eBPF を用いた EPICS アプリケーションの監視ツールの検討

INVESTIGATION OF MONITORING TOOLS FOR EPICS APPLICATION USING eBPF

佐々木信哉#, A)
Shinya Sasaki #, A)
A) High Energy Accelerator Research Organization (KEK)

Abstract

EPICS is used to build the control systems for the accelerators at KEK. In the EPICS-based control system, there are many IOCs and Channel Access clients, and records and process variables are complicatedly connected. This complexity makes it difficult to understand the status of the system. Therefore, it is necessary to monitor the status of the EPICS application to achieve stable operation and to investigate the cause of problems. On the other hand, eBPF, the technology to run sandboxed program in the Linux kernel space, is widely used for observability, networking and security. To investigate the usefulness of utilizing eBPF to monitor EPICS applications, two test programs were developed. The test programs showed that eBPF program can be dynamically loaded to monitor the status of EPICS applications without recompiling and rebooting IOCs. In addition, it makes it possible to obtain data that would be difficult to obtain using conventional approaches, such as tracing a record process chain.

1. はじめに

KEK では加速器の制御システム構築のために EPICS[1]が利用されている。EPICS はネットワーク分散型制御システムを構築するためのフレームワークである。EPICS では I/O Controller (IOC)上で EPICS アプリケーションを実行し、所望の制御システムを構築する。

EPICS を用いて構築された制御システムでは、多数の IOC や Channel Access (CA)クライアントが存在し、レコードや Process Variable (PV)が複雑に絡まりあうシステムとなる。制御システムの規模が大きくなるほどその複雑性は増し、システムの動作や状態を把握することが難しくなる。そのため、制御システムの安定な動作や、トラブル時の原因究明のためにも EPICS アプリケーションを監視し、その動作や状態を把握することは重要である。

既に様々な施設で EPICS アプリケーションの動作を監視するためのアプリケーションが利用されている。それらの多くはサポートモジュールなどの形態で IOC に組み込んで利用するか、IOC そのものには手を加えず、IOC の外側からその動作を監視するものとなっている。IOC に組み込んで利用する場合、IOC の起動前にコンパイルや設定を完了しておく必要がある。また、IOC 外からの監視では、取得できる情報に限りがある。

一方、Linux を利用するシステムにおいて eBPF[2]を利用するアプリケーションが近年広く利用されるになってきている[3]。eBPF を利用すれば、OS のカーネル空間やユーザー空間で発生するイベントにアタッチし、そのイベントに付随する情報の取得や一部動作の変更が可能となる。これを EPICS アプリケーションの監視に利用することで、IOC には直接手を加えず、その外側から IOC の動作や内部状態を従来よりも詳細に観測できると考えた。そこで、eBPF を利用した EPICS アプリケーションの監視ツールを試作し、その有用性について検討した。

本稿では、はじめに既存の監視アプリケーションに関してまとめ、その課題を整理する。続いて、eBPF の動作と特徴をまとめる。そして、試作した eBPF を利用する監視アプリケーションを説明し、その利点と課題に関してまとめる。

shinya.sasaki@kek.jp

2. 既存の監視アプリケーション

この章では、EPICS アプリケーションの監視に用いられる既存のアプリケーションを、その動作形態から以下の 3 つに分類して整理する。

A. レコードを経由した監視
B. IOC から直接外部サービスへ情報を送信する監視
C. IOC の外部から動作や状態を観察する監視

以下からは、それぞれの動作形態に分類した監視アプリケーションをまとめ、既存の監視アプリケーションの課題について整理する。

2.1 レコードを経由した監視

IOC の内部状態や障害判別の状況をレコードとして用意しておき、アーカイバなど加速器運転で用いられるソフトウェアや専用の CA クライアントを利用した監視が行われている。

