DEIM Forum 2011 C9-2

リアルタイムシステムにおけるログ解析フレームワークの設計

五十嵐　健† 　中田　晋平† 　石綿　陽一†† 　加賀美　聡†† 　菅谷みどり†††

倉光　君郎†††

† 横浜国立大学大学院工学府物理情報工学専攻　〒 240-8501 神奈川県横浜市保土ヶ谷区常盤台 79 － 1

†† (独) 産業技術総合研究所デジタルヒューマン研究センター　〒 135-0064　東京都江東区青海 2-3-26

††† 横浜国立大学大学院工学研究院　〒 240-8501 神奈川県横浜市保土ヶ谷区常盤台 79 － 1

E-mail: †{igarashi,shinpei}@ubicg.ynu.ac.jp, ††{kimio,midori}@ynu.ac.jp

あらまし　近年，システムの複雑化に伴い，障害の原因を特定することが困難になってきている．障害の原因を特定するために，システムが出力するデバッグレベルの情報をログとして保存し，解析を行うログ解析がある．我々は，リアルタイム OS でログ解析を行うことにより，リアルタイムシステム固有の障害を特定できるのではないかと考えた．しかし，ストレージの容量には限界がある点から，大量のログを保存することに問題が生じてしまう．この問題に対しては，オンラインでログ解析を行い対処することが可能である．オンラインでログ解析を行う場合，システムに与える負荷が大きくなってしまい，リアルタイム性が保証されなくなる可能性がある．そこで，本研究では負荷分散型ログ解析アーキテクチャを構築することで，リアルタイム性を阻害することなく障害発生の原因を特定することを提案し，このアーキテクチャを用いて優先度逆転のようなリアルタイムシステム固有の障害発生原因を特定することができるかの実験を行い，評価を行った．

キーワード　リアルタイムシステム，ログ解析，ART-Linux

Design of online log analysis framework in Real-Time System

Ken IGARASHI†, Shinpei NAKATA†, Youichi ISHIWATA††, Satoshi KAGAMI††, Midori SUGAYA†††, and Kimio KURAMITSU†††

† Graduate School of Engineering, Yokohama National University

79-1, Tokiwadai, Hodogaya, Yokohama, Kanagawa, 240-8501 Japan

†† Digital Human Research Center, National Institute of Advanced Industrial Science and Technology

2-3-26, Oume, Koutou, Tokyo, 135-0064 Japan

††† Division, Yokohama National University

79-1, Tokiwadai, Hodogaya, Yokohama, Kanagawa, 240-8501 Japan

E-mail: †{igarashi,shinpei}@ubicg.ynu.ac.jp, ††{kimio,midori}@ynu.ac.jp

Abstract　Recently, the Real-Time Operating System that guarantees real-time on general purpose Operating System is researched. Identifying the cause of the failure that occurred in the system as the system complicated became difficult. One of the techniques for identifying the cause of disability includes the log analysis. It is possible to identify the cause by the amount of output log as large as for log analysis. However, to apply the log analysis to the Real-Time system, there is a limit in the storage to preserve the log and the point where the load that the log analysis gives the system is large become problems. We aims to identify the cause of the failure without obstructing real time by constructing the load-balancing log analysis architecture.

Key words　Real-Time System, Log analysis, ART-Linux

1. はじめに

近年，汎用 OS のリアルタイム拡張を行い，リアルタイム性を保証することのできる研究が数多くおこなわれるようになってきた [1], [2], [8]. 車載システムやロボットシステムなどに用いられるリアルタイムシステムでは，障害が発生した場合，大きな事故につながる可能性がある．しかし，システムが複雑化するにつれて，システムの詳細を理解することや障害の原因を特定することは難しくなる [10] ため，障害の原因を特定することは我々にとって重要な課題であるといえる．

障害が発生した際，障害の原因を特定する方法として，システムが出力するデバッグレベルの情報をログとして保存し，この情報を解析する方法（ログ解析）がある．

例えば，リアルタイム OS における問題の一つである優先度逆転を例に挙げると，タスクの状態遷移や優先度をもとにしたログを解析することにより特定することができる可能性がある．

