Vol.2011-SLDM-149 No.22
Vol.2011-EMB-20 No.22
2011/3/18

組込みシステムトレース技術とその応用

長野岳彦† 亀山達也†

組込みシステムの高機能化・高信頼化が進んでいるが，メーカ間のシェア競争は激化し，開発コストの削減が望まれている．組込みシステムの障害には様々あるが，工程遅延に大きく影響を与える障害として，再現性の低いソフトウェア障害がある．再現性の低いソフトウェア障害が工程遅延に影響する原因の一つは，組込みシステム自体の記録領域が小さいため，再現条件特定のために，何度も試行錯誤しながら情報を取得する点にある．本報告では，この試行錯誤的な対策を回避するため，外部記録媒体に記録可能な，長時間トレースシステムを提案し，評価した．また，既存のトレース技術の課題であった，製品毎への移植の必要性と，今後増えるインターネット接続による，出荷後のソフトウェア構成変更に対応するための，動的なトレース内容の変更を実現する，製品出荷後トレース技術を提案し，評価した．更に，長時間トレース実現の副作用である，対象のログ増大による解析効率悪化という新たな課題に対し，データマイニングを用いて，解析対象のログを削減する手法を提案し，解析効率向上の見込みを得た．

Embedded System Trace Technology and the application

Takehiko Nagano† and Tatsuya Kameyama†

Performance and reliability of embedded systems has advanced. However, it is desired to reduce costs. Because the competition for getting market share among manufacturers is intensifying. Although many variety of embedded system failure is exist, One of the main reasons for the delay is not reproducible software failure. Recording Resources of embedded systems is small, so we need trial and error to obtain reproduction information. In this paper, we propose a long time tracing system to solve above problem. And we examined the proposed method.
In addition, We propose the running products tracing method to solve the problems of existing tracing method. Above problem consists of two problems, One is ported by product, Second is dynamic changing tracing contents for software configuration changes using internet. And we examined the proposed method.
Finally, we got a new problem of the efficiency degradation due to increased log data using long time tracing system. To solve the problem, we propose a log reducing method using data-mining. And we obtained good result to improve efficiency analysis.

1. はじめに

組込みシステムの高機能化・高性能化が進んでいるが，メーカ間のシェア競争の激化から，低コスト化や，開発期間の短縮に対する要求が強くなっている[1]．そこで，開発コスト削減を目的として，ソフトウェアのバグに代表される障害の対策期間削減による開発効率の向上が望まれている．

また，高機能化・高性能化に伴い，インターネットに接続可能なシステムが増加している．今後は，OSGi*フレームワーク等を利用した，インターネットを利用した組込みシステムのソフトウェアの構成変更が可能となる[2]．その結果，製品出荷後に導入されたソフトウェアや設定等，製品出荷前には想定出来ない，外部要因に起因した障害の発生が懸念される[3]．この様な出荷後の構成変更に起因する障害は，事前にテストで検出し，対策することが困難と考えられ，出荷後における対策が必要と考えられる．

この様に，製品の開発時から出荷後に掛けて，組込みシステムの品質・信頼性の確保が大きな課題になっている．特に組込みシステムの場合，アプリケーションソフトウェアだけで無く，プラットフォームを構成するハードウェア，ソフトウェアの構成が異なり，ハードウェアを直接制御するプログラムも多い．そこで，アプリケーションソフトウェアやデバイスドライバ等，システムを俯瞰出来，問題点を切り分け可能な，システムワイドな解析をする必要がある．OS はハードウェアの抽象化，システム全体の制御をしており，その挙動情報がシステムワイドな解析に役立つ．

OS の挙動情報を用いて時系列解析をするツールは，デジタルテレビに代表される情報系組込みシステムに採用されており，実際に成果を上げている．組込みシステム向けの Linux では，LKST[4]や LTTng[5]等のトレーサが公開されており，製品開発への適用が進み，一定の成果を得ている．しかし，以下に示す課題を持っている．

(1) 組込みシステムはメモリ等のリソースが少ないため，OS の挙動情報等を大量に保存出来ない

† 株式会社 日立製作所 中央研究所 組込みシステム研究部
Hitachi, Ltd., Central Research Laboratory Embedded Systems Research Department

