Vol.2013-OS-127 No.6
Vol.2013-EMB-31 No.6
2013/12/3

Cyber-Physical System を指向したクラウドベースロギング機構の開発

PRAWEEN AMONTAMAVUT[†1]　　早川栄一[†2]

複数の組込み機器のシステムログデータと複数のセンサからのセンサデータを，クラウドストレージ上にロギングし，データにアクセスする機構について述べる．データは組込みシステムからストリーミングによってクラウドストレージ上にキャプチャされる．また，HTTP ベースのログ参照や機器操作 API を定義し，Web ブラウザから各機器へのロギング操作，およびログデータの参照を可能にした．この機構を基に，KernelShark によるセンサおよびシステムログの可視化機構を構築する．

Development of a Cloud Based Logging Mechanism for Cyber-Physical Systems

PRAWEEN AMONTAMAVUT[†1]　EIICHI HAYAKAWA[†2]

This paper describes a logging mechanism that is operating on a cloud storage that can handle log data from anywhere. The log data comes from multiple embedded devices called system log data and multiple sensors called sensor data. They are streamed from embedded devices and captured to cloud storage. Furthermore, we defined and implemented the HTTP based API for handling the log data, controlling the devices logging operation from browsers. We utilize the logging mechanism to implement a visualizing mechanism that can visualize both sensor and system log operated with KernelShark.

1. はじめに

Cyber-Physical System (CPS) は組込みシステムをはじめ，ネットワークを介して，コンピューティングシステムを統合する次世代の組込みシステムである．組込みシステムがターゲットとする分野は幅広いという特徴がある．家電製品やスマートフォンのように日常生活に根ざしたものから，電気自動車の制御のように分散されたデバイスでの制御も含んでいる．これらの特徴を CPS に統合するためには，組込みシステムの特徴を活かして，サイバー世界と物理世界との協調に動作可能な効率化が必要になる[1].

広範囲に分散した要素が協調して動作するシステムを開発するには，要素ごとの振舞いを把握し，監視できる枠組みが必要になる．開発者が，物理世界からのイベントの発生状況を把握し，それに応じたサイバー世界でのシステムの動作を把握していくには，ネットワーク上の複数の組込み機器に繋がる複数のセンサに対して，リアルタイムにその動作や生成されるデータを監視し，そのデータに対してアクセスできるサービスが必要になる．

Service-Oriented Architecture (SOA) [2] を利用して CPS を構築するサービス提供可能なモデルのアプローチがある[3]. HTTP プロトコル上に XML 記述を利用して，アプリケーション層の間を Simple Object Access Protocol (SOAP) [4]によってサービスを連携している．

しかし，これらのシステムではリアルタイム性を保証することは難しい．このようなシステム開発ではタスクスケジューリングや I/O，アプリケーションの動作といったシステム自身の振舞いを把握する必要がある．これらを計算機資源に限界のあるシステム上で提供しなければならない．また，分散された複数のデバイスから取得されたこれらのデータを統合，管理し，利用可能な形態にして提供する必要がある．

本研究は CPS をターゲットして組込みシステムから低オーバヘッドでシステムの振舞いをログとして取得する機構，およびその複数のデバイスからのログを統合してスケーラブルにアクセス可能にする機構の構成について述べる．この機構では，低オーバヘッドでログ取得を行うためにそれぞれの役割ごとに専用のサーバを用意している．また，インメモリによるログ管理機構を用意し，ログへのアクセスをスケーラブルに行えるようにした．また，デバイスへのログ取得制御機構も提供し，システムログについて一貫して扱える環境を提供する．

2. Cyber-Physical System を指向したクラウドベースロギングの概念

図 1 は CPS を指向したクラウドベースロギング機構の概念を表している．CPS ではセンサを搭載した組込みシステムという物理的エンティティを通して，ネットワークを介し各種の情報を取得する．これをサイバードメインで処理して，物理的エンティティであるアクチュエータによって，物理ドメインへ影響を及ぼす．このようなシステムの開発や運用においては，センサの情報に加えて，これら物

†1 拓殖大学大学院電子情報工学専攻
Graduate school of Electronic and Information Science, Takushoku University.
†2 拓殖大学工学部情報工学科
Department of Computer Science, Faculty of Engineering, Takushoku University.