優先度逆転を特定するためには，デバッグ情報をできる限り出力するシステム上で解析を行うことが考えられる．しかし，運用時において，このデバッグ情報は必要最低限の情報のみ出力するように調整することが考えられるため，運用時にログ解析を行うためには，以下の二つが問題となる．

（1） ログを保存するためのストレージの制限

ストレージの容量には限界があるため，容量の小さいストレージにログを残している場合，解析を行う際に必要となるログが保存されていない可能性がある．特にリアルタイムシステムが利用されるようなシステムでは，ストレージの容量が少ないことが多い．

（2） ログ解析の処理が他の処理に与える負荷

ログ解析の負荷は解析対象となるログの量・解析するアルゴリズムに依存するため，設計したスケジューリングに影響を及ぼす可能性がある．

障害発生の原因を特定するためには，特定したい種類だけログを取得しなければならない．リアルタイムシステムにおける障害原因は，優先度逆転に限らず，スケジューリング設計のミスやメモリリークなどがある．これらの障害原因を特定するためには，取得するログの量が多量になることが考えられるため，ログを保存するストレージの容量について検討しなければならず，複数のログ解析が動作することが考えられるため，他の処理に与える負荷についても考慮しなければならない．

本研究では，ストレージの容量に依存することなくログ解析を行い，ログ解析による負荷を低減し，リアルタイムシステムにおける既知の障害原因を特定することを目的とする．

我々は，汎用リアルタイム OS である ART-Linux [1] 上で負荷分散型ログ解析アーキテクチャを実現し，対象となる処理と解析を行う処理を別々のコアで動作させることにより，ログ解析の処理が他の処理に負荷を与える問題に対処する．また，取得したログをストレージに保存してから解析を行うのではなく，取得したログを順次解析することによりストレージの問題に対処する．更に，ログ解析には，ART-Linux の状態遷移を作成し，このログと LTTng にて取得できるログを利用する．

以下，第 2 章で問題提起を行い，第 3 章でログの取得・解析，負荷分散アーキテクチャの設計，第 4 章でこのアーキテクチャを用いた実験とその評価を述べる．また，第 5 章で関連研究との比較を行い，最後に第 6 章で本研究を総括する．

2. 問　　題

運用時のシステムに対してログ解析を用いるためには，以下二つの問題が挙げられる．

（1） ログを保存しておくストレージの容量

ログの量は任意に調節することが可能であるが，特定しようとする種類の分だけ取得するログの量を増やさなければならない．

本研究では，通常のログに加え，各タスクの状態遷移に関するログを取得することで，優先度逆転だけではなく，スケジューリング設計のミスやメモリリークなど，様々な原因を特定することを想定している．しかし，全てのタスクの状態遷移に関するログを取得する場合，ログの量は膨大なものとなる．この結果，ログを取得し，一度システムを停止させてから解析を行う場合，解析に必要なログがストレージに保存されておらず，正しいログ解析を行うことができなくなってしまう可能性がある．特に，リアルタイムシステムではストレージが小さい場合が多いため，必要なログが保存されていない可能性は高い．

（2） ログ解析にかかるオーバーヘッド

取得するログの量が多いほど，ログ解析にかかる時間は増し，システムにかかる負荷は増加する．

本研究では，リアルタイム OS を対象としたログ解析を試みており，リアルタイムタスクが優先度の低いタスクと通信を行っている場合，オンラインで行っているログ解析がリアルタイムタスクに負荷を与え，リアルタイム性を阻害し，重大な事故を引き起こしてしまう可能性がある．例えば，車載システムにおけるブレーキシステムのリアルタイム性が阻害されてしまった場合，ブレーキを踏んでからすぐに応答することができず，大きな事故を引き起こしてしまうことがある．他にも，歩行しているロボットシステムが転倒してしまったり，モータを期待通りの速度で回転させたりすることができなくなってしまう．このように，リアルタイムシステムでリアルタイム性を阻害されてしまった場合，そのシステムはどのような動作をするか予想することができなくなってしまう．