* OSGi は OSGi の略称であり，また米国 OSGi Alliance の登録商標

Vol.2011-SLDM-149 No.22
Vol.2011-EMB-20 No.22
2011/3/18

(2) カーネルの標準機能として取り込まれていないため，製品に搭載されたカーネル向けに移植が必要
(3) Linux カーネルに静的に組み込まれているため，デバイスドライバの変更等に代表される，製品出荷後の構成変更等のトレースに対応していない．

そこで本報告では，上記課題 3 点を解決する対策として以下 2 点を検討し，実装・評価した．

(A)外部接続機器を利用した長時間トレース技術
(B)Linux の最新トレース技術の動向調査を考慮した組込みシステム製品出荷後トレース技術の提案と評価．

更に，上記(A)を実現した結果，解析対象の情報増加に伴う解析効率低下という新たな課題が発生した．本報告では，その対策に関する一提案として，長時間トレース結果の解析効率向上技術案を検討し，実装評価した．

2. 提案手法

本章では，長時間トレースシステムと，製品出荷後トレース技術について述べる．

2.1 長時間トレースシステム

組込みシステムの開発において，開発期間の短縮が求められており，ソフトウェアのバグに代表される障害の対策期間削減による開発効率の向上が望まれている．

先に述べた通り，組込みシステムはメモリ等のリソースが少ないため，OS の挙動情報等を大量に保存することが出来ない．その課題が，組込みシステムの障害解析効率を悪化させる原因となっている．

組込みソフトウェアの障害には様々な例があるが，再現性が低く，解決が困難な障害がある[6]．この様な再現性の低い障害に対し，従来の OS トレーサは記録容量が少ないため，再現条件特定をするために，何度も情報を取得し，思考錯誤をしながら障害解析繰り返す必要がある．結果，障害解析工数の削減を目的として使用している OS トレーサの制約が，障害解析工数増加の要因となってしまっている．

そこで，本報告では，再現性の低い障害の解析効率の向上を目的とし，障害解析に必要十分な情報を一度に取得する長時間トレースの手法を提案する．

組込みシステムはメモリリソースが少ないケースや, HDD に代表される記録用のデバイスを保有してないケースが多い．そこで，提案手法では，システムが外部にデータを出力する外部出力バスを持っていることを前提とし，外部バス経由で記憶容量が大きい外部記録装置に OS の挙動情報を記録する方式を提案する（図 1）．

今回, OS の挙動情報の収集には LKST（Linux Kernel State Tracer）を使用した．LKST は Linux カーネルの状態変化トレースする機能で，日立，IBM，富士通，NEC の 4 社協業で作成された Linux の拡張機能である[7]．従来の LKST において，長時間のトレースをするには，メモリ上に複数面のバッファを確保し，そのうちの 1 面のバッファ領域に対し，イベントが発生する毎に，内容を記録し，そのバッファを使い切った後

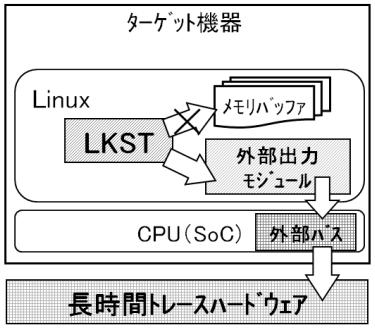

図 1　長時間トレースシステム

バッファを切り替え, 元のバッファの記録結果を外部媒体に, ファイルとして出力し，長時間トレースを実現している．しかし，組込みシステムに搭載されるメモリは容量に制限がある．また，出力先のストレージデバイスも無い場合が多い．

そこで本システムでは，外部バスに LKST のトレース結果を送出するための出力モジュールを追加し，外部記録装置のインタフェースの有無を問わず，挙動情報の出力を可能にした．この出力モジュールには，LKST のイベントハンドラを用い，トレース内容をカーネルに記録する処理を Hook して，メモリ領域に記録する情報をそのタイミングで 16bit のデータ列に分解し，外部バスに出力する．外部バスには，HDD を保有する長時間トレースハードウェア（今回は横河デジタルコンピュータの TRQer†を接続し，バスにイベントの出力が流れる度，長時間トレースハードウェアに記録する仕組みになっている．このシステムにより，トレースの記憶容量は組込みシステムが保有する数 MB～数十 MB 程度から 200GB に増加する．