理的エンティティ自身のシステム情報を取得する必要がある．これは，システム状況の把握や連携に不可欠だからである．

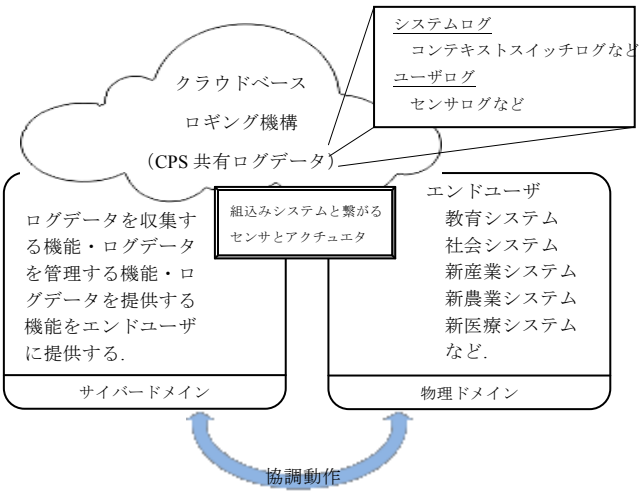

図 1　Cyber-Physical System を指向した
クラウドベースロギングの概念

本システムでは，システム開発や保守に必要となるシステム情報をログとして扱う．一つの組込みシステムから取得ができるこれらのログを Cyber-Physical System ログ（CPS ログ）と呼ぶ．これらはシステムログとユーザログの二つのログに分類することができる．

システムログは，組込みシステム上に動作するシステムソフトウェア・オペレーティングシステムに関するログを表す．このログはシステムの行動を把握することができる．次に例を挙げる．

- コンテキストスイッチログ（BSF ログ[5]）
- I/O ログ，など

これらはシステムのログサイズが大きく，ログ取得機構そのものがシステム全体の動作に影響を及ぼす．そこで，組込み機器のリアルタイム性を保証しつつ，効率的にロギングする必要がある．

もう一つはユーザログである．ユーザログは，システム上で動作するユーザプログラムが利用するデータである．組込みシステムでは，センサから情報を取得し，組込みシステム上で動作するアプリケーションが扱うことになる．具体的には次のとおりである．

- センサログ
- ユーザアプリケーション動作ログ，など．

ユーザログはシステム全体の動作に影響を与えないが，開発時にはアプリケーションやシステムの動作と関連づける必要がある．

システム開発を行う上では，システムログとユーザログとは不可分の関係にある．特に，組込みシステムではユーザからのイベントによってシステムの挙動が決まることが多いことから，これら二つの情報は統合して扱う必要がある．

また，CPS では複数の組込みシステムが協調して動作することから，これらのログが複数生成されることになる．協調システムではこれらの CPS ログを統合し，それを共有する必要がある．共有して利用が可能なログデータを Cyber-Physical System Shared Log (CPS 共有ログ)と呼ぶ．サーバにこれらのログを置き，アクセスを可能にすることで，開発者にとって，次のメリットが生じる．

- 分散システム全体の挙動を把握しやすくなる．
- 可視化ツールやモニタリングツールなど開発や運用に便利なツール群の開発が容易になる．
- 組込みシステムへフィードバックすることで，自律的な制御の基盤とすることができる．

