ミッションクリティカル向け syslog 改善機能の提案と実装

Proposal and implementation of syslog improvement method for mission critical

細内 昌明* 山崎 謙太* 森田 滋** 中島 藤夫** 大辻 彰*** 三瓶 英智***
Masaaki Hosouchi Kenta Yamasaki Sigeru Morita Fujio Nakajima Akira Ootsuji Hideaki Sanpei

*株式会社日立製作所 システム開発研究所 ***株式会社日立製作所 ソフトウェア事業部
**日立ソフトウェアエンジニアリング株式会社

syslog の記録確実性・関数応答時間安定性の改善手法を提案する。Linuxのミッションクリティカルシステム適用支援のためのトラブルシュート機能強化の一環として，syslog をミッションクリティティカルアプリケーションから利用するための要件である記録確実性・関数応答時間安定性を改善する方法を考案し，拡張SYSLOGとして実装・評価した。glibc の syslog 関数に対してエラーチェック処理を強化し，syslogd 未起動・メモリ不足・長大メッセージの欠落等のログメッセージ喪失を防止し記録確実性を向上した。ソケットのノンブロッキング化とソケットパスの選択インタフェース提供により，他プログラムによる高頻度ログ出力が生じても，リトライ率を1％に抑えつつ最大関数応答時間を従来の 200ms から 1.2ms に改善し，関数応答時間安定性を向上した。ログメッセージの文字コード変換機能提供により，ログファイル監視機能の確実性を向上した。

１． はじめに

1.1 動機

Linux コミュニティによる性能・信頼性改善の積み重ねの結果，メインフレームの独壇場であったミッションクリティカルな基幹システムに対しても，Linux を適用する事例が出てきている。

稼働率 99.999%以上の高可用性や高信頼性が求められるミッションクリティカルシステムでは，プログラム不良やハード障害の発生率低減・早期発見・早期通報・解決時間短縮の有効な手段である障害情報収集手段が重要な役割を占める。

障害情報収集手段の要件の1つに，記録確実性がある。記録情報を喪失すると，解析や検証が不十分となる。特に再現性がない不良では影響が大きい。また，障害情報を契機とした障害通報の確実性も求められる。

記録性能も重要である。障害・稼動情報記録によるアプリケーション処理時間の悪化は，一定時間内の処理が要求されるオンライントランザクションやリアルタイム処理では特に影響が大きい。記録手段の処理を一定時間内に終了させる応答時間の安定性が求められる。

Linux のミッションクリティカル対応強化を図るには，Linux 上の障害情報収集手段に対しても，これらのミッションクリティカル向け要件を満たす必要があると考えた。

1.2 syslog の概要

Linux上の障害情報収集手段の1つに，syslogがある。syslogは，アプリケーションプログラムの稼動状況や障害情報をログメッセージとしてログファイルに記録するプログラム[1]であり，Linuxのロギング方法のデ・ファクト標準である[2]。代表的なものに，glibc(GNU C Library)の共有ライブラリに含まれるsyslog関数群と，sysklogdパッケージに含まれるsyslogdデーモンがある。

syslog関数群は，syslogdへの接続を行うopenlog関数や，ログメッセージを組み立てて syslogd に送信する syslog 関数などから構成される。syslogd は，syslog 関数からログメッセージを受信しログファイルに記録する常駐プログラムである。

syslogでの記録には障害情報統合による利点がある。すべての障害・稼動情報をアプリケーション固有のファイルに記録していては，ログ収集ツールを使っても，障害直前の情報が固有ファイルに残っている場合があり，障害情報を参照するだけで時間がかかってしまう。これに対して，障害発生や障害要因を判別可能な情報を syslog に記録することで，単一のログファイルを参照するだけでこれらの情報を時系列に入手できる。このため，初動調査時間が短縮され，障害監視オーバヘッドが低減する。

1.3 目的

syslog には上記のような利点があるが，障害情報収集手段の要件である記録性能や確実性において，問題や懸念がある。記録性能では，ログ高頻度出力時の性能低下（syslog 関数の待ち発生）問題が報告されている。記録確実性では，リモートロギング以外のログ喪失危険性の検証が不十分である。

本報告では，syslog におけるログメッセージの記録確実性・通報確実性の検証および向上と，関数応答時間安定性の向上を目的とする。ミッションクリティカルアプリケーションからのログメッセージ出力に適したロギング機能提供を目指す。

1.4 拡張 SYSLOG の概要

記録確実性・通報確実性の向上と関数応答時間安定性というミッションクリティカル向け要件を満たすsyslog 関数として，拡張 SYSLOG を実装した。

拡張SYSLOG は，glibcに含まれるsyslog関数処理のソースコードをベースとしており，高信頼ログ基盤HA Logger Kit For Linux[3]の一機能としてソース公開中である。拡張SYSLOGの外部仕様と機能構造の概要を以下に示す。