2.2 製品出荷後トレース技術

本節では，今後の組込みシステムの高機能化に伴う，インターネット経由の構成変更を想定した製品出荷後トレース技術について，以下の流れで述べる．まず既存の組込みシステムのトレース技術の動向を述べ，その後，最新の動向を考慮した，システムの構成変更に柔軟に対応可能な，新たなトレース技術の提案をする．

† TRQer は横河デジタルコンピュータ株式会社の登録商標

Vol.2011-SLDM-149 No.22
Vol.2011-EMB-20 No.22
2011/3/18

2.2.1 既存トレース技術の動向

OS のトレース技術は,古くから有るが，Linux をターゲットとしたトレーサは 2000 年以降，LKST[4]や LTTng[5]等が立て続けにリリースされた．その後，LKST や LTTng では対応出来ない，動的なトレース内容の変更を実現する Kprobes や，Kprobes を安全に使用するための SystemTap 等，様々なトレーサが公開され，システム開発に使われてきた．

しかしそれらのトレーサは，Linux の標準機能にはなっておらず，Linux カーネルが更新される度に，トレーサの修正を，トレーサの使用者が実施する必要があった. Linux カーネルの更新頻度，規模は大きく，例えば Kernel2.6.18 から 2.6.24 までのマイナーバージョンアップにおける，平均増加行数は 158,119 行である．また，マイナーバージョンアップ時に，重要な機能の名称変更や，ファイルの統合等も行われる．結果，バージョンアップの度に, 膨大なコードの解析や, メーリングリストの追跡を実施し，トレースポイントの特定や，検証をしなくてはならず，開発効率向上を促進するツールにも拘らず，その導入が開発者の負担となっていた．

しかし，LinuxKernel 2.6.28 から正式に ftrace というトレーサと，tracepoint というフレームが組み込まれ，状況が改善した．ftrace は gcc のオプション(-pg)を利用し，カーネル内部の関数の入出力を全トレースする仕組みのイベントトレーサである．このトレーサの特長は，トレース実施時にのみ，トレース関数へ分岐し，普段は NOP 命令(No Operation Performed)列としておくことで，処理負荷を軽減する点と，Generic ring buffer という汎用的なデータ格納処理を実装している点である．tracepoint は，対象のイベントを Hook するインタフェースと，Hook に関連した処理の on/off を動的に実現するフレームワークであり，動的に取得イベントを切り替えるための基盤として活用が進んでいる．

これらの新しいトレース基盤が整うことで，様々なトレーサが Linux に取り込まれることになり，Linux2.6.35.7 の時点では，10 以上のトレーサがカーネルに取り込まれている. 特に ftrace event tracer と呼ばれる静的な汎用トレーサは, 従来 LKST や LTTng で実施していた，静的な OS イベントの収集（の一部）を実現しており，様々なユーザへの普及が見込まれている．

2.2.2 製品出荷後トレース技術

上記に述べた様々なトレーサを使い，製品出荷後の組込みシステムからの情報取得について検討する．

製品出荷後の情報収集では，システムの機能要件/非機能要件を損なうことなく，問題の解決に必要な情報を収集することが必要になる．また，ネットワーク経由でソフトウェア構成が変更された場合，新たなトレース内容を追加する，解析内容を深堀するため，トレース内容を変更する等，動的な構成変更が必要になる．以上より，以下 2 点を製品出荷後トレース技術実現上の前提とした．

（1） 機能要件/非機能要件に影響を与えない低負荷なトレーサ
（2） 動的に取得イベントを変更出来るトレーサ

その上で，2.2.1 既存トレース技術の動向で提起した，導入工数の問題をクリアする必要がある．よって，既にカーネルに組み込まれている機能を中心に構成する．

図２に, 新たなトレーサの案を示す. このトレーサは, ftrace ring buffer と, tracepoint フレームワーク，LKST Event trace module から成る．