例えば、J-PARC では PCMON をベースにしたデバイスサポートを IOC に組み込み、IOC のリソースや状態の監視を行っていることが報告されている[4]。レコードの値をアーカイバで記録することで、障害時の原因究明にも利用されている。SuperKEKB では監視対象の IOC に iocStats で提供されるレコードを用意し、IOC の計算機資源の状態や CA の接続状況を取得できるようにしている[5]。レコードの値は Zabbix において収集され、異常の検知や原因究明に利用されている。PF 及び cERL では IOC のタイムスタンプレコードや、ある 2 つの数値レコードの差分を監視する監視用 IOC を用意して IOC の監視を行っていることが報告されている[6]。このシステムでは CSS Alarm と連携してアラートの通知を行うほか、Python で作成したメール通知システムとも連携して動作する。

2.2 IOC から直接外部サービスへ情報送信する監視

レコードを経由せず、内部情報を直接外部サービスに送信するアプリケーションを IOC に組み込むと、レコードの形式に縛られない多様な監視を行うことが可能となる。

SLAC の LCLS では IOC に caPutLog[7]を組み込み、CA Put 操作のログを記録していることが報告されている[8]。ログはレコードを経由せず Logstash へ送信され、Elasticsearch へ保存、Grafana 上で表示されている。

EPICS の PV に関するディレクトリサービスを提供する Channel Finder[9]では、その情報を IOC 内で動作する RecSync[10]というモジュールで収集する。RecSync が取得した情報は独自のプロトコルでサーバーに送信される。Channel Finder で収集した情報には PV 名だけでなく、その PV のメタデータを含んでいる。Channel Finder はアーカイバへの登録の自動化などのシステム管理にも利用されるが、障害時の原因究明にも有用となる。

2.3 IOC の外部から動作や状態を観察する監視

IOC そのものに手を加えず、外部サービスからその動作を監視するシステムは、監視による IOC への影響を少なくし、多数の IOC の監視を一括して行いやすい。

SuperKEKB では IOC や CA クライアントから送信されるブロードキャストパケットを監視し、可視化するシステムを運用している[11]。

RIBF では主に IOC の通信レイヤの部分を外部から死活監視することが試みられている[12]。監視対象となる IOC の情報は EPICS PV 管理システムから取得する。EPICS PV 管理システムは IOC のスタートアップスクリプトや EPICS データベースファイルをパースすることで情報を収集して保存する。LCLS でも同様に、IOC のスタートアップスクリプトなどをパースして IOC に関する情報を提供することが試みられている[13]。これらのシステムで収集される情報は Channel Finder と重複する部分もあるが、IOC に直接手を加えないという点で異なる。

2.4 既存のアプリケーションの課題

上記の監視アプリケーションは EPICS アプリケーションの状態や動作を監視するのに有用である。一方、その動作形態のために実現が難しい機能もある。

A や B の形態では、サポートモジュールや監視用のレコードを監視対象の IOC へ事前に組み込むことが必要となることが多い。そのため、事前に組み込んでいなかった IOC へ動的にロードすることができない。また、PCAS[14]や Python のみで動作する IOC[15-17]などの EPICS Base を利用していない IOC ではサポートモジュールを利用出来ない。その場合、それぞれのアプリケーションで必要となる監視を別々に実装する必要がある。

一方、A や C の形態で IOC に直接手を加えず、その外側から情報を取得するような場合、IOC 内部の詳細な情報を得ることが難しい。また、A や B の形態でサポートモジュールを組み込んでいたとしても、取得できる情報は EPICS Base で利用できる API に制限される。

eBPF を利用する事で上記の課題を解決し、IOC の外側からその動作や内部状態を従来よりも詳細に観測できると考え、eBPF を用いた監視アプリケーションの有用性に関して検討を行った。

3. eBPF

eBPF は Linux カーネル空間のサンドボックス化された環境でプログラムを実行する機構である[18]。eBPF プログラムはイベントドリブンで動作し、システムコールの呼び出しなどのイベントを起点にして実行される。eBPF プログラムの引数と戻り値はアタッチするイベントによって異なる。アタッチするイベントによっては、戻り値の値でカーネルの振る舞いを変更できるものもある。eBPF プログラムは実行時に Just-In-Time (JIT)コンパイルによってバイトコードから機械語のコードへ変換される。機械語のコードはロードされる前に検証器による確認が行われるため、プログラムの安全性が担保される。

eBPF プログラムは eBPF Map を利用する事でデータの保存および取得が可能である。Map はユーザー空間からもアクセスが可能であるため、eBPF プログラムとユーザー空間のプログラム間の相互のやり取りにも利用できる。そのため、eBPF プログラム本体の他に、それをカーネルにロードし、Map を介して得た情報を所望の形態で利用するユーザー空間のプログラムを組み合わせて利用することもできる。