(1)外部仕様

拡張 SYSLOG が提供する関数の仕様を以下に示す。

```
int _hopenlog (const char *ident, int option, int facility, const char *socket);
  int _hsyslog (char *ctype, int priority, const char *format, ...) ;
  void _hcloselog (void) ;
```

下線部分が，glibcのsyslog関数と異なる仕様である。従来のopenlog関数に相当する_hopenlog関数では，デーモンと通信するUNIXドメインソケットのパス名を指定する第四引数socketを追加した。また，関数の成功時に0を，失敗時に-1 を返却するため，戻値(リターンコード)の型をintとした。

従来の syslog 関数に相当する_hsyslog 関数では，戻値の型変更の他に，メッセージの文字コードを指定する第一引数 ctype を追加した。

(2)機能構造

拡張 SYSLOG の概要図を図1に示す。拡張 SYSLOG は，共有ライブラリとして提供する。ミッションクリティカルアプリケーションは拡張 SYSLOG が提供する関数を使用し，エラーを返却したときのリトライ処理などの高信頼化処理を追加コーディングする。拡張 SYSLOG は syslog 関数のみを改善対象としているため，syslogd は sysklogd などの既存機能を使用する。

拡張 SYSLOG は，従来の syslog 関数の共有ライブラリを完全に置換するものではない。完全に置換してしまうと，既存のアプリケーションでログメッセージ喪失が発生してしまうためである。拡張SYSLOG 関数は応答時間を考慮して，syslogd が受信処理しきれない状況では待たずにエラーを返す。しかし，従来の syslog 関数を利用していたアプリケーションでは，syslog 関数のエラーチェックを行っていない。このため，置換してしまうと，受信処理しきれない状況のときに喪失となる。

拡張 SYSLOG と従来の syslog との共存のため，拡張 SYSLOG が syslogd に送信するログメッセージの形式の互換性は保った。また，syslogd が glibc の syslog 関数からのログメッセージも拡張SYSLOG のログメッセージも混在して受信・出力可能であることは，実験とソースコード検証により確認した。

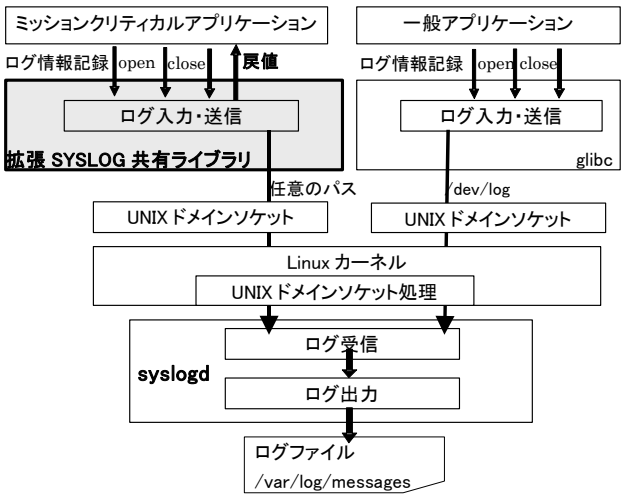

図1 拡張 SYSLOG の概要

1.5 論文の流れ

本論文の流れは，以下のとおりである。まず，2章ではsyslog改善に関する動向を紹介する。3章では，従来の syslog の記録確実性検証結果と拡張 SYSLOG で実装した記録確実性向上処理の概要を示す。4章では，拡張 SYSLOG で実装した記録オーバヘッド低減と記録確実性の両立方式の概要と評価結果を示す。5章では，拡張 SYSLOG で実装した通報確実性向上方法を示す。最後に6章で論文のまとめを示す。

2. 関連研究・プロジェクト動向

syslog関連の動向として， RFC-3195[4]を実装したrsyslogや，syslog-ng[5]などの代替syslogが開発・公開されている。これらの代替syslogは，sysklogdパッケージのsyslogdの問題点であるシステム間転送時におけるログメッセージ喪失発生を，TCPプロトコルによる転送によって改善している。

しかし，代替 syslog は，すべてデーモン側を対象としており，syslog 関数側の処理は含まれていない。また，ホスト間のログ転送に関する改善が中心であり，同一ホスト内のログ記録に対する改善はされていない。glibc に

含まれる syslog 関数処理も，マルチスレッド対応程度の改善しかされていない。

syslog に関する研究は，改ざん防止や検索・解析などはあるが，信頼性に関する報告は見られない。

３.ログ喪失条件の検証と喪失対策手法

３．１ ログ記録確実性の課題と検証対象

ログ喪失により障害要因分析が不十分となりえるため，syslog 関数には記録確実性が求められる。

syslog のログ喪失は，他ホストへのログ転送を伴うリモートロギングにおいて報告されている。リモートロギングのログ喪失は，TCP プロトコルをサポートする代替 syslog によって対策がなされている。