マシンのサイズや規模はシステムの大きさによって大きく変わることから，ログ管理機構はある程度スケーラブルな形態で提供すべきである．

3. Cyber-Physical System を指向したクラウドベースロギング機構

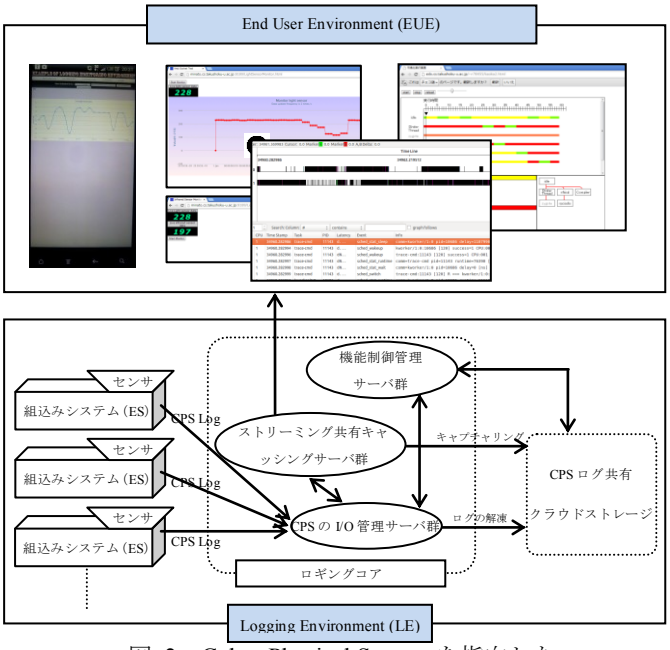

図 2　Cyber-Physical System を指向した
クラウドベースロギング機構

図 2 に CPS を指向したクラウドベースロギング機構を表す．システムは，ロギングに関する処理を行う「ロギング環境」(LE)と，その環境が提供する API を用いて開発者への可視化や組込みシステムの制御などを行う．また物理的エンティティからのロギングは，著者らが開発したロギング機構によって行う．これは低オーバヘッドでシステムログを取得可能なものである．これにユーザログの取得機構を追加し，各物理的エンティティからデータを収集する．API は，インターネット上に点在するエンティティからの通信を可能にするために HTTP ベースで開発した．また，ロギング環境そのものはクラウドサーバ上に実現し，組込みシステムの台数に合わせてスケーラブルにする．この

API を利用することで，組込みシステムのシステムロギングモジュールとクラウドベースロギング環境を容易に利用することができる．

3.1 ロギング環境（LE）

ロギング環境は複数の組込みシステムからのCPSログの収集・管理・提供が可能なロギングコア，およびこれらを共有ログとして管理するストレージ部分から構成する．これらはハイブリッドクラウド環境上で動作する．ロギングコアはプライベートクラウド上で動作し，CPSログ共有クラウドストレージはパブリッククラウド上での動作を想定している．

3.1.1 ロギングコアのクラウド

ロギングコアは，次の三つから構成される．

- 機能制御管理サーバ群
- ストリーミングデータ共有サーバ群
- ログ収集サーバ群

(1) 機能制御管理サーバ群

機能制御管理サーバは，クラウドストレージと協調してロギングシステム全体の制御を行うサーバである．これはロギング環境の機能動作を行うサーバ群である．CPS ログデータの収集・管理機能，アクチュエータの操作のための API を提供する．操作 API は LE の機能操作命令を他のサーバに渡すための拡張可能なアプリケーションインターフェースを定義する．図 3 は本機構が定義した API の基本形式を表す．

基本形式説明：

この API は W3C に定義された HTTP/1.1 を使用する．API を簡潔化するため，シンプルな GET メソッドを使用する．API は認証必須の「プライベート API」と認証不必要でオープンな「公衆 API」に分割する．アクチュエタの操作やシステムレベルに関する全ての API はプライベート API とする．

GET メゾッド：

GET /[API 形式] HTTP/1.1

[HTTP/1.1 に従う HTTP 要求ヘッダ]

API 形式：

etlog?[命令]&[命令オプション]

[命令]：instruction=[命令名]

[命令オプション]：optN=[オプション属性]

$N \epsilon \{1, 2, 3, \ldots\}$

オプションには拡張が可能な API の要素である．

各[命令]に対して必ず第 1 目の[命令オプション]がある．

第 1 目の命令オプション：

opt1=[ES_IP_TAG] | noneFix

[ES_IP_TAG]は組込みシステムに繋ぐためのタギングされた Gateway の IP Address である．

図 3 API の基本形式

これを基づいて，API の基本インターフェースを作り出す．図 3 は API の基本形式を示す． この API が利用可能なアプリケーションはウェブブラウザ上に操作するアプリケーションだけではなく，HTTP 通信機能可能なデバイスであればすべて利用可能であり，各システムのプラットフォームに非依存である．

この API は開発ツールだけではなく，組込みシステムからも利用可能である．これによって LE を通して結果を組込みシステムにフィードバックすることも可能になる．