3. 提案・設計

本章では，ストレージに依存することなくログ解析を行い，このログ解析によってリアルタイム性が阻害されることがないアーキテクチャの設計について述べる．

3.1 ログの順次解析

取得するログの量がストレージの容量を上回ってしまった場合，ログ解析を行う時にログが保存されておらず，障害発生の原因を特定することができない，という問題が発生する可能性がある．この問題は，取得したログを順次解析することにより対処する．この手法では，ストレージにログを保存する前に解析を行うため，ストレージに保存しきれなかったログを解析で

きないという問題を解決することができる．このログ解析機構は，スクリプティング言語 Konoha [7] にて提供されている，Stream Evidence Engine(SEE) を採用した．

3.2 マルチコアを用いた負荷分散アーキテクチャ

ログ解析がシステムに与える負荷を分散させるために，マルチコアを用いた負荷分散アーキテクチャを提案する．このアーキテクチャで，対象となるタスクとログ解析を行うタスクをそれぞれ別の CPU で動作することにより，対象となるタスクの負荷を低減する．このアーキテクチャ図を図 1 にて示す．

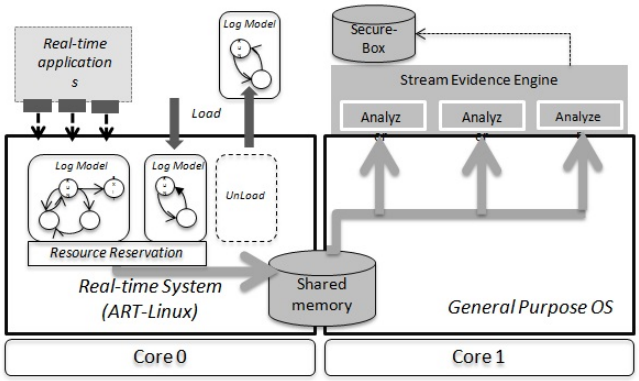

図 1　負荷分散アーキテクチャ

この手法では直接ハードウェア上で複数の OS を動作させるため，仮想化を行うなどのオーバーヘッドは気にしなくてもよい．しかし，ログを出力する CPU とログ解析を行う CPU は別々の CPU であるため，取得したログをログ解析を行う側の CPU に転送しなければならない．今回の障害発生原因を特定するために出力したログの量はおよそ 25[MB/sec] であり，一つのプローブから出力されるログのデータサイズはおよそ 150[Byte] である．このログの量を超えるようなデータ転送機構を用いなければ，全てのログを正しく転送することができない．そこで，このログ出力量よりも高速にデータを転送するために，本研究では共有メモリを用いた OS 間データ転送を設計，実装した．ART-Linux で共有メモリを利用する際には手動で排他制御を行わなければいけないため，ログを書き込む際には読み込み禁止のフラグを追加し，ログを読み込む際には書き込み禁止のフラグを追加する．また，障害発生の原因を解析するための処理は Konoha にて記述する．共有メモリを用いた OS 間データ転送の設計を図 2 で示し，転送の手順をこの図を用いて説明する．

（1） NRT task の仮想アドレス空間に共有メモリをマップ
（2） Log analysis task の仮想アドレス空間に共有メモリをマップ
（3） 共有メモリにログを書き込み（書き込み終了後，読み込み許可にフラグを変更）
（4） 共有メモリからログを読み込み（読み込み終了後，書き込み許可にフラグを変更）

4. 実験・評価

本章では，予めシステムにバグを入れておき，本研究で提案

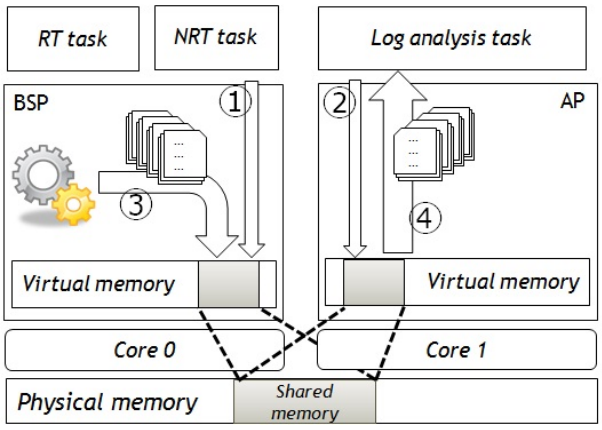

図 2　共有メモリを用いた OS 間データ転送

したアーキテクチャを用いてバグを正しく特定することができるかという実験を行った．以下，バグとその特定方法に関して 4.1 節で述べ，ログの容量削減に関しては 4.2 節，ログ解析タスクを別のコアで行うことによるシステムの負荷分散に関しては 4.3 節で述べる．