4. eBPF を利用した監視アプリケーション

eBPF を用いた監視アプリケーションの有用性を検証するために、「CA Put 操作の監視」および「レコードのプロセスのトレース」を行うプログラムを作成した。どちらの開発にも BPF Compiler Collection (BCC)を用いた。この章では、まず開発に利用した BCC に関する説明を行う。そして、開発した 2 つのプログラムに関して、その背景と eBPF プログラムおよびユーザー空間のプログラムの動作、その用途と課題についてまとめる。

4.1 BPF Compiler Collection

BPF Compiler Collection (BCC) は eBPF プログラムを作成するためのツールキットである[19]。ユーザー空間のコード開発に Python が利用できるため、比較的簡単に eBPF プログラムの開発ができる。

BCC では、実行時に eBPF プログラムコードがバイトコードに毎度コンパイルされる。そのため、カーネルバージョンの異なる環境へのプログラムの移植も比較的容易に行うことができる。しかし、プログラムの実行の度にコンパイル処理が行われるため、実行に時間がかかる。また、コードを実行したいすべてのマシンに BCC の実行環境を用意しておく必要がある。

4.2 CA Put 操作の監視

4.2.1 背景

EPICS を利用した制御システムでは、アクセスセキュリティなどによって制限を加えていない限り、ネットワーク上のどの CA クライアントからも Put 操作によって PV の値を変更することができる。そのため、PV に対する Put 操作を誰がいつ行ったかを記録しておくことは、PV の値が予期せぬ値に変更された際の原因究明に有用な情報となる。

caPutLog を利用することでも Put 操作を記録することができる。しかし、caPutLog は EPICS のサポートモジュールであるため、IOC の起動前に IOC に組み込んでおく

必要がある。また、EPICS Base を用いていない EPICS アプリケーションでは利用することが出来ない。

IOC が動作する計算機上で cashark[20]を適用した Wireshark を実行することで、Put 操作を記録することも可能である。しかし、cashark ではプロトコルの解析を行うポート番号が 5064 番および 5065 番のみとなっている。したがって、同じ計算機上で複数の IOC を立ち上げている場合など、5064 番以外のポートで IOC が立ち上がっていると、プロトコルの解析を行うことが出来ない。また、CA プロトコルでは PV 名を接続の確立時にしかやりとりせず、以降は Client ID (CID) や Server ID (SID) という識別番号で処理する[21]。そのため、どの PV に対して接続が確立し、Put 操作が行われたかを識別するためのプログラムを別に用意する必要が出てくる。

IOC 外部で動的に起動し、複数の IOC が起動している場合であっても Put 操作を記録する機能の実現のために、eBPF を用いた Put 操作の監視アプリケーションを試作した[22]。

4.2.2 eBPF プログラム

eBPF ではプログラムタイプによってアタッチ可能なイベントや呼出可能なヘルパー関数などが決まる[23]。本プログラムでは、BPF_PROG_TYPE_SOCKET_FILTER を利用した。このタイプの eBPF プログラムは Wireshark のように、ネットワークインターフェース上でやりとりされるパケットのキャプチャのために利用される。送受信されるパケットのデータのコピーが、ユーザー空間のプログラムで開いた raw ソケットへ受け渡される。eBPF プログラムは、ユーザー空間へパケットのデータを渡す前に実行され、その戻り値によってユーザー空間にパケットを受け渡すか、ドロップするかのフィルターを行う[24]。

作成したプログラムのブロック図を Fig. 1 に示す。作成した eBPF プログラムでは、以下の条件を満たすパケットのみをユーザー空間へ受け渡すように作成した。

