﾄされます。

レプﾘケーｼｮﾝが正常に行なわれているか確認するには、以下の方法が使えます。

ﾚﾌﾟﾘｹｰｼｮﾝ用ｽﾚｯﾄﾞが起動したことを示すﾛｸﾞ:

```
nabira# tail -f /var/db/mysql/nabira.err
  ...
[Note] Slave I/O thread: connected to master 'repl@yasu:3306',  replication
started in log 'yasu-bin.000002' at position 4
  ...
```

Slave_IO_Running、Slave_SQL_Running のｽﾃｰﾀｽが Yes であれば問題ありません。:

```
nabira# mysql -u root -p
  ...
mysql> show slave status;
  ...
| Waiting for master to send event | yasu       | repl       |       3306
|          60 | yasu-bin.000003 |          36420952 |
nabira-relay-bin.000002 |     36420994 | yasu-bin.000003      | Yes
| Yes             | nagios,perfparse |                  |
|                    |                          |                          |
0 |          |          0 |          36420952 |       36420994 | None
|              |            0 | No               |                 |
|               |              |                  |                  0 |
started in log 'yasu-bin.000002' at position 4
  ...
```

4.5. Nagios イベントログのバックアップ

4.6. Nagios ハートビートのチェック節の logcopy 関数を参照してください。

このスクリプトの基本ロジックは、フェイルオーバが発生していなければ、マスタからイベントログを rsync でバックアップするという点です。フェイルオーバが発生した場合はスレーブの Nagios が起動されるという前提に基づくものです。

4.6. Nagios ハートビートのチェック

スレーブに以下のスクリプトを用意して、cron で実行させます。

```
nabira# cat /usr/local/bin/slavetools4nagios
#!/bin/sh
init() {
    if [ ! -f /usr/local/etc/slave2master.conf ] ; then
        exit
    fi
    . /usr/local/etc/slave2master.conf
}

logcopy() {
    if [ -f $FAILOVER_OCCURED ] ; then
        return
    fi
    $RSYNC   -az   -e   "$SSHOPTS"   $MASTER:$SRC_NAGIOS_ARCHIVESLOG
$DST_NAGIOS_ARCHIVESLOG > /dev/null 2>&1
    $RSYNC   -az   -e   "$SSHOPTS"   $MASTER:$SRC_NAGIOS_CURRENTLOG
$DST_NAGIOS_CURRENTLOG > /dev/null 2>&1
}

notification() {
    $NOTICE_CMD                          --notificationtype="FAILOVER"
--servicedesc="heartbeat: error detect" --hostname=$MASTER ¥
             --hostalias="master      -      Nagios      redundancy"
--hostaddress="`host $MASTER`" --servicestate=FAILOVER ¥
             --datetime="`date`"     --output="FAILOVER     detected
generation.
Nagios is started on $SLAVE.
Please confirm the watch." ¥
             --contactemail="sysadm@forschooner.net"
--customername="ForSchooner"
}

activate() {
    notification

    sed -e 's/server-id.*2/#&/' ¥
        -e 's/master-host/#&/' ¥
        -e 's/master-user/#&/' ¥
        -e 's/master-password/#&/' ¥
        -e 's/replicate-do-db/#&/' ¥
        -e '/server-id.*1/ {
s/#//
}' -e '/# binary logging is required for replication/ {
n
s/#//
}' $MYCNF 2> /dev/null > /tmp/.`basename $0`.$$.tmp

    mv /tmp/.`basename $0`.$$.tmp $MYCNF
    chown mysql:mysql $MYCNF
    rm $MASTERINFO

    $FAILOVER_MYSQL_CMD restart > /dev/null 2>&1
    $FAILOVER_PERFPARSE_CMD start > /dev/null 2>&1
    $FAILOVER_NAGIOS_CMD start > /dev/null 2>&1

    touch $FAILOVER_OCCURED
```

```
    mv  $FAILOVER_NAGIOS_CMD  `echo  $FAILOVER_NAGIOS_CMD  |  sed  -e
's/.disable//'`
}

heart_beat() {
    if [ -f $FAILOVER_OCCURED ] ; then
        return
    fi

    alive=`ssh  -o  "ConnectTimeout  $RETRY_INTERVAL"  -l  $SSHUSER  -i
$SSHIDENTIFY $MASTER "$MASTER_NAGIOS_SEARCH_CMD" 2> /dev/null`
    if [ -z "$alive" ] ; then
        cnt=`expr $1 + 1`
        if [ $RETRIES -gt $cnt ] ; then
            sleep $RETRY_INTERVAL
            heart_beat $cnt
        else
            activate
        fi
    fi
}

# entry point.
init
logcopy
heart_beat 0
```

1 分間隔で cron から実行させます。

```
nabira# cat /etc/crontab
  ...
*/1 *   *   *   *   root    /usr/local/bin/slavetools4nagios
  ...
```