一方，他ホストへの転送を伴わないローカルロギングでは，ログ喪失の報告はない。しかし，ログ喪失がないという検証報告もない。通常ケースでの喪失がなくても，リソース不足など再現性に乏しいケースでログ喪失が発生してしまうと，不良対策が十分にできず，ミッションクリティカルレベルの信頼性を確保できない。syslog 関数はエラー情報を返さないため，ログ喪失が発生した場合に呼び出し元アプリケーション側でリトライ処理などの喪失低減対策を実装することもできない。

このため，以下に示す①～④のソースコード解析により syslog 関数ローカルロギングにおけるログ喪失ケース(syslog 関数呼出元が出力失敗を認識できないのにログメッセージがログファイルに出力されないケース)を洗い出した。

処理① glibc-2.5.0 に含まれる syslog 関数処理
処理② linux2.6.19.2 に含まれる UNIX ドメインソケットの UDP 送信処理
処理③ linux2.6.19.2 に含まれる UNIX ドメインソケットの UDP 受信処理
処理④ sysklogd-1.4.1 に含まれる syslogd の処理

検証結果の詳細は３．２章で報告するが，ローカルロギングにおいても，リソース不足時などいくつかのケースでログ喪失が認められた。このため，３．３章に示すログ喪失対策を実施した。

３．２ 検証結果

ソースコード検証により，他計算機への転送を伴わない場合であってもログメッセージを喪失してしまうケースを洗い出した。喪失ケースを以下に示す。

なお，ログメッセージを高頻度に出力した場合は，syslog 関数ではブロッキングモードに設定しているため，backlog の上限または送信バッファ最大長を超えても，上限に収まるまで待ち続けるだけであり，ログメッセージ喪失は発生しない。

(1) syslogd 停止時（処理①）

syslogd が起動されていない状態で syslog 関数を呼び出した場合，syslog 関数内で発行する connect 関数や send 関数がエラーとなる。syslogd が起動されていない状態となりうるケースを以下に示す。

(a) システムを起動してから一度も syslogd を起動しなかった場合
(b) syslogd が停止した場合
(c) syslogd が異常終了した場合
(d) shutdown 中 syslogd を停止させてから syslog 関数を発行した場合

send 関数エラーの場合，内部で一度だけリトライが行われるが，通常はリトライでも状況はかわらない。リトライでもエラーであれば，そのまま終了するので，ログメッセージはログファイルに出力されず失われる。

(2) send 関数エラー時（処理①）

UDP 送信ソケットがクローズされたときに，送信バッファ中に残っている未受信のログメッセージは解放され喪失する。UDP のため，syslogd の受信を syslog 関数で確認することがなく，ログメッセージがすべて送信完了になるまでクローズを延期することもしないからである。

send 関数がエラーとなった場合，一旦ソケットをクローズし，再接続後リトライする。このため，send 関数がエラーとなると，呼び出した syslog 関数に指定されたログだけでなく，直前の syslog 関数のログまでがソケットクローズにより喪失されてしまう。

send 関数がエラーとなる典型的なケースは，openlog 実行後に syslogd を停止させたり syslogd が異常終了した場合である。

(3) 割込発生時（処理①）

syslog 関数から呼び出す send 関数の処理中にシグナルを受信した場合，シグナルハンドラ処理後の動作が「再実行」でなければ，メッセージを送信バッファに格納する前に send 関数が中断され，EINT エラーとなる。syslog 関数は send 関数がエラーの場合一度切断するため，送信バッファ中に残っている未受信のログメッセージを喪失する。

Linux のデフォルトは「再実行」であるが，呼び出し元のプログラムでシグナルハンドラを設定している場合など「再実行」でない場合がありえる。

(4) メモリ不足時（処理①）

以下のメモリ割当失敗のケースにおいて，呼び出し元の syslog 関数にてエラーチェックをしていないため，ログメッセージ喪失が発生する。

(a) syslog に出力するメッセージを格納するメッセージバッファの割当関数 open_memstream が失敗した場合。syslog 関数は，コンソールまたは標準エラーへの追加出力が指定された場合は，スタック内の静的バッファを使わずに動的にバッファを割り当てる。かわりにスタック内静的バッファを利用してエラーメッセージ”Out Of memory”は出力されるが，ログメッセージは無視されてしまい喪失する。
(b)メッセージ組み立てに利用する snprintf 関数内での作業用バッファの割り当てに失敗した場合。syslog 関数は，snprintf 関数の戻値が負値の場合は何もしないため，喪失する。
(c)send 関数内で送信バッファ確保に失敗した場合。send 関数は ENOBUF エラーを返すが，syslog 関数では一度再実行するだけであるため，2度目の send 関数が失敗すればログメッセージ喪失となる。