- TCP/IP のパケットである。
- ペイロードが 16 バイト以上である。これは、CA プロトコルでやり取りされるデータのヘッダーサイズであり、最小のデータサイズである。
- ペイロードのデータが監視したい CA プロトコルのデータと一致している。具体的には、データのヘッダー領域に含まれる command と payload_size が一致することを確認する。

なお、実装の簡素化のため、フラグメント化されたパケットのリアセンブルは行っていない。また、ペイロードのデータタイプは DBR_STRING であることを前提としている。

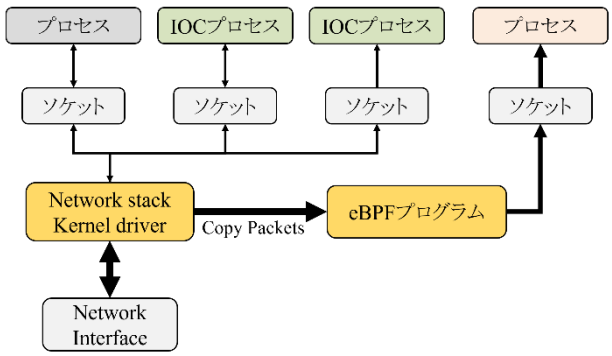


Figure 1: Block diagram of the eBPF program for monitoring the CA put operations. The eBPF program filters the packets copied from the network stack and passes the CA packets to the program in user space. The program in user space analyzes the filtered packets and prints information about the put operations to a standard output.

4.2.3 ユーザー空間のプログラム

ユーザー空間のプログラムでは、eBPF プログラムによるフィルターを通過したパケットの解析を行う。チャネル確立時のやりとりを解析し、下記の情報と PV 名を紐づけるテーブルの作成を行う。

- CA サーバーの IP アドレスとポート番号
- CA クライアントの IP アドレスとポート番号
- CID
- SID

Put 操作に当たる Write コマンドのパケットが届くと、そのパケット内の情報とテーブルの情報を照らし合わせ、どの PV に対する操作であるかを判別する。そして、各情報を標準出力する。

実行結果の例を Fig. 2 に示す。今回作成したプログラムでは、Put 操作が行われた PV 名およびその値、Put 操作の実行元および実行先のホストの情報が標準出力へ出力される。

4.2.4 用途と課題

本プログラムは IOC に caPutLog のようなサポートモジュールを組み込むことが難しい場合や、既に IOC が起動していてサポートモジュールを組み込むことが出来ない場合での利用が考えられる。情報の出力先を標準出力から Elasticseach などのログ収集システムに変更すれば、より柔軟な表示も可能となる。

一方、PV 名と紐づける必要のある情報はチャネルの確立時にしかやり取りされない。そのため、eBPF プログラムがロードされる前から接続が確立している PV に対しては、PV 名の照合ができないことが課題として挙げられる。

4.3 レコードのプロセスのトレース

4.3.1 背景

IOC における EPICS データベースでは、レコードがプロセスした際にそのレコードの持つ値の更新や、値の出力などの処理が実行される。また、レコード同士をリンクすることで、レコードを連鎖的にプロセスさせることができる。多くのレコードが連鎖的にプロセスするような場合、なぜそのタイミングでプロセスしたのかを追跡することが難しい場合がある。そのため、レコードがいつどのような順序でプロセスしたか、またどのような値に変化したかという情報は、システムの動作を追跡する際に有用な情報

```
$ sudo python3 caput-monitor.py -i enp0s8
binding socket to 'enp0s8'
load eBPF program
start
Write: ET_SASAKI:TEST1 1 from 192.168.56.1:50664 to 192.168.56.101:5064
Write: ET_SASAKI:TEST2 1 from 192.168.56.1:50676 to 192.168.56.101:5064
Write: ET_SASAKI:TEST3 1 from 192.168.56.1:50662 to 192.168.56.101:33369
```

Figure 2: Example of the result of running the program for monitoring the CA put operations.