表 1 HTTP ベース機構操作 API

命令名	命令の意味
startlogging	ロギング開始命令
stoplogging	ロギング停止命令
getbsflogsize	BSF ログサイズ取得命令
extractlog	ログ解凍命令
getlog	ログハンドル・ログ監視命令
ls	ログリスト・組込み機器リスト命令
sensornetwork	センサデータアクセス命令
actuatorcontrol	アクチュエタを操作する命令

(2) ストリーミングデータ共有キャッシングサーバ群

これはリアルタイムにログを監視する機能・キャプチャ機能を提供するサーバ群である．ユーザログのセンサログを複数のユーザにリアルタイムに監視する場合には組込みシステムの処理負荷を削減するために組込みシステムへのデータ要求の容量を削減する必要である．

図 4 はストリーミングデータ共有サーバにおけるキャッシングの手法を表す．図 4 の組込みシステム(A)はセンサログを一端サーバ内のメモリ空間内にストア(B)し，これを API やストレージサーバで利用可能なように，インデクスを付加してコピーしていく(C).

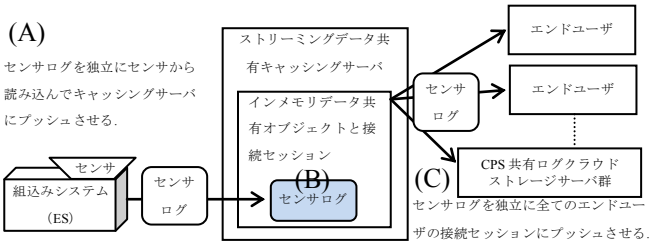

図 4 ストリーミングするためのキャッシング処理

組込みシステムからのログを一端キャッシングする機構を用意することで，組込みシステム側のロギングをサーバによってブロックすることがなくなる．これにより，組込みシステム側にログデータを滞留させて，計算機資源の消費を防ぐことができる．また，ストレージや API からのデータ要求の容量と送信セッション処理の手間を資源に余裕のあるサーバで処理できることから，多数の API セッションと同時にリアルタイムでセンサログをキャプチャするこ

とができる.

(3) **CPS の I/O 管理サーバ群**

ログ収集サーバ群は CPS ログを複数の組込みシステムから収集する機能を提供する. CPS ログのシステムログの収集は各組込みシステムのリアルタイム性を保つ必要がある. 本機構は図 2 の機構を用いた.

複数の組込みシステムをロギング可能にするためには各組込みシステムの接続セッションをログ収集サーバで扱う必要がある. しかし, 組込みシステムのリアルタイム性は各 ES の処理負荷とネットワークの帯域に依存することから, リアルタイム性を維持しつつ処理負荷とネットワーク帯域に応じた複数 ES のセッション処理を考えなければならない.

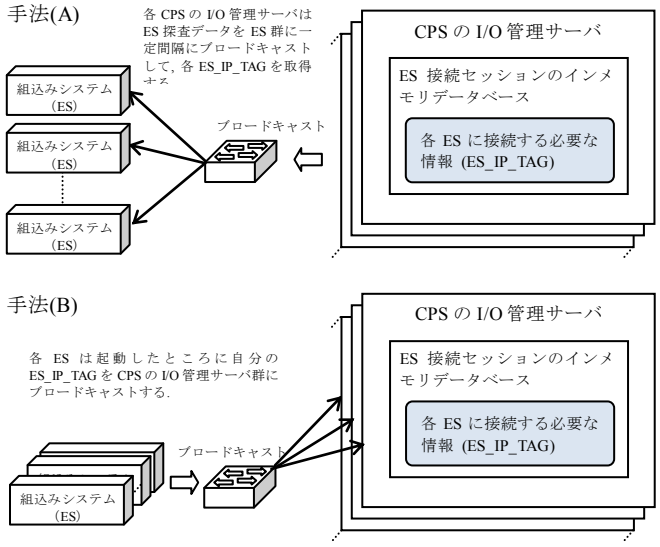

図 5 複数組込みシステムの自動接続セッション機能処理の手法案

図 5 は複数の組込みシステムを接続する手法を示す. 接続においては, 各 ES を接続するのに必要な情報を ES からどのように得て, それをどのように更新するかが重要である. 本システムではこれを ES_IP_TAG と呼ぶ. この更新手法には大きく二つがある.