まず，tracepoint のフレームワークを利用することで，Kernel Module としてイベント毎のトレーサを動的に追加/削除/記録/停止を実現可能にした. 次に, ftrace ring buffer を利用することで，トレーサの格納処理の作り込みや修正を無くした．最後に LKST Event trace module であるが，これは既に有用性が示されている LKST のトレース内容[4][7]を，如何に上記のフレームワークで実現するかという点を考慮して作成した．LKST Event trace module は，LKST の各イベント(例 : context switch,IRQ 等)毎に作成する．LKST Event trace module 内部は，trace point フレームワークに則り，Hookpoint に登録する関数を設定し，通過時の処理内容を ftrace ring buffer に記録する．記録する内容は，LKST に準拠する．また，Hookpoint は LKST と ftrace event tracer で共通化し，出来る所は，可能な限り ftrace 側に準拠することで，開発工数を削減する．

3. 長時間トレース技術の評価

第 2 章で提案した長時間トレースシステムの実用性を評価するため，Apache Bench を用いて，外部から評価対象に対し負荷をかけ，性能を測定した．評価項目は CPU の使用率と Apache Bench によるテスト実行時間を用いた．なお，長時間トレースの評価は，製品開発時を想定し，ピーク負荷時にどの様な変化があるかを考慮し，評価を実施した．

3.1 測定環境

組込みシステム（Atmark Techo 社 Armadillo‡-300）と負荷を外部から与える PC（OS:Linux）をローカルネットワークに接続し，組込みシステム上で，web サーバ（thttp）を動作させ，PC から Apache§ Bench を使用し，組込みシステム上の web サーバに大量のアクセスを発行, 負荷を与える. 使用したコマンドラインは“ab –n 30000 –c5 アクセス先 URL”である. CPU の使用率は, Snap Gear 社の Greg Ungerer 氏が OSS で公開している cpu.c を用いた．cpu.c は/proc/stat 以下にある CPU の動作状況から，システム時間，ユーザ時間等を算出するツールである．

‡ Armadillo は株式会社アットマークテクノの登録商標
§ Apache は Apache Software Foundation の登録商標または商標

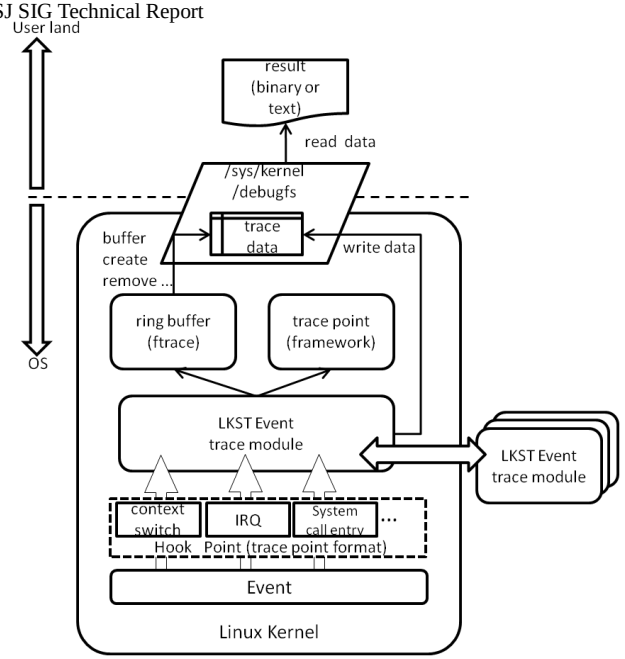

図 2.提案するトレース手法の概要

上記を使い，LKST 無し，LKST のトレース結果をメモリ上に記録，同トレース結果をファイルに記録（従来の長時間記録手法），提案手法である長時間トレースシステムに記録の 4 例で比較した．このうち，トレース結果を定期的にファイル出力する方式は，Linux サーバ等で長時間記録する際に使用される手法である．LKST のマスクセット[2]は全て ON になっている

3.2 評価結果

結果を表 1 に示す．Apache Bench による負荷が高いため全体的に CPU 使用率は高いが，従来手法である LKST のトレース結果をファイルに記録する手法の CPU 負荷が最も高く，3.02%上昇している．それに対し，長時間トレースを用いる場合 0.23%の上昇と，それ程 CPU 負荷に影響が出てない．しかし，CPU 使用率のうち，OS が CPU を使用した結果であるシステムの項目を見ると，長時間トレースが，LKST 無しの場合に比べ 20.14%上昇している．理由は外部バスにトレース結果を出力するカーネルモジュールが頻繁に動作しているためである．従来手法のシステムの項目が低い理由は，メモリからファイルに結果を出力する際，ユーザランドのデーモンプログラムが頻繁に動作しているためである．