となる。

EPICS データベースにおいても、レコードのプロセスをトレースする機能が用意されている。標準的なレコードには TPRO フィールドが備わっており、このフィールドの値を 0 以外に設定することで、そのレコード以降のプロセスのトレースが有効化される[25]。しかし、トレースを有効化したレコードより前にプロセスしたレコードの情報を取得することはできない。また、プロセスしたレコードの名前は取得できるが、そのレコードの値などは取得できない。加えて、iocsh 上の標準出力に出力されるため、外部ツールでの利用が難しい。

また、プロセスのトレースに利用できるような関数やフックは 2024 年 7 月現在 EPICS Base では提供されていない。そのため、サポートモジュールによってレコードのプロセスの情報をトレースすることも難しい。

一方、IOC の EPICS アプリケーションが共有ライブラリを利用するようにコンパイルされている場合、レコードのプロセス時に、EPICS Base の libdbCore のライブラリに含まれる dbProcess 関数が呼び出される。そのため、uprobe によって libdbCore の dbProcess 関数にアタッチする eBPF プログラムを作成することでその動作を追跡することができる。そこで、dbProcess 関数にアタッチし、プロセスの動作を追跡する eBPF プログラムを試作した[26]。なお、対象とする EPICS Base のバージョンは 7.0.8 とした。

4.3.2 eBPF プログラム

作成したプログラムのブロック図を Fig. 3 に示す。プレセスのトレースのために、dbProcess を含む以下の 3 つの関数にアタッチする eBPF プログラムを作成した。

- dbProcess
- dbCreateRecord
- dbGetRecordName

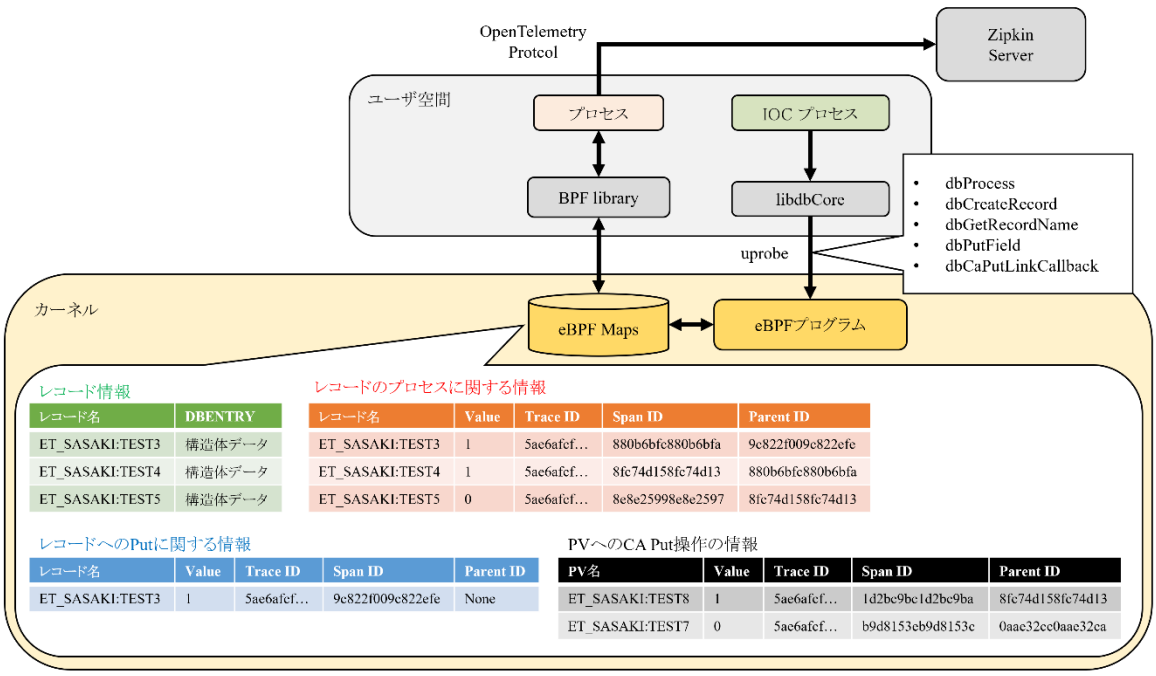
dbCreateRecord は IOC 起動時にレコードを作成する時、dbGetRecordName は iocsh において dbl コマンドを実行する時に呼び出される関数である。これらの関数へアタッチするプログラムは、IOC の持つ各レコードの名前とそのレコードに関する構造体のデータを紐づけて Map に保存するために利用している。IOC 起動前から監視プログラムを起動していれば dbCreateRecord からレコードの情報を取得する。監視プログラムが IOC 起動後に起動された場合は、その IOC の iocsh 上で dbl を実行することで、その IOC の持つレコードの情報を取得することができる。dbProcess にアタッチするプログラムは、dbProcess の開始時および終了時のレコードの情報を記録する。