(5)長大メッセージ送信時(処理①(関連②④))

ヘッダまで含めたログメッセージのサイズが send 関数の送信バッファ最大長(32KB)を超える場合，send 関数は EMSGSIZE エラーとなり，ログメッセージ喪失となる。

また，32KB 未満でも 1024 バイト以上であれば，syslogd の受信バッファ長を超えるため，1024 バイトを越える部分が欠落して記録される。

通常のログメッセージは 100 バイト前後と想定され，通常使用の範囲では欠落は起こらないと思われる。しかし，書式文字列に埋め込む変数の文字列が不当な場合などがありえる。欠落したことがわからないと，こうした不良の早期摘出が難しい。

(6)ログファイル出力失敗時(処理④)

syslogd において，ボリューム障害や容量不足等によりログファイルのオープン関数や出力関数が失敗した場合，ログファイルには出力されず，ログファイルを見てもメッセージが喪失したことはわからない。

syslog 関数がメッセージを送信しても，syslogd は応答を返さないため，syslog 関数やその呼び出し元アプリケーションは，syslogd のエラーを検出できない。

3.3 実装

syslog 関数がログメッセージ出力失敗を判断できる戻値等の情報を呼出元に提供しないため，上記のような喪失要因が発生しても，呼出元は見逃してしまい喪失となる。出力失敗がわかれば，呼出元アプリケーションで以下のような実装を追加することで，ログメッセージ喪失削減が可能である。

(a) 再度 syslog 関数を発行する。
(b) 別のスレッドを用いてバックグラウンドでログ出力を試みる。
(c) メッセージを残して出力を延期する。
(d) 別のファイルに出力する。
(e) ログ出力に失敗したという情報だけは残し，出力が可能になったときに少なくともログ出力に失敗したことでけはわかるように，出力失敗情報をログ出力する。

このため，拡張 SYSLOG では，syslog 関数処理に以下のエラー検出処理を追加した。エラーを検出した場合，戻値－1とエラー理由 errno を呼出元に返却する。
(1) メッセージバッファ割当・拡張を行う open_memstream 関数が失敗した場合
(2) メッセージ組み立てを行う snprintf 関数が失敗した場合
(3) 組み立て後のメッセージが 1024 バイトを超えている場合
(4) デーモン未起動等により，connect 関数が失敗した場合
(5) デーモン異常終了やカーネル内メモリ不足などにより，send 関数が失敗した場合。また，失敗の理由が割込(EINTR エラー)や送信バッファ満杯(EAGAIN エラー)のときは，送信バッファ中のログの喪失を防ぐため，ソケットはクローズしない。

これらのエラー検出処理追加により，3.2 章の喪失要因のうち，以下の喪失要因については，少なくとも失敗したことを呼出元に通知することが可能となった。
(1)の(a)(b)(c) syslogd 未起動や停止・異常終了後のログ出力
(3) syslog 関数処理中の割込発生(ただし，後述のソケットノンブロッキング化により発生しなくなると思われる)
(4) メモリ不足
(5) 長大メッセージ

上記改善でも以下の喪失要因は検出できない。
(1)の(d) shutdown 中の syslogd 停止後に出力したログ
(2) syslogd の停止/異常終了等でソケット切断時に送信バッファに残っていたログ
(6) ログファイル出力失敗上記改善によっても喪失が発生するケースへの対応

ケース(1)(6)における喪失を防止するには，共有メモリ等のソケットが切断されても残る領域にログメッセージを保存しログファイルに出力してから消去する対策が必要となる。ケース(3)(d)のメッセージを保存するには，ファイル以外の手段に保存する必要がある。

４.syslog関数応答時間安定性向上手法

４.１ syslog 性能の課題

基幹業務におけるオンライントランザクション処理では，数 10ms オーダの時間内に処理を完了することが要求される。このようなミッションクリティカル処理やリアルタイム処理から呼び出される関数には，一定時間内に処理を完了することが求められる。syslog 関数の実行に割ける時間はせいぜい 1ms 程度が上限であろう。

しかし，短時間に syslog 関数を多数回実行して backlog の制限を越えたり，多量のログメッセージを送信して送信バッファ量の上限を超えたりした場合，現状の syslog 関数処理では，syslogd が処理して上限に収まるまで待ち状態となる。このため，高頻度のログ出力により syslog 関数の応答時間が悪化し，トランザクションが想定時間内に終わらない場合が生じることが懸念される。