次に，Apache Bench の測定結果を表 2 に示す．LKST が動作しない場合に対し，ファイル記録は 156.62 秒の遅延，長時間トレースは 128.11 秒の遅延が発生している．双方とも，LKST 無しよりは性能が悪化しているが，ファイル記録（従来法）に比べ，提案手法は 17.88%性能が改善している．

表 1　CPU 負荷測定結果

	LKST 無	LKST ﾒﾓﾘ記録	LKST ﾌｧｲﾙ記録	LKST 長時間ﾄﾚｰｽ
CPU 使用率	92.00%	92.45%	95.02%	92.23%
内 system	36.80%	42.32%	34.47%	56.94%

表 2　Apache Bench 測定結果

	LKST 無	LKST ﾒﾓﾘ記録	LKST ﾌｧｲﾙ記録	長時間ﾄﾚｰｽ
所要時間(秒)	159.49	173.16(+8.57%)	316.11(+98.20%)	287.60(+80.32%)

4. 製品出荷後トレース技術の評価

第 2 章で提案した製品出荷後トレース技術の実用性を評価するため，3 章同様ターゲット機器に Apach Bench を用いて負荷を掛け，性能を測定した．評価項目は CPU の使用率と，UnixBench の finalscore を用いた．製品出荷後トレースは，システム運用時を考慮し，CPU の負荷が 30%(比較的処理負荷が低い場合)及び 50%(比較的処理負荷が高い場合)を中心に評価を実施した．

4.1 測定環境

CPU 使用率は，Snap Gear 社の Greg Ungerer 氏が作成した OSS ツール cpu.c を使い測定した．ベンチマークツールは，Unix 系 OS の性能を多角的に検証可能な Unixbench[8]を使用した．

比較対象は，トレーサを組込んでいない linux-2.6.23(vanilla)と，既存トレース技術を代表して LKST との比較を実施した．測定したハードウェアの環境は，表 3 の通りである．情報系組込みシステムを想定し，大型なものを想定した 1 と，比較的小型な機器を想定した 2 を用意した．また，上記環境上で，実システム運用を想定し，http サーバ(thttp)を動作させ，ネットワーク外部から，Apache Bench[9]を使い，対象機器に対し，CPU 負荷を 0%,30%,50%となる様調整をしながら，測定を実施した．

Vol.2011-SLDM-149 No.22
Vol.2011-EMB-20 No.22
2011/3/18

表 3. ベンチマーク測定環境（ハードウェア）

1　PC	2 組込みシステム
PC CPU:Intel Celeron** T3000　(1core)1.8GHz Memory:2GB	OpenBlockS†† 266 CPU:PowerPC‡‡405GPr　266MHz Memory: 128MB

4.2 評価結果

まず，CPU 負荷の測定結果を表 4 に示す．トレーサの負荷は，OS の処理として扱われるので，system の値が，各負荷でどの程度増加したかを測定した．以下に見るとおり，PC では 1%以下，OpenBlockS では 5%程度と，実用上問題無いと考えられる．しかし，既存手法（LKST）よりは，負荷が高いという結果を得た．

表 4　CPU 使用率(system)増加分(対 linux-2.6.23 vanilla 単位は%)

1.PC				2.Open BlockS			
負荷 30% 時		負荷 50%時		負荷 30%時		負荷 50%時	
LKST	提案	LKST	提案	LKST	提案	LKST	提案
+0.56	+0.23	+0.70	+0.06	+0.87	+5.07	+0.40	+5.18

次に，OS 自体の負荷が高い状態における影響を，Unix Bench の Final Score(複数指標を統合した結果)で評価した．結果を表 5 に示す．この表は，CPU 負荷 0%の場合に比べ，スコアが劣化した比率を示している．各 CPU 負荷において，提案手法は LKST と比べ,Open BlockS では平均 10.63%,PC では 13.13%劣化しており，OS の負荷が高騰した場合は，LKST より副作用が大きいという結果を得た．