手法(A)はログ収集サーバにおける ES_IP_TAG 接続状態を把握するために一定時間間隔ごとにログ収集サーバ群からブロードキャストをする必要があることから, ネットワーク帯域および ES の ES_IP_TAG 送信処理負荷が増やす. しかし, 手法(B)は ES が起動したところに一回だけ自分の ES_IP_TAG をログ収集サーバ群にブロードキャストする. ログ収集サーバにおける ES_IP_TAG の接続情報の更新は ES が負担するわけではなく各 ES_IP_TAG の送信先は十分な時間間隔に応答がない場合にその ES がデッドであることをログ収集サーバの負担で判断することができる. これで, ブロードキャスト処理回数を削減することができるので, ES の数が多くなっても, ブロードキャストするのにかかるネットワーク帯域はかわらない. これで, 本研究は手法(B)を選択する.

CPS ログの収集機能は (1) CPS ログ共有クラウドストレージに収集する機能 (2) リアルタイムに監視する同時にキャプチャする機能がある.

(1) **CPS ログ共有クラウドストレージに収集する機能**

これは圧縮された BSF ログであるシステムログを収集する機能である. ロギング開始命令が機能制御管理サーバ群から到着したら, ES 接続セッションのインメモリデータベースにおける ES_IP_TAG の接続情報からロギングフェースに移し, ログデータを収集する. 同様にロギング停止命令がきたら, これを ES に通知してロギング停止フェースに移す.

(2) **リアルタイムに監視する同時にキャプチャする機能**

これはセンサログというユーザログに対する機能となる. センサログはリアルタイムに監視する要求と CPS 共有ログクラウドストレージにキャプチャする要求がある. これらを効率に満たすためにはストリーミングデータ共有キャッシングサーバ群の機能が必要である.

ストリーミングデータ共有キャッシングサーバの準備フェースは次のステップの通りに実行する.

Step 1. 機能制御管理サーバがエンドユーザに対してサーバプッシュが可能なプロトコルにアップグレードして, ストリーミングデータ共有キャッシングサーバとログ収集サーバに命令と命令オプションを知らせる.

Step 2. ストリーミングータ共有キャッシングサーバは命令オプションに指定する ES_IP_TAG が ES に常に接続するかを確認する. 常に接続する場合は準備実行を終了する. そうではない場合には次を実行する.

Step 3. ストリーミングデータ共有キャッシングサーバはデータプッシュする接続セッションを作成する. インメモリ共有オブジェクトがない場合には新しくこれを生成する.

Step 4. ログ収集サーバは指定した ES_IP_TAG に対する接続情報をストリーミングデータ共有キャッシングサーバに知らせる.

Step 5. ストリーミングデータ共有キャッシングサーバは ES_IP_TAG を基づいて ES を接続する.

準備フェースが終了したら, 図 4 の通りにセンサログをメモリにキャッシングしながら, エンドユーザにストリーミングさせる. ただし, キャプチャの場合には CPS ログ共有クラウドストレージに指定した間隔分だけにプッシュさせる.

3.1.2 CPS ログ共有クラウドストレージ

これは CPS ログを長時間に保存することができ, ログデ

ータをエンドユーザに共有させるクラウドストレージである．クラウドストレージを構築するために，検討する必要である要因はディスク I/O とネットワークインタフェース I/O の帯域のジッタである．これらのジッタが少ないほど，ネットワークの混雑が少なくなるので ，リアルタイム性を保証しやすくなる．

4. 実験と評価

本研究は fio benchmark を利用して，ローカルディスク I/O を基準として，NFS と GlusterFS の I/O 帯域を測定する．表 2 は本実験の環境を表す．fio benchmark において，それぞれに同期に 4KB のブロックサイズで 8GB のデータを乱数に Read と Write を 2 分間にテスト実行して，図 6 にプロットする．