レコードへの値の Put およびレコードからの CA Put も監視するために以下の関数にもアタッチしている。

- dbPutField
- dbCaPutLinkCallback

dbPutField はレコードに対する Put 操作が行われる際に呼び出される関数である。そのため、レコードのプロセスのきっかけとなる Put 操作がいつ行われたのかという情報を取得できる。一方、dbCaPutLinkCallback はレコードから CA リンクで接続された PV へ Put 操作が行われる際に呼び出される関数である。そのため、このプロセスの連鎖において、どの PV に対する CA Put 操作が行われるかという情報が取得できる。

eBPF プログラムで収集したプロセスの情報は、Map を介してユーザー空間のプログラムへ渡される。



レコード名	DBENTRY
ET_SASAKI:TEST3	構造体データ
ET_SASAKI:TEST4	構造体データ
ET_SASAKI:TEST5	構造体データ

レコード名	Value	Trace ID	Span ID	Parent ID
ET_SASAKI:TEST3	1	5ae6afcf…	880b6bfc880b6bfa	9c822f009c822efe
ET_SASAKI:TEST4	1	5ae6afcf…	8fc74d158fc74d13	880b6bfc880b6bfa
ET_SASAKI:TEST5	0	5ae6afcf…	8e8e25998e8e2597	8fc74d158fc74d13

レコード名	Value	Trace ID	Span ID	Parent ID
ET_SASAKI:TEST3	1	5ae6afcf…	9c822f009c822efe	None

PV名	Value	Trace ID	Span ID	Parent ID
ET_SASAKI:TEST8	1	5ae6afcf…	1d2bc9bc1d2bc9ba	8fc74d158fc74d13
ET_SASAKI:TEST7	0	5ae6afcf…	b9d8153eb9d8153c	0aae32ec0aae32ca

Figure 3: Block diagram of the eBPF programs for tracing record process chains. The eBPF programs are attached to the functions of the libdbCore library that are called by IOC processes. Each eBPF program collects the data and stores it in eBPF maps. The program in user space reads the data from the maps and sends it to the Zipkin server using the OpenTelemetry protocol.

4.3.3 ユーザー空間のプログラム

ユーザー空間のプログラムでは、Map を介して渡された情報を OpenTelemetry のトレースデータとして処理する。OpenTelemetry はテレメトリデータと称されるトレースやメトリクス、ログといったデータを測定、生成、収集、エクスポートするためのフレームワークである[27]。eBPF プログラムにおいて Trace ID や Span ID、Parent ID を各イベントに付与しておくことで、その順序関係を OpenTelemetry の形式で扱うことができるようにしている。

OpenTelemetry では OpenTelemetry Protocol (OTLP) をサポートするアプリケーションにテレメトリデータを送信することで、そのデータを活用することができる[28]。例えば、OTLP によるトレースデータの受信に対応する分散トレーシングシステムとして、Jaeger[29] や Grafana Tempo[30]、Zipkin[31]などがある。本プログラムでは Zipin へデータを送信するように作成した。

本プログラムから送信したデータを Zipkin 上で表示した画面のスクリーンショットを Fig. 4 に示す。トレースしたデータが画面左側に図として表示され、レコードのプロセスや Put 操作の順序関係が明らかになる。また、画面右側では各イベントの詳細を確認することができる。

4.3.4 用途と課題

本プログラムは障害発生時やデバッグ時に動的にロードし、レコードがプロセスする順序関係やその値に問題がないか確認するための利用が考えられる。レコードに Put された値や、外部の PV への CA Put 操作も記録されるため、IOC の外部との接続状況に関しても確認できる。