表 5　Unixbench(finalscore)の劣化率(単位は%)

	1.PC			2.Open BlockS		
負荷	0%	30%	50%	0%	30%	50%
LKST	20.20	41.64	42.43	21.86	32.04	56.74
提案	37.11	51.32	55.23	35.33	43.56	63.62

** Intel , Celeron は米国およびその他の国における Intel Corporation の登録商標または商標
†† OpenBlockS はぷらっとホーム社の登録商標
‡‡ PowerPC は米国およびその他の国における米国 International Business Machines Corp.の登録商標

5. 長時間トレース結果の解析効率向上手法の提案

２章で提案した長時間トレースの実現により，解析対象の情報取得量が KB～MB オーダから，一気に 200GB に記録容量が拡大した．結果，解析対象の情報が大幅に増え，解析の効率が低下した．その問題を解決するため，本報告ではデータマイニングを利用し，解析対象のデータ量を削減する手法を提案する．

本報告の目的は，再現性の低いソフトウェア障害の解析効率を向上することである．しかし，再現性の低いソフトウェア障害は，開発者が定めたテスト試行回数内に発生しない場合，工程との兼ね合いから無視されることも多くあり，その事例の収集が困難である．従って今回は，再現性の低いソフトウェア障害の事例を収集するのではなく，ソフトウェアの正常な動作を，データマイニングを用いてパタン学習し，その学習結果を元に，ソフトウェア動作の判定を行う， Anomaly Detection[10][11]のアプローチを用いる．

具体的には取得した長時間トレースデータを元に，解析対象のソフトウェアの正常動作部分を切り出し，クラスタリングを用いて正常状態を学習，解析時には，学習結果から，単位時間単位で正常クラスタのセントロイド Ci と入力データ X の最小値 minD(X,Ci)を求め，その値を異常度として算出し，異常度の大きい箇所周辺を解析対象として絞り込み，解析効率を向上する（図 3）．

今回，LKST のトレースデータをクラスタリングに適用するため，単位時間毎に LKST のイベント種別毎に発生回数を数え上げ，テキストデータをベクトルデータに変更した．

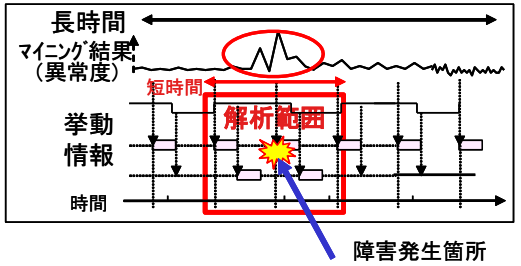

図 3 データマイニングを用いた解析イメージ

6. 障害発生箇所の特定と解析範囲絞込み評価

本章では，データマイニングを用いた問題点の絞込み結果の有効性を，異常度の変化と変化時刻から検証する．

本報告では，再現性の低いバグの解析を対象としている．そこで，(1)複数プログラムが並列動作することに起因するバグ，(2)ハードウェア等の外部要因に起因するバグ，を想定し，以下の 2 種類の障害発生時刻の特定を評価した．

Vol.2011-SLDM-149 No.22
Vol.2011-EMB-20 No.22
2011/3/18

6.1 デッドロックの検出（並列動作起因）

本節では, 複数プロセスが関連し発生する障害の発生が特定可能か評価を実施した. Armadillo-300 上で同一の資源を利用するプロセスを 2 種類起動し，500 秒経過後に片側のプロセスが資源を解放せず，意図的にデッドロック問題を発生させた．表 6 に結果を示す.

表 6　デッドロック発生時の異常度変化

	通常時	デッドロック時
異常度	1	4.00

表 6 の結果からわかる通り, デッドロック発生時には異常度が 4 倍上昇した. また, 異常発生時刻と異常度の変化は同期しており，異常発生箇所を特定出来た.

6.2 ネットワーク障害の切り分け（外部要因）

本節では，開発対象のソフトウェアに対し，外部要因に起因して発生する障害の発生箇所を特定可能か，評価を実施した.

評価手法は負荷測定時と同様の機器構成で, Armadillo-300 上で動作する web サーバ（thttp）に対し, apache-bench を使用して 30 秒毎に外部から大量アクセスを発生させ, その際の平均の異常度の変化を測定した．表 7 に結果を示す.