4.1 障害解析

本実験では，対象とするリアルタイム OS を ART-Linux とし，混入させるバグを以下の四種類とした．

（1） 誤ったシステムコールの呼び出し
（2） 優先度逆転の発生
（3） スケジューリング設計の誤り
（4） メモリリーク

これらのバグは，OS 間通信でログを転送し，オンラインでログ解析を行うことで特定した．以下でバグの特定方法について記述する．

4.1.1 ART-Linux におけるタスクの状態遷移

我々は，ART-Linux 上で，リアルタイムシステム特有の障害発生原因を特定するために，タスクの状態遷移図を作成し，この情報をログとして取得することにより解析の手助けとした．図 3 にタスクの状態遷移図を示す．

また，LTTng [3] により出力される情報をログとして取得し，カーネルソースコード内にプローブを挿しこむことにより，ART-Linux 固有のログも取得した．ART-Linux 固有のログは LTTng のフォーマットに合わせ，ログ情報を統一的に扱えるようにした．

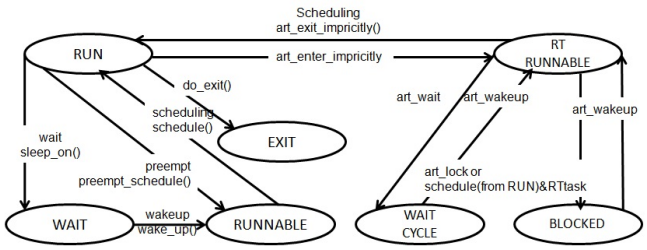

図 3　ART-Linux における状態遷移

4.1.2 システムコール

ART-Linux でリアルタイム処理を行うためには，ART-Linux に用意されている API を用いて特別なシステムコールを発行しなければいけない．以下のソースコードにその一例を示す．

```
void artsv_servo(void)
{
  int i;

  /* art_enter is ART API */
  if(art_enter(ARTSV_PRIORITY_SERVO,
               ART_TASK_PERIODIC,
               ARTSV_PERIOD_SERVO) == -1){
        perror("art_enter error");
        exit(1);
      }

      /* art_wait is ART API */
      while(artsv_enabled_flag){
      art_wait();
      i++;
      }
}
```

art_enter() はノンリアルタイムタスクをリアルタイムタスクに変換することのできる関数であり，*art_wait*() は次の周期時刻まで休止状態に遷移する関数である．上記の API を用いなければ，システムの実行周期をマイクロ秒単位で指定する事ができず，リアルタイムタスクとして動作させることができない．

このバグは，システムコールが正しい順番で呼ばれているかを解析しなければいけない．このため，作成した ART-Linux の状態遷移図に従い，リアルタイム処理を行うタスクが ART-Linux の用意したシステムコールを正しく呼び出しているかを解析することでバグを特定した．

4.1.3 優先度逆転

一般的に，リアルタイムシステムでは，優先度に沿ってスケジューリングが行われる．しかし，このスケジューリング方式では優先度の低い処理が優先度の高い処理が必要としている資源を占有している時に，優先度逆転が発生する可能性がある．優先度逆転とは，優先度の低い処理がその資源を解放するまで優先度の高い処理を実行することができず，優先度が逆転して処理が実行されてしまうことである．この時，優先度が中程度の別の処理が動作を始めた場合，優先度の低い処理はこの処理に実行権を渡してしまうため，優先度の高い処理が実行できないまま優先度が中程度の処理を実行してしまうことになり，リアルタイム性を満たすことができなくなる．

図 4 は以下の順序で動作したため優先度逆転が発生した例である．

- 最も優先度の高いタスク

共有資源のロックを取得 …(1)
共有資源のデータを読み込み …(2)