特に，他プログラムによる高頻度ログ出力の影響が懸念される。ミッションクリティカルアプリケーションでは，事前に性能見積もりを行い，一定以上のログメッセージを出力しないように事前に出力量を絞る対策をとることが多い。しかし，他プログラムのログ出力の影響までは見積もるのは困難である。ミッションクリティカルアプリケーションと同一ホスト上で動作させた性能見積もりなどを行わないアプリケーションなどから，誤ってまたは意識的に高頻度にログを出力するケースが生じないとも限らない。

syslog 関数は，どのプログラムからも発行可能であり，認証・権限を必要としない。また，ファイルを1つに集約したほうが障害時の情報参照がしやすいため，デーモンやログファイルはプログラム間で共有する。このため，上記のような他プログラムによる高頻度出力が発生しやすい。

また，高頻度出力による応答時間悪化の対策として，ファイルシステム同期の省略がある。Linux 標準の sysklogd パッケージに含まれる syslogd のデフォルトの設定では，ログ記録直後のシステム障害や停電等によるログメッセージの喪失を防ぐため，ログメッセージを出力するごとに fsync 関数を発行してログメッセージのバッファキャッシュをストレージデバイスと同期させる。同期によりログメッセージごとに I/O 完了待ちが発生するため，単位時間当たりの出力メッセージ数を一定以上あげることができない。このため，より多くのメッセージが出力できるように fsync 関数を省略することができる。sysklogd では，設定ファイル syslog.conf に記述する出力先ログファイルのパス名にマイナス記号”-”を接頭することで，すべての fsync が抑制される。同期するメッセージ数間隔や時間間隔の設定が可能な syslogd もある。しかし，この方法では，システムダウンや停電等にログメッセージを失う危険性が増大してしまう。

４.２ 応答時間安定性向上手法の概要

上記性能課題の対策手法として，送信待ちブロックの解除による応答時間改善と，一定時間内の処理が要求されるアプリケーションのログメッセージの通信路分離・優先処理による，応答時間改善に伴う送信エラー発生の頻度低減を提案する。拡張 SYSLOG で実装した関数仕様および処理の改善方法を以下に示す。

（1） ソケットのノンブロッキングモード設定による送信待ち時間削減

syslogd とのログメッセージ送信に利用する UNIX ドメインソケットを，ブロッキングモードからノンブロッキングモードに変更する。これにより，送信バッファ満杯等によりメッセージをソケットの送信バッファに入れることができない場合，待たずにエラー終了する。send 関数の戻値をチェックして戻値を返す仕様としたため，送信バッファ満杯エラーであることを呼出元アプリケーションがチェックし，3.3 章で示したような対応をとることができる。

（2） ソケット選択インタフェースの提供による他プログラムによる高頻度出力影響の低減

syslogd とのログメッセージ送信に利用する UNIX ドメインソケットのパス名を，アプリケーションから選択可能とした。パス名は，_hopenlog 関数の引数に指定する。

syslogd はすでに複数の UNIX ドメインソケットから受信可能になっている（chroot 環境での syslog 出力対応のため）。このため，syslogd の処理は変更しない。syslogd の起動コマンドにオプション a を指定すると，従来の/dev/log からのログメッセージと，アプリケーションが指定したソケットからのログメッセージの両方を受信・記録できる。

ソケット選択の狙いは，他アプリケーションによる高頻度出力影響の低減である。syslogd は，複数のソケットが指定された場合，すべてのソケットを指定して select で受信待ちを行う。各ソケットのログメッセージは，交互に受信・処理される。このため，あるアプリケーションから/dev/log ソケットに対して高頻度にログを出力しても，ソケットを分離したアプリケーションのログメッセージは，ログメッセージの受信待ちキューに埋もれることなく処理される。これにより，backlog 上限による送信エラーが減少し，リトライ頻度も減少する。

ミッションクリティカルアプリケーションでは拡張 SYSLOG を使用し，syslog のソケットと異なるパスを指定することで，安定した関数応答時間を確保しつつ，エラー頻度を抑えることができる。不良や悪意のあるアプリケーションが syslog を高頻度に出力しても，処理時間への影響を抑えることが可能となる。なお，悪意のあるアプリケーションがミッションクリティカルアプリケーションのソ

ケットと同じパスを使用しないように，ソケットには適切な権限を設定しておくことを推奨する。

4.3 評価

上記に示した高頻度ログ出力時の関数応答遅延防止効果検証のため，高頻度ログ出力時の拡張 SYSLOG 関数の応答時間（関数終了時刻と関数開始時刻との差）と関数エラー率(_hsyslog 関数が EAGAIN エラーで終了する割合)を測定し，従来の syslog と比較した。測定環境と測定方法を示す。

測定環境

CPU	Athron1800+
OS	Fedra Core6

測定方法

(a) _hsyslog 関数または syslog 関数を発行して，ログメッセージの平均長と想定される 100byte のメッセージを送信し，関数応答時間を求める。_hsyslog 関数の場合は，関数の戻値を参照しエラーの有無も求める。