本プログラムを常時動かすことで、障害発生時の動作に問題がなかったか後から確認することもできる。しかし、常時動かし続けるような場合、記録されるデータが非常に多くなってしまう。そのため、不要なデータの間引きなどの処理が必要になってくる。

また、実現したい処理を EPICS データベースの外部から Put 操作を主として実装している場合、その処理を追跡することは本プログラムでは難しい。例えば、EPICS Sequencer[32]によって構築された処理では、PV の値の変更が CA クライアントから Put 操作によって行われるため、どのような条件のもとにレコードの値が変更されたのかを追跡することが出来ない。

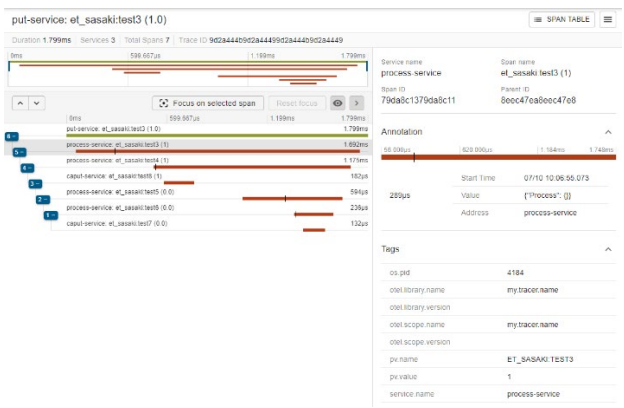


Figure 4: Screenshot of the trace data on Zipkin. The graph on the left shows the order of the process chain. The details of the event are shown on the right.

5. eBPF を利用する利点と課題

4 章において挙げた 2 つの eBPF プログラムの開発を通して、eBPF の利用により既存の EPICS アプリケーションに手を加えず、その動作の監視ができることを確認できた。eBPF プログラムは動的にロードすることができるため、既存のアプリケーションの再起動も不用であった。また、レコードのプロセスのトレースのように従来の方法では観測が難しい情報も、eBPF を利用する事で取得することが出来た。

一方、eBPF プログラムを動的にロードできたとしても、必要とする情報が十分に取得できない場合があることも確認できた。CA Put 操作の監視においては、プログラムのロード前に確立されたチャネルの PV 名を取得することは出来なかった。また、プロセスのトレースにおいては、IOC 起動前にプログラムをロードしていない場合、レコードの情報を取得するために iocsh 上で dbl コマンドを実行する必要があった。

eBPF を利用できる OS が限定されることも課題として挙げられる。eBPF は 2024 年 7 月現在、Linux での利用が中心となっているため、利用できる環境が限定されてしまう。また、カーネルバージョンが古い Linux においても利用する事ができない。

プログラムの移植性も課題として挙げられる。eBPF プログラムではカーネル空間やユーザー空間のデータ構造にアクセスする。Linux カーネルや EPICS Base のバージョンが異なると eBPF から利用しているデータ構造が変化する可能性がある。Linux カーネルのバージョンの違いに対しては、CO-RE アプローチ[33]を採用することで解決できる。しかし、EPICS Base 内のデータ構造の違いに対してどのように対応するかは検討が必要である。また、EPICS Base のバージョンの違いによってプログラム構造が変わった際に、uprobe でアタッチしていた関数が利用されなくなる可能性も考慮する必要がある。

6. まとめ

EPICS アプリケーションを監視する eBPF プログラムを試作し、eBPF を利用する利点と課題について検討を行った。eBPF を利用することで、監視アプリケーションを動的にロードし、動作や状態の監視に利用できることが確認できた。また、プロセスのトレースなど、従来のアプリケーションでは観測の難しい情報も取得できることが確認できた。一方、eBPF を利用できる OS が限定されていることや、プログラムの移植性に関する課題も確認できた。今回試作したプログラムは eBPF の有用性の検証のために作成した。そのため、本番環境において利用するためには移植性を考慮した実装への変更や、負荷試験が必要となる。

参考文献

[1] EPICS-Controls website, https://epics-controls.org/
[2] eBPF, https://ebpf.io/
[3] eBPF Case Studies, https://ebpf.io/case-studies/
[4] H. Nemoto *et al.*, “IOC Surveillance System for J-PARC MR Control”, Proc. 9th Annual Meeting of Particle

Accelerator Society of Japan, Toyonaka, Japan, Aug. 2012, pp. 745-748.