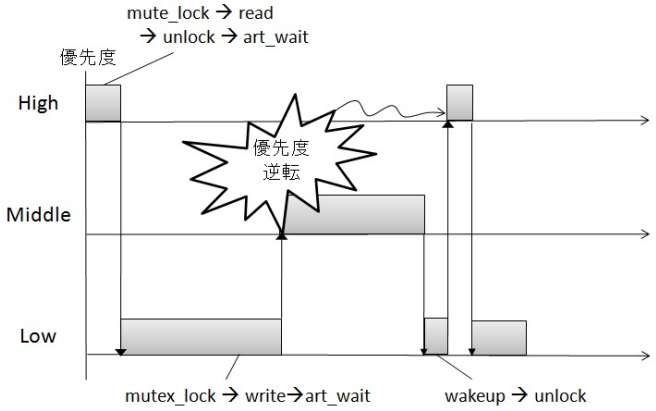

図 4　優先度逆転

共有資源のロックを解放 …(3)
休止状態に遷移 …(4)

- 最も優先度の低いタスク

共有資源のロックを取得 …(5)
共有資源にデータ書き込み …(6)
中優先度の割り込みにより休止状態に遷移 …(7)

- 最も優先度の高いタスク

上記と同様の処理を試みるが，共有資源のロックが取得できないため休止状態を継続 …(8)

- 最も優先度の低いタスク

中優先度のタスクが処理を終了した後，処理を再開し共有資源のロックを解放 …(9)

本来，最も優先度の高いタスクは 8 の段階で処理を開始し，共有資源のデータを読み始めなければならないが，優先度逆転が発生してしまい，障害が発生している．このバグは，状態遷移図を用いて，優先度の高い処理が共有資源を利用する際，優先度の低い処理が正しく休止状態に遷移しているかを解析することで特定することができた．

4.1.4 スケジューリング設計ミス

ART-Linux のリアルタイムスケジューリング設計は，プリエンプティブで固定優先度を使用するレートモノトニックスケジューリングを採用している．レートモノトニックスケジューリングが全デッドラインを満たすことを保証することができる CPU 利用率の限界は 69.3 %程であり，残りの 30.7 %はノンリアルタイム処理に使用可能であるが，この数値は処理の組み合わせがどのようなものでもスケジューリング可能となる数値であり，ランダムに生成した周期処理の場合，CPU 使用率が 85 %以下ならば全デッドラインを満たす [4]．しかし，この数値を上回ってしまうと，以下の図のようにデッドラインを超過し，リアルタイム性が満たされなくなってしまう可能性がある．

図 5 では，以下のようなスケジューリング設計を行ったため，デッドラインを超過してしまった．

- 高優先度のタスク : 処理時間が 1[ms]，周期時間が 10[ms]
- 中優先度のタスク : 処理時間が 8[ms]，周期時間が 10[ms]
- 低優先度のタスク : 処理時間が 4[ms]，周期時間が 20[ms]

上記の設計で，全てのタスクが同時に動作を開始した場合，ま

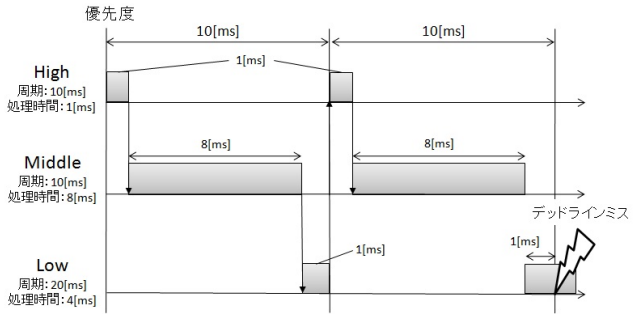

図 5　スケジューリング設計ミス

ず最も優先度の高いタスクが 1[ms] 動作する．次に，中優先度のタスクが 8[ms] 動作し，実行権を低優先度のタスクに渡す．低優先度のタスクは 4[ms] の動作を試みる．しかし，1[ms] 動作したところで高優先度のタスクの周期時間に到達してしまうため，実行権を高優先度のタスクに渡す．先ほどと同様に高優先度のタスクが 1[ms]，中優先度のタスクが 8[ms] 動作した後低優先度のタスクが動作を再開する．しかし，低優先度のタスクが 1[ms] 動作した所で，周期時間である 20[ms] に到達してしまい，デッドラインミスが発生してしまう．