表 2　fio benchmark で I/O 帯域を測定する実験の環境設定

Spec.	Local File System	Network File System		
		NFS	GlusterFS	
		Node1	Node1	Node2
CPU	Intel(R) Core(TM)2 CPU 6400 @ 2.13GHz	Intel(R) Core(TM)2 CPU 6400 @ 2.13GHz	Intel(R) Core(TM)2 CPU 6400 @ 2.13GHz	AMD Phenom(tm) 9750 Quad-Core Processor
Network interface	RTL-8110SC/8169SC Gigabit Ethernet	RTL-8110SC/8169SC Gigabit Ethernet	RTL-8110SC/8169SC Gigabit Ethernet	RTL8111/8168 PCI Express Gigabit Ethernet
Network clock	66MHz	66MHz	66MHz	64MHz
IDE interface	Intel Corporation 82801HR/HO/HH (ICH8R/DO/DH)	Intel Corporation 82801HR/HO/HH (ICH8R/DO/DH)	Intel Corporation 82801HR/HO/HH (ICH8R/DO/DH)	Advanced Micro Devices [AMD] nee ATI SB7x0/SB8x0/SB9x0

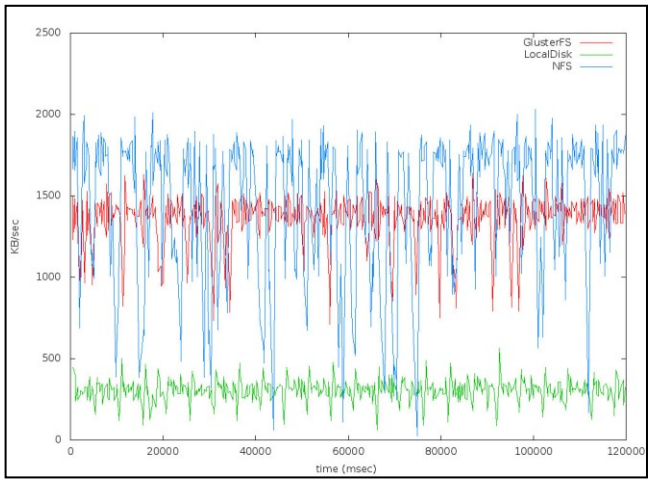

図 6　fio benchmark による random Read/Write の I/O 帯域測定結果

表 3　fio benchmark による I/O 帯域の jitter

File System	I/O 帯域 [KB/sec]		
	Minimum	Maximum	Jitter
Local File System	100	500	400
NFS	100	2100	2000
GlusterFS	750	1650	900

図 6 と表 3 はローカルディスクを基準として，GlusterFS と NFS のディスク I/O とネットワークインタフェース I/O の帯域のジッタを表すものである．図 6 の x 軸は fio benchmark を実行する時間を表し，y 軸は I/O 帯域を表す．表 3 を参照すると，GlusterFS は NFS より使用する I/O 帯域が少ないので，本機構を構築するのに最適なファイルシステムであることが明らかになった．

4.1 エンドユーザ環境

エンドユーザ環境は HTTP ベース機構操作 API を利用し，各エンドユーザに合わせたアプリケーションを構築するための環境である．本環境のユーザビリティを確認するため，ウェブブラウザによる可視化ツール・センサリアルタイムモニタリングツールと KernelShark による可視化ツールをユーザアプリケーションとした可視化機構を構築した．

ウェブブラウザによる可視化ツールとセンサのリアルタイムモニタリングツールは JavaScript の jQuery を基づいて，ツールを実装する．KernelShark は Linux のシステムログを可視化するツールである．これらを用いて組込みシステムのシステムログを遠隔に可視化するためには，本機構の HTTP ベース機構操作 API を使用するための HTTP 通信機能を追加した．

5. 関連研究

複数の CPU コアを持つ組込みシステムにおいて，リアルタイムコアと汎用コアに分割したハードウェアアキテクチャに対したコミュニティに関わるロボットをはじめ，汎用コアにログストリミング機能を追加した研究がある[6]．これは Stream Analysis Engine (SAE)を基づいて組込みシステム側の開発フレームワークとしてストリーミング API を組込みシステム開発者にしか提供しない．リアルタイムコアのシステムコンテキストを取るためには組込みシステムに内蔵する共有メモリを利用しネットワークを介して SAE にストレーミングさせる．また，リアルタイムコアに対しても Socket Log を転送する必要がある．CPU のコアが少ないアーキテクチャで対応することはできない．また，複数の物理的エンティティの動的な変化を把握しながらデバッグが必要である．しかし，SAE には複数の組込みシステムを統一して扱う機能は存在しない．

サイバー世界に存在する組込みシステム上に動作するシ

ステムソフトウェアにおいて物理世界のセンサ・アクチュエタを連帯するモデル開発の研究が存在する[3]. ここでは, 3 ティアモデルの「環境ティア」, 「制御ティア」, 「サービスティア」を紹介した. 組込みシステムのリアルタイム性が重要であると主張したが, リアルタイム性を保証する方法は触れられていない.