(b) syslogd の設定は，-a オプションの追加指定以外はデフォルトの設定とする。メッセージごとに fsync が発生する。

(c) 上記処理を，待ち時間（nanosleep 関数実行）をはさんで，1000 回繰り返し実行する。

(d) 関数応答時間の平均値および最大値を求める。_hsyslog 関数の場合は，関数エラー率も求める。

(e) 誤差を考慮して上記測定を3～5回繰り返し，平均値を求める。

(f) 待ち時間（nanosleep 関数の引数）を変更して関数実行間隔による推移を調べる。実行間隔の測定範囲は，トランザクションあたり1件のログが出力されるものと想定し，現在の基幹業務システムにおけるオンライントランザクションのピーク上限である数千件/秒を中心に，その 1/10～数 10 倍の範囲で測定した。

まず，同一プログラム内での高頻度ログ出力時の関数応答時間を測定した。測定結果を図2～4に示す。

従来の syslog 関数では，5ms 以下の間隔で発行すると応答時間が最悪 10ms 以上に悪化する。拡張 SYSLOG 関数では，応答時間は最悪でも 2ms 未満に収まるが，かわりに，1ms 以下の間隔になると関数エラー率が増大する。

拡張 SYSLOG を使用するアプリケーションは，ログ出力頻度を抑える設計をするミッションクリティカルプログラムに限定されるため，1ms 以下の間隔で出力されることはまれである。どうしてもログ出力量が多くなってしまう場合の対策方法としては，障害時のログメッセージ喪失危険性を低減しながら性能をあげる手法として，重要でない(システムダウンによりメッセージが失われても影響が少ない)メッセージを同期対象からはずなどの方法を提案する。優先順位やソケット識別子等の手段により，呼び出し元アプリケーションから同期するメッセージを制御できるのが望ましい。

また，一定時間内の関数応答とエラー率低減を両立する他の方法として，アプリケーションが指定した送信タイムアウト時間をソケットに設定する方法が考えられる。

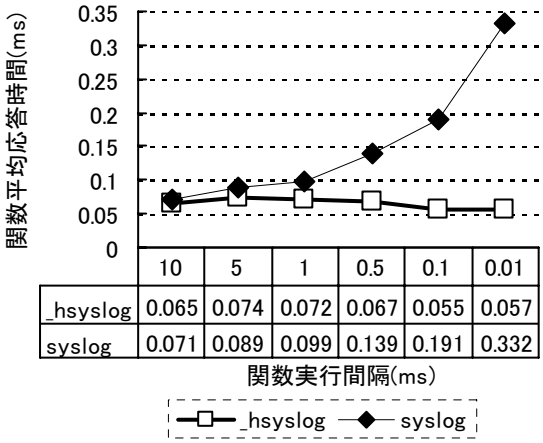

図2 関数の平均応答時間

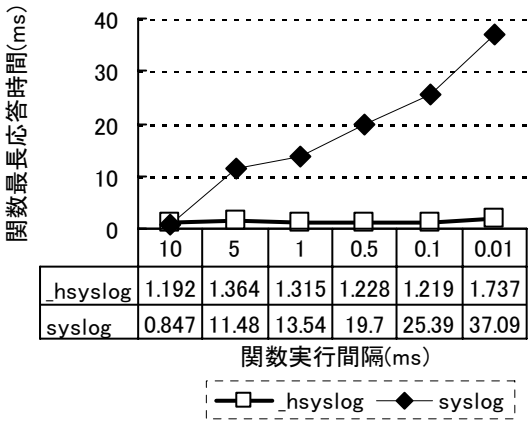

図3 関数の最長応答時間

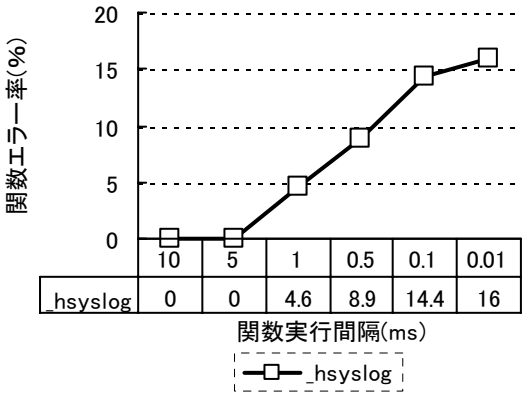

図4 関数エラー率

次に，他プログラムの高頻度ログ出力による関数応答時間への影響を調べた。syslog 関数を一定間隔で実行してログメッセージを高頻度に出力するプログラム A と，syslog 関数または_hsyslog 関数を 10ms 間隔で実行する影響測定対象のプログラム B とを同時に実行した。プログラム A の実行間隔を変えて，プログラム B の関数実行時間とエラー率を測定した。測定結果を図５～7に示す。