このミスを特定するためには，予め決められた各リアルタイムタスクの周期，および一周期中に実行される時間を全て把握しておかなければならない．しかし，すべてのタスクに関する周期を常に把握しておくことは困難である．そこで，出力するログに，各リアルタイムタスクに設定した周期時間を付加し，この値を解析することでバグを特定した．

4.1.5　メモリリーク

メモリを解放することなく大量のメモリを確保してしまい，メモリが枯渇し，複数のタスクがメモリを確保しようとした時に遅延が発生し，場合によってはメモリ割り当てが行われないこともある．

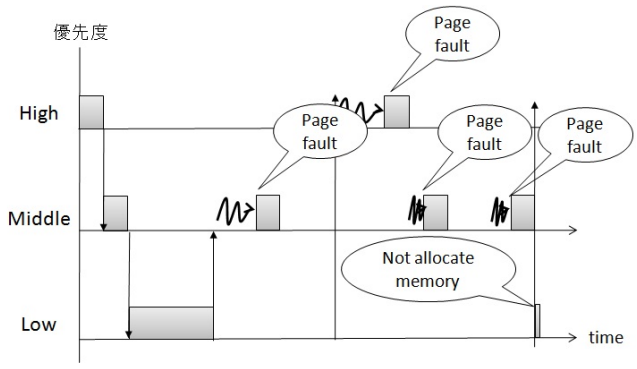

図 6　メモリリーク

上図のように，それぞれのタスクがメモリを確保したまま動作を続けると，メモリの空き容量が少なくなる．その結果スワップ領域への入出力を繰り返すようになり，遅延が発生したり，メモリの割り当てを行うことができなくなりシステム自体がダウンしてしまう．このバグは，LTTng にて取得することのできるメモリの空き容量やスワップの入出力を解析することにより特定した．

4.2　ログの容量削減

本研究では，出力された情報を保存せずにオンラインでログ解析を行うために，Stream Evidence Engine を採用した．これにより，出力された情報は順次解析を行う事が出来るため，ログを保存するストレージの問題を解決することができた．しかし，リアルタイム処理を行う CPU から出力されるログは，ログ解析を行うために用意した別の CPU に転送しなければいけない．この時，転送速度が遅すぎた場合，正しくログ解析を行う事ができない可能性がある．

そこで本節では，ソケット（TCP/IP）を利用した OS 間データ転送と共有メモリをを利用した OS 間データ転送方式での転送速度に関する実験を行い，どちらの転送方式がログ解析に適しているかを評価した．以下にその実験結果を記載する．

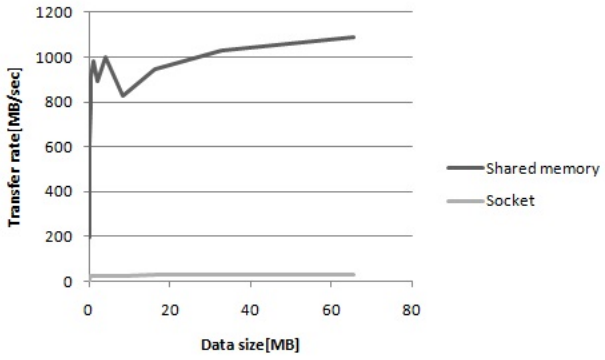

図 7　ログ転送速度

図 7 から，ソケットを用いた OS 間データ転送では約 25[MB/sec] の転送量であるのに対し，共有メモリを用いたデータ転送では毎秒約 1[GB] のデータを転送することができた．ソケットを用いたデータ転送は，データの読み書きを行う際に，ユーザ空間からカーネル空間，またはカーネル空間からユーザ空間へデータのコピーが発生することが転送の遅延につながったと考えられる．今回の実験で取得したログは，毎秒 25[MB] 以上のデータが出力されることもあったことから，共有メモリを用いたデータ転送でなければ転送することができなかったといえる．