[5] S. Sasaki T. T. Nakamura and M. Hirose, "Monitoring system with Zabbix at SuperKEKB", Proc. PASJ2019, Kyoto, Japan, Jul.-Aug. 2019, pp. 596-599.

[6] Y. Kameta and T. Obina, "Development of equipment monitoring system for PF and cERL", Proc. PASJ2018, Nagaoka, Japan, Aug. 2018, pp. 593-596.

[7] caPutLog, https://github.com/epics-modules/caPutLog

[8] K. R. Lauer, "Centralized Logging and Alerts for EPICS-based Control Systems with Logstash and Grafana", presented at the 19th Int. Conf. Accel. Large Exp. Phys. Control Syst. (ICALEPCS'23), Cape Town, South Africa, Oct. 2023, paper THPDP089, unpublished.

[9] Channel Finder, https://channelfinder.readthedocs.io/en/latest/

[10] RecSync, https://github.com/ChannelFinder/recsync

[11] S. Sasaki, T. T. Nakamura and M. Hirose, "Monitoring System for IT Infrastructure and EPICS Control System at SuperKEKB", Proc. ICALEPCS'19, New York, NY, USA, Oct. 2019, pp. 1413-1417. doi:10.18429/JACoW-ICALEPCS2019-WEPHA134

[12] A. Uchiyama and M. Komiyama, "An Attempt to Implement the Alive Monitoring System for Reliable EPICS-based RIBF Control System", Proc. PASJ2016, Chiba, Japan, Aug. 2016, pp. 664-667.

[13] K. R. Lauer, "whatrecord: A Python-Based EPICS File Format Tool", Proc. ICALEPCS'23, Cape Town, South Africa, Oct. 2023, pp. 1206-1211. doi:10.18429/JACoW-ICALEPCS2023-THMBCMO08

[14] PCAS, https://github.com/epics-modules/pcas

[15] pythonSoftIOC, https://github.com/dls-controls/pythonSoftIOC

[16] caproto, https://github.com/caproto/caproto

[17] PCASpy, https://github.com/paulscherrerinstitute/pcaspy

[18] eBPF Documentation What is eBPF?, https://ebpf.io/what-is-ebpf/

[19] BPF Compiler Collection (BCC), https://github.com/iovisor/bcc

[20] cachark, https://github.com/mdavidsaver/cashark

[21] Channel Access Protocol Specification, https://docs.epics-controls.org/en/latest/internal/ca_protocol.html

[22] CA Put monitor with eBPF, https://github.com/sasaki77/caputmon-ebpf-pasj2024

[23] eBPF Program Types, https://libbpf.readthedocs.io/en/v1.4.5/program_types.html

[24] Program type BPF_PROG_TYPE_SOCKET_FILTER, https://ebpf-docs.dylanreimerink.nl/linux/program-type/BPF_PROG_TYPE_SOCKET_FILTER/

[25] IOC Test Facilities, https://docs.epics-controls.org/en/latest/appdevguide/IOCTestFacilities.html

[26] Process chain trace with eBPF, https://github.com/sasaki77/proctrace-ebpf-pasj2024

[27] OpenTelemetry, https://opentelemetry.io/

[28] OTLP Specification, https://opentelemetry.io/docs/specs/otlp/

[29] Jaeger, https://www.jaegertracing.io/

[30] Grafana Tempo, https://grafana.com/oss/tempo/

[31] Zipkin, https://zipkin.io/

[32] Official mirror of the stable branch of the EPICS Sequencer, https://github.com/epics-modules/sequencer/

[33] libbpf Overview, https://libbpf.readthedocs.io/en/v1.4.5/libbpf_overview.html