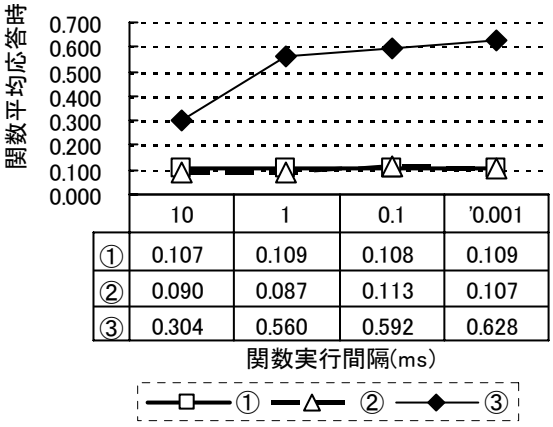

図5 他 AP 高頻度出力時の平均応答時間

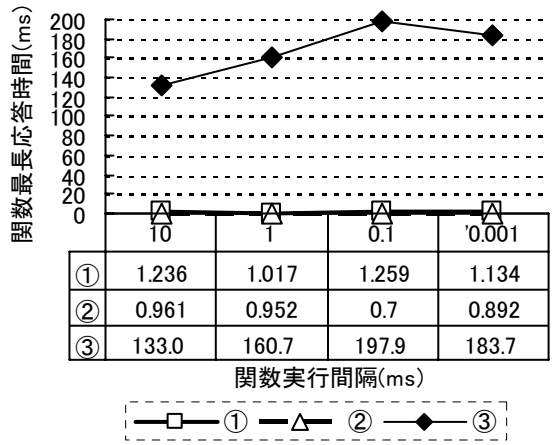

図6 他 AP 高頻度出力時の最長応答時間

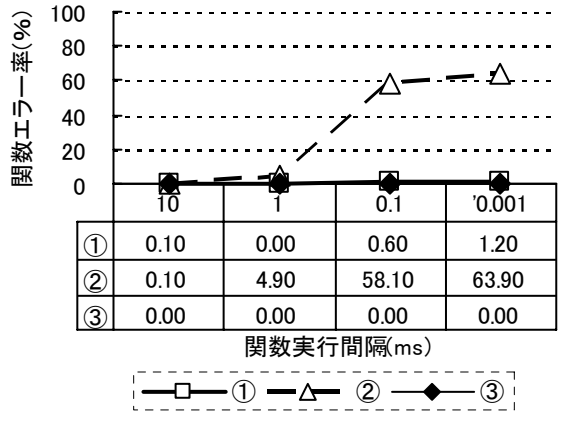

図7 他 AP 高頻度出力時の関数エラー率

図において，ケース①はプログラム B が_hsyslog 関数を使ってプログラム A と異なるソケット/dev/log2 に出力した場合の結果である。ケース②は，プログラム B が _hsyslog 関数を使ってプログラム A と同じソケット /dev/log に出力した場合の結果である。ケース③は，プログラム B が syslog 関数を使った出力した場合（ソケットはプログラム A と同一）の結果である。

図6をみると，syslog 関数で出力したケース③では，プログラム A の出力頻度上昇に伴って関数応答時間が悪化し，最長で 200ms に達する。一方，_hsyslog 関数で出力したケース①②の関数応答時間は最長でも 1.2ms と安定しており，他プログラムによる高頻度出力の影響を受けにくいといえる。

図7をみると，ケース②の場合はプログラム A の出力頻度上昇に伴ってエラー率が悪化するが，ケース①ではパス分離の効果により，最大でも1％程度のエラーにとどまる。この程度であれば，アプリケーションによるログメッセージ喪失対策でカバー可能と思われる。

５.通報確実性向上手法

(1) 課題

Linux の標準文字コードは UTF-8 であるが，ミッションクリティカル業務で使われることの多い COBOL 言語では，行内のオフセットを意識するため，可変長文字やシフトコードがある文字コードを扱いにくい。このため，固定長文字コードかつシフトコードがない Shift-JIS で記述されることが多い。

このようなプログラムの制限や，ユーザによる環境変数設定によって UTF-8 以外の文字コードに設定している場合，ログファイルにも UTF-8 以外の文字コードで出力される。その結果，ログファイルに複数の文字コードのログメッセージが混在し，ログファイルを表示するときに文字化けしてしまう。

文字化けより問題が大きいのが，ログメッセージを契機としたイベントの誤動作である。ログファイル監視機能では，定期的に読み込んだログファイルから事前に登録した文字列を検索して，文字列の内容に対応したイベントを発生させる。このため，ログメッセージの文字コードが異なると，登録された文字列と一致せず，イベントが発生しなくなってしまう。