4.3　ログ解析の負荷低減

本節では，対象となるタスクとログ解析タスクを一つのコアで動作させた場合と，別のコアでそれぞれを動作させた場合の結果を述べる．

一つのコアで対象となるタスクとログ解析タスクを動作させた場合，その CPU 利用率はほぼ 100 %となってしまい，システムに大きな負荷を与える結果となった．一方，別のコアでそれぞれを動作させた場合，対象となるタスクが動作するコアではログ解析タスクによる負荷を受けることなく動作していることがわかる．この結果から，本研究で提案したアーキテクチャでログ解析の負荷を低減することができたといえる．

5.　関 連 研 究

現在，リアルタイムシステムへの負荷を分散させるサービス

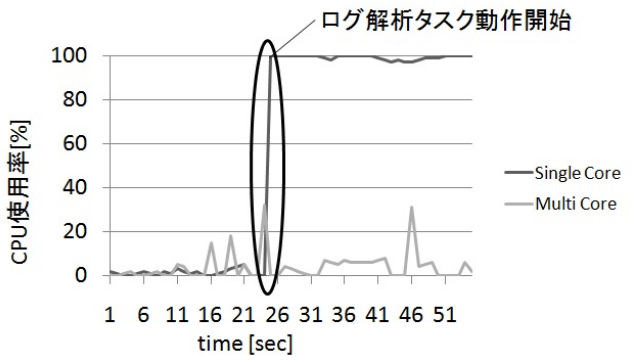

図 8　システムに対するログ解析負荷

として Real-Time Hypervisor（RTH）[9] が挙げられる．RTH は汎用 OS とリアルタイム OS をそれぞれ別の CPU にて並列動作させる仮想ソフトウェアである．それぞれ別の CPU にて動作させているため，リアルタイム性を損ねることはない．我々の提案した負荷分散アーキテクチャとの違いとして，ハイパーバイザーを利用している点が挙げられ，我々の提案するアーキテクチャでは，直接ハードウェア上で複数の OS が動作するため，仮想化によるオーバヘッドが生じない．

6. 結　　論

リアルタイム OS ART-Linux を用いて，ログ解析により生じる負荷を分散するアーキテクチャを設計し，リアルタイムタスクがログ解析により受ける負荷を低減した．これにより，リアルタイムタスクのリアルタイム性を阻害することなくログ解析を行えることを示し，取得したログを順次解析することにより，ログを保存するストレージに関する問題に対処することができることを示した．また，実際にリアルタイムシステムに障害を発生させ，取得したログを解析することにより，障害発生の原因を特定することができた．

文　　献

[1] Youichi Ishiwata, Satoshi Kagami, Koichi Nishiwaki and Toshihiro Matsui, ART-Linux 2.6 for Single CPU: Design and Implementation
[2] Victor Yodaiken and Michael Barabanov. A Real-Time Linux, Proceedings of the Linux Applications Development andd Deployment Conference (USELINUX), Anaheim, CA, Januzry 1997. The USENIX Association.
[3] M.Desnoyers and M.R. Dagenais. The lttng tracer: a low impact performance and behavior monitor for gnu/linux. In Proceedings of the Linux Symposium, volume1, pages 209-224, 2006
[4] J.Lehoczky, L. Sha and Y.Ding, The Rate monotonic scheduling algoritum: exact characterization and average case behavior, IEEE Real-Time Systems Symposium, pp.166-171, December 1989
[5] Andreas Haeberlen, Petr Kouznetsov, Peter Druschel, PeeReview: Practical Accountability for Distributed Systems, Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles, October 14-17, 2007, Stevenson, Washington, USA
[6] M.Lee, A.S.Krishnakumar, P.Krishnan, N.Singh, and S.Yajnik, Supporting Soft Real-Time Tasks in the Xen Hypervisor, in VEE '10: Proceedings of the 6th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments. New York, USA: ACM, 2010, pp. 97-108.
[7] Konoha Scripting Language, http://konoha.sourceforge.jp/
[8] T.Inc, Timesys linux, http://www.timesys.com
[9] Real-Time Hypervisor, http://www.linx.jp/product/rth
[10] Nancy G. Leveson, Safeware: System Safety and Computers, ACM, New York, 1995