6. おわりに

本報告は CPS を志向したクラウドベースロギングの概念を述べた. サイバードメインと物理ドメインを協調に効率で統合するために, 各ドメインのエンティティの動作を把握しながら操作する必要がある. CPS を志向したクラウドベースロギング機構は組込みシステムの特徴を利用して, HTTP ベース操作 API を定義したことで, 複数の組込みシステムであっても複数のエンドユーザに, ロギング操作機能・ログハンドル機能・リアルタイムセンサログ監視機能・アクチュエタの操作機能を提供することができる.

CPS を志向したクラウドベースロギング機構を構築するために, 複数組込みシステムの接続手法を述べた. リアルタイムセンサログ監視機能はストリーミング共有サーバにおいて組込みシステムに影響を与えずに効率化したリアルタイムセンサ監視機能の構築が可能であった.

また, この機構を基づいてカーネルシャークによる可視化機構・ブラウザによる可視化機構・ブラウザによるリアルタイムセンサログの監視アプリを構築した.

今後の課題は CPS ログ共有クラウドストレージ部分の実装と評価, およびインメモリキャッシュの KVS（Key-Value Store）の適用を検討する.ことである. これらの実装後に実際の組込みシステムで本システムの有効性を検証, 評価していく.

謝辞 本研究は拓殖大学の先生方・早川研究室の皆様, 研究会・学会の皆様のご意見・ごコメントに, 謹んで感謝の意を表する.

参考文献

1) 独立行政法人科学技術振興機構研究開発戦略センター, “CPS(Cyber Physical Systems)基盤技術の研究開発とその社会への導入に関する提案”, 戦略プロポーザル, CEDS-FY2012-SP-05, March 03, 2013

2) R. Perry and M. Lycett, “Service-oriented architecture,” Applications and the Internet Workshops, ISBN 0-7695-1873-7, pp.116-119, January 27-31, 2003

3) Hyun Jung La and Soo Dong Kim, “A Service-based Approach to Designing Cyber Physical Systems,” IEEE/ACIS 9th International Conference on Computer and Information Science, ISBN 978-1-4244-8198-9, pp. 895-900, August 18-20, 2010

4) Gunnar Mein, Shankar Pal, Govinda Dhondu, Thulusalamatom Krishnamurthi Anand, Alexander Stojanovic, Mohsen Al-Ghosein, and Paul M. Oeuvray, “Simple Object Access Protocol ,”Microsoft Corperation, US006457066B1, September 24, 2002

5) Praween Amontamavut, Yuki Nakagawa and Eiichi Hayakawa, “組込みシステムを指向した Linux プロセスログ取得機構”, EMB, ISSN 09196072, 2011

6) Midori Sugaya, Hiroki Takamura, Youichi Ishiwata, Satoshi Kagami and Kimio Kuramitsu, “Online Kernel Logging and Analysis for Real-time Robotics Applications,” EMB, ISSN 1882-7764, 2012