(2) 提供仕様

拡張 SYSLOG では，プログラムが事前に文字コードを変換しなくても，イベントの誤動作を防止可能とするため，ログメッセージの文字コード変換機能を提供する。

syslog 関数の引数に，変換前のログメッセージ（引数に指定した書式文字列と埋め込む変数内の文字列）の文字コードを指定するインタフェースを追加した(1.3 章

参照)。拡張 SYSLOG では，指定された文字コードから UTF-8 に変換してからメッセージを syslogd に送信する。

変換前の文字コードの指定方法として，採用した引数による明示指定方法のほかに，ロケールや環境変数による暗黙指定方法も考えた。しかし，ログメッセージごとにコードが異なる場合に関数を実行する必要があり記述量が多くなること，ユーザの設定などにより文字コードが誤認される危険性があること，スレッド共通であるロケールや環境変数を利用するとマルチスレッド環境でのログメッセージ出力が複雑になること，を懸念し，採用しなかった。

(3) コード変換ライブラリの初期化方法

文字コード変換には，外字が必要であったため日立コード変換が提供するライブラリを使用したが，試作では iconv で動作確認しており，オープンソースベースでも実現可能である。文字コード変換ライブラリのロードと初期化は，特に文字コード数が多くなったときのコード変換テーブルによるメモリオーバヘッドを考慮し，openlog 時点ではなく，コード変換がはじめて必要となったときに実施する。

(4) コード変換範囲

以下のログメッセージ形式

```
<n> Mmm dd hh:mm:ss HostName ident[PID]: message
①  ②  ③     ④        ⑤        ⑥     ⑦     ⑧
```

のうち，メッセージ本体⑧のみをコード変換する。ヘッダ部の①～⑦はコード変換しない。

⑤の HostName や⑥の ident にはホスト上の設定や _hopenlog 関数の引数の指定によっては日本語メッセージが含まれる可能性がある。これらの文字コードもログメッセージ本体の⑧とともに文字コード変換してしまうと，同じホスト名であっても異なる文字コードでログファイルに記録されてしまい，ログメッセージを収集・解析するときに，異なるホストからのメッセージと誤認されてしまう可能性がある。このため，文字コードの変換範囲は，⑧の部分のみとした。

なお，glibc の syslog 関数ではタイムスタンプ取得に strftime 関数を使用していたので，ロケールによっては日本語のタイムスタンプとなりタイムスタンプが 2 重に出力されるなどの現象が生じた。このため，ロケールに依存しない ctime 関数に置換し，常に英語のタイムスタンプで出力するようにした。

6.まとめ

Linux のミッションクリティカル機能強化のため，glibc の syslog 関数に対するミッションクリティティカルアプリケーション向け改善手法を提案し，拡張 SYSLOG として実装・評価した。その結果，以下の記録確実性・関数応答時間安定性・通報確実性の向上を実現した。

(1) ログメッセージの記録確実性向上

エラーチェック処理追加により，従来ログメッセージ喪失の要因となっていた以下のケースについて，エラー情報を返却することで，アプリケーションでの再送や別手段への出力対策を可能とした。

(a) syslogd 未起動
(b) メモリ不足によるバッファ獲得失敗・関数エラー
(c) 長大ログメッセージの一部または全体の欠落
(d) send 関数処理中の割込(ソケットクローズによる未送信メッセージの解放防止も含む)

(2) 関数の応答時間安定性向上

syslogd へのログメッセージ転送に使用する UNIX ドメインソケットをノンブロッキングモードに変更し，ソケットのパスをアプリケーションごとに選択可能とした。その結果，他アプリケーションによるログ高頻度出力時の関数応答時間悪化を改善した。リトライ率を1%に抑えつつ最大関数応答時間を従来の 200ms から 1.2ms に，平均でも 0.6ms から 0.1ms に削減した。

(3) ログファイル監視機能の通報確実性向上

ログメッセージの文字コードを UTF-8 に変換する機能を実装した。ログファイル中の文字コードが統一されるため，ログファイル中のメッセージ文字列をトリガとして動作するログファイル監視機能の誤動作が減少する。

参考文献

[1] “The Ins and Outs of System Logging Using Syslog”, Ian Eaton

[2] “The BSD syslog Protocol”, RFC3164, C.Lonvick, August 2001

[3] “高信頼ログ基盤 HA Logger Kit for Linux”, http://www.hitachi.co.jp/Prod/comp/linux/download/oss/files/log_0801.pdf
http://www.hitachi.co.jp/Prod/comp/linux/download/oss/index.html

[4] “Reliable Delivery for syslog”, RFC3195,D. New, M. Rose, November 2001

[5] http://www.balabit.com/network-security/syslog-ng/