表 7　ネットワーク攻撃時の異常度変化

	通常時	攻撃時
異常度	1	598.32

表 7 の結果からわかるとおり, 外部から攻撃を受けている際は, 異常度が平均で 598 倍上昇し，異常な箇所が推定出来た．また，4.1 同様，異常発生時刻と異常度の変化は同期しており，異常発生箇所を特定出来た.

7. まとめと課題

組込み機器は，メーカ間のシェア競争の激化から，低コスト化，開発期間の短縮要求が強くなっており，開発コスト削減を目的として，ソフトウェア障害の対策期間削減による開発効率の向上が望まれている．組込みソフトウェアの障害には様々な物があるが，再現性が低く，解決が困難な障害がある[6]．この様な再現性が低い障害の解析効率悪化の原因は，組込み機器自体のデータ記録領域の小ささに起因し，再現条件特定をするために，何度も情報を取得し，試行錯誤をしながら障害解析繰り返す点にある.

本報告では，上記試行錯誤的な対策工数の改善を目的として，長時間トレース技術を提案，評価した.

上記提案技術により，本体に十分なメモリやストレージが無い組込みシステムでも, 記録時間が飛躍的に伸びた．本報告の場合 32MB⇒200GB まで記録容量が増加した. また従来のサーバ向けの長時間記録方式と比較し, Apache Bench の実行結果で 17.88%の性能改善が見られた.

次に，既存のトレース技術の課題であった，製品毎やカーネル毎への移植の必要性，及びに今後増えるだろうインターネット接続による出荷後のソフトウェア構成変更に対応する動的なトレース内容の変更に対応するため，製品出荷後トレース技術を提案し，評価した．結果，システムの CPU 負荷が 50%程度あった場合において，全イベントをトレースした場合でも，負荷の増加は 5%程度に抑えることが出来，実用化の目途がついた. 但し, LKST と比較すると CPU 負荷で 4.78%増加, UnixBench の finalscore では，スコアが 10.63%劣化しており，今後はメンテナンスに影響の無い範囲での改善が必要である.

最後に，長時間トレースにより，再現性の低い障害に対する試行錯誤的な障害再現条件の特定に対する工数の削減は成功したが，対象のログ増大による効率悪化という新たな課題が発生した．上記に対し，本報告では，データマイニングを用いて，解析対象のログを削減する目途をつけることが出来た．今後の課題としては，より実用的な障害に適用し，効率の改善度を評価する必要がある.

文　献

[1] 高田広章:組込みシステム開発技術の現状と展望,情報処理学会論文誌,Vol.42,No.4(2001)

[2] Benefits of Using OSGi:http://www.osgi.org/Main/HomePage

[3] 長野岳彦,亀山達也:組込みシステムのソフトウェア障害予兆検出に適した挙動情報収集手法の検討,FIT2010

[4] 畑崎恵介,中村哲人,芹沢一,“稼動中システムのデバッグを考慮した OS デバッグ機能”情報処理学会研究報告.[システムソフトウェアとオペレーティングシステム]pp.33-39.2003

[5] Mathieu Desnoyers et al :“The LTTng tracer; A low impact performance and behavior monitor for GNU/Linux”,OLS2006 Proceedings (2006)

[6] 篠崎孝一，大田弘，早水公二，星野光勇:モデル検査のデバッグへの適用，JaSST’06:Japan Symposium on Software Testing 2006

[7] 畑崎恵介，中村哲人，芹沢一:システムの挙動に対応して動作の切替えが可能なイベントトレーサ LKST の開発, 情報科学技術フォーラム一般講演論文集 2002(1), 177-178, 2002-09-13

[8] Unix Bench : http://www.tux.org/pub/tux/niemi/unixbench/

[9] Apache HTTP server benchmarking tool
: http://httpd.apache.org/docs/2.0/programs/ab.html

[10] Christina Warrender et al, “Detecting Intrusions Using System Calls”,Alternative Data Models,IEEE Symposium on Security and Privacy,pp.133-145,(1999)

[11] Roy A.Maxion,Kymie M.C. Tan, “Anomaly Detection in Embedded Systems” ,IEEE Transactions on computers,Vol51,No.2（2002）