付録

付録 A.1 HTTP ベース機構操作 API

命令名	命令オプション	命令の説明	API の区分種類	応答の説明
startlogging	opt1=[ES_IP_TAG]	システムログにおける BSF ログのロギングを開始する.	プライベート API	システムログのロギング開始の結果
stoplogging	opt1=[ES_IP_TAG]	システムログにおける BSF ログのロギングを停止する.	プライベート API	xml 記述表現であるログリストのデータ
getbsflogsize	opt1=[ES_IP_TAG]	ロギングフェースまたはロギング済フェースに BSF ログデータのサイズを取得する.	公衆 API	BSF ログというシステムログのサイズ
extractlog	opt1=[ES_IP_TAG] opt2=cts	システムの基準形式に圧縮されたシステムログを解凍する. 一切ロギングしていない場合には使用不可能である.	プライベート API	解凍完成失敗の結果
	opt1=[ES_IP_TAG] opt2=json	JSON 形式に圧縮されたシステムログを解凍する. 一切ロギングしていない場合には使用不可能である.	プライベート API	解凍完成失敗の結果
	opt1=[ES_IP_TAG] opt2=xml	XML 形式に圧縮されたシステムログを解凍する. 一切ロギングしていない場合には使用不可能である.	プライベート API	解凍完成失敗の結果
getlog	opt1=[ES_IP_TAG] opt2=cts	システムの基準形式に解凍したシステムログを要求する.	公衆 API	システムの基準形式のコンテキストスイッチのログデータ

	opt1=[ES_IP_TAG] opt2=json	JSON 形式に解凍したログを要求する．一切，解凍していない場合は使用禁止である． また，ログを部分にハンドルする場合には opt3=[yyyymmdd] opt4=[ハンドルしたいログの行番号] という命令オプションを追加する． 使用例： 2013 年 3 月 20 日のログにおける第 2 行目から第 4 行目までの範囲をハンドルするなら opt3=20130320 opt4=2~4 にする．	公衆 API	JSON 形式のログデータ
	opt1=[ES_IP_TAG] opt2=xml	XML 形式に解凍したログを要求する．一切，解凍していない場合は使用禁止である． また，ログを部分にハンドルする場合には opt3=[yyyymmdd] opt4=[ハンドルしたいログの行番号] という命令オプションを追加する． 使用例： 2013 年 3 月 20 日のログにおける第 2 行目から第 4 行目までの範囲をハンドルするなら opt3=20130320 opt4=2~4 にする．	公衆 API	XML 形式のログデータ
	opt1=[ES_IP_TAG] opt2=strbin	ロギング済のシステムの基準形式のシステムログを要求する．	公衆 API	システムの基準形式のシステムログデータ
ls	opt1=noneFix opt2=cts	cts 形式のログをフィルタしてログリストを要求する．	公衆 API	CTS 形式のログリストデータ
	opt1=noneFix opt2=json	json 形式のログをフィルタしてログリストを要求する．	公衆 API	JSON 形式のログリストデータ
	opt1=noneFix opt2=xml	xml 形式のログをフィルタしてログリストを要求する．	公衆 API	XML 形式のログリストデータ
	opt1=noneFix opt2=edconnected	接続されてる組み込み機器の状態データを要求する．	公衆 API	組込みシステムの接続状況のデータ
sensornetwork	opt1=[ES_IP_TAG] opt2=[SENSOR_MODULE] opt3=[SIGNAL_TYPE]	この命令は拡張が可能な命令であり，センシングデータを要求する命令である． [SENSOR_MODULE]は組込みシステム上におけるセンサを操作するモジュール名を表す． [SIGNAL_TYPE]はセンサから発信する信号の種類を表す． 第 4 番目のオプションからには各モジュールによって変わるので，ここに定義しない．拡張された各モジュールの操作 API を参照が必要である．	拡張モジュールレベルである API によって変わる．	センサログというユーザログを静的に応答するのはデフォルトであるが，リアルタイムに監視が可能なモジュールの場合には，サーバプッシュが可能なプロトコールにアップグレードして，キャッシングサーバ郡からセンサログをサーバプッシュ処理を行う．
actuatorcontrol	opt1=[ES_IP_TAG] opt2=[ACTUATOR_MODULE] opt3=[SIGNAL_TYPE] opt4=[SIGNAL_VALUE]	この命令は拡張が可能な命令であり，アクチュエタを操作する信号度数を渡す命令である． [ACTUATOR_MODULE]は組込みシステム上におけるアクチュエータを操作するモジュール名を表す． [SIGNAL_TYPE]はアクチュエータに渡す信号の種類を表す． [SIGNAL_VALUE]は信号の度数である． 第 5 番目のオプションからには各モジュールによって変わるので，ここに定義しない．拡張された各モジュールの操作 API を参照が必要である．	プライベート API	コンテンツ内容がない応答である．