このスクリプトの基本ロジックは、マスタへ ssh 接続を行なって Nagios のステータスをチェックするプラグインを実行するというものです。Nagios のステータスに異常を検出すると再試行を行い、なお異常が認められる場合にフェイルオーバを実施します。

slavetools4nagios スクリプトはコンフィギャファイル slave2master.conf を参照しますので、このファイルに適切なパラメータを設定します。

```
nabira# cat /usr/local/etc/slave2master.conf
  ...
# base.
MASTER=yasu
SLAVE=nabira

# HA account information.
SSHUSER=hb
SSHIDENTIFY=/home/hb/.ssh/id_dsa
SSHOPTS="ssh -l $SSHUSER -i $SSHIDENTIFY"

# Nagios event log.
RSYNC=/usr/local/bin/rsync
SRC_NAGIOS_CURRENTLOG=/var/spool/nagios/nagios.log
SRC_NAGIOS_ARCHIVESLOG=/var/spool/nagios/archives/
DST_NAGIOS_CURRENTLOG=/var/spool/nagios
DST_NAGIOS_ARCHIVESLOG=/var/spool/nagios/archives

# Nagios failover.
FAILOVER_OCCURED=/var/spool/nagios/failover.lock
FAILOVER_NAGIOS_CMD=/usr/local/etc/rc.d/nagios.sh.disable
FAILOVER_PERFPARSE_CMD=/usr/local/etc/rc.d/perfparse.sh
FAILOVER_MYSQL_CMD=/usr/local/etc/rc.d/mysql-server.sh
FAILOVER_OCCURED=/var/spool/nagios/failover.lock
```

```
RETRIES=3
RETRY_INTERVAL=10
MASTER_NAGIOS_SEARCH_CMD="/usr/local/libexec/nagios/check_nagios_db.pl"
NOTICE_CMD="/usr/local/bin/notice.pl"
MYCNF=/var/db/mysql/my.cnf
MASTERINFO=/var/db/mysql/master.info
  ...
```

スレーブのフェイルオーバは、以下の処理が行なわれます。

- フェイルオーバ発生の通知メールを送信する。
  notice.pl スクリプトを流用します。

- レプリケーションを停止してマスタとして再起動する。
  レプリケーション機能を無効にし、マスタとして動作するように my.cnf を編集します。

- Nagios を起動する。
  起動後は、Nagios の起動スクリプトファイル名が nagios.sh.disable から nagios.sh に変更されます。

- 元マスタの状態に関知しない。
  概念的にはフェイルオーバが発生すると nabira がスレーブからマスタに昇格したことを意味しており、実装として touch コマンドでフェイルオーバが発生したことを示すフラグファイルを作成します。slavetools4nagios スクリプトの logcopy、heart_beat 関数はフラグファイル failover.lock を走査します。

  nabira がスレーブ状態で OS をリブートした場合には、MySQL はレプリケーションを継続し、Nagios は起動しません。しかし、一度フェイルオーバが発生すると nabira はマスタとなり、この状態で OS をリブートすると MySQL はレプリケーションを行ないませんし、Nagios が起動されるようになります。

  再度スレーブのセットアップを行なう場合には、failover.lock ファイルを手動で削除する必要があります。あとは、本章の手順に従えば問題ないはずです。

5. まとめ

5.1. 総括

本書で言及している Nagios 冗長化検証について、以下のようにまとめることができます。

まとめ

特性	内容
冗長構成	マスタ(現用系) - スレーブ(待機系)。 Nagios のポーリング、および通知機能はマスタでのみ動作する。
フェイルオーバ時の相互作用	マスタの Nagios 情報すべてと PerfParse 情報すべてがスレーブに引継がれる。 たとえば、ある CRITICAL インシデントがオープンしている状態もスレーブに引継がれる。このため、フェイルオーバが発生した後で、この CRITICAL インシデントに対する通知は発生しない。 よって、CRITICAL/WARNING/UNKNOW に対するインシデントと RECOVER インシデントは常に 1 対 1 で処理される(ログ監視やトラップ監視などは除く)。
ミラーリング手法	MySQL レプリケーション、rsync。
ハートビートプロトコル	独自のシェルスクリプトで実装。 スレーブからの一方向ハートビートチェックが行なわれる。マスタは冗長構成に関する一切の情報を持たず、スレーブだけが障害に気が付く。 マスタ Nagios のプロセス死活を監視し、かつ Nagios がアクティブであることを示す足跡を走査する。具体的には programstatus テーブルの last_update 列にタイムスタンプが更新されるので、この列を参照し過去 5 分以上更新されていなければ、非アクティブとみなす。 このスクリプトは cron により 1 分間隔で実行される。 マスタ障害の検出からフェイルオーバの発生に要する時間は最大 2 分となる。この間が総合的な監視サービスのダウンタイムと考えられる。
フェイルバック	直接的なフェイルバックは不可能。 状態としてフェイルオーバのみ発生し、それにより各ノードのマスタ - スレーブとする機能の昇格･降格が循環される。
未検証事項	MySQL レプリケーションの SSL 認証。

Nagios redundancy material

改版履歴

Version 1.0	2005/07/09	新規作成。

製作

Isidore.

本書は 2005 年 7 月現在の情報を元に作成されております。本書に記載されております内容は、許可なく変更されることがあります。