オンプレクラウド移行を支える eBPF：ネットワーク遅延解析とミドルウェアのデバッグ

rev0.1 Junji Hashimoto

自己紹介

- 橋本順之
- グリー、12年
- インフラエンジニア
  - シニアリードエンジニア
  - KVS Unitのチームリーダー
- 業務
  - KVSの運用、パフォーマンス分析(本日の焦点)
  - Web3のバリデータ運用(DB=LevelDB、運用自動化=LLM)
- 社外活動
  - hasktorch(Haskel)やgpu.cpp(WebGPU)の機械学習のフレームワーク開発
  - AIの安全性の分析と修正（画像系)

目次

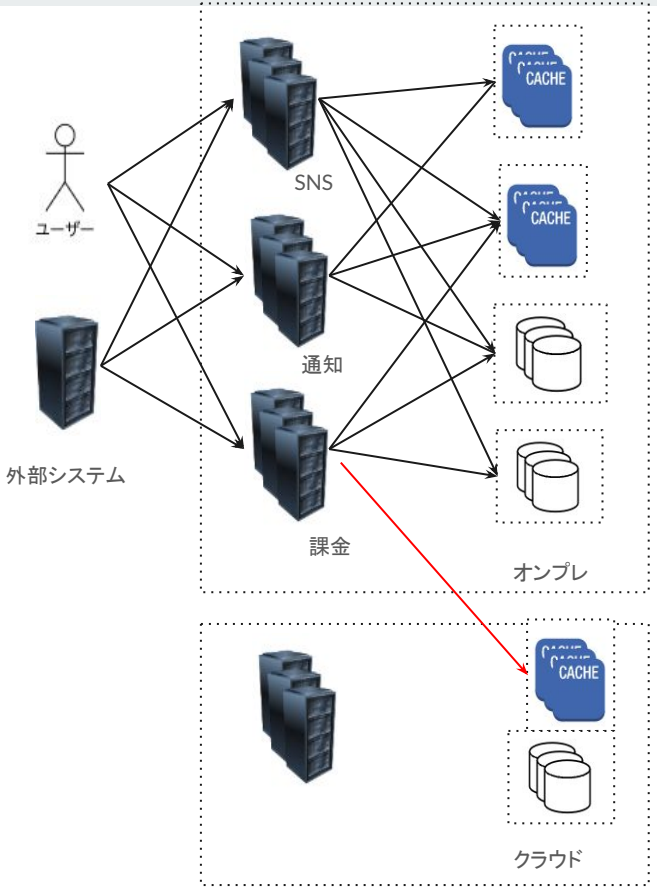
- eBPFを使う動機
- 利用事例（ユースケース）の紹介
  - データセンターの移設（遅延の原因分析と影響範囲調査）
    - ネットワーク遅延が大きい
    - 無停止で移設したい。
  - ミドルウェアへのアクセスのスパイクの原因分析
    - C++のトレースをする。
  - PHPエクステンションのデバッグ（ Cのエクステンションの挙動の調査）
    - 接続の永続化が効いてないように見える。
    - 本番環境でprintfデバッグをしたい。
  - PHPエクステンションのデバッグ（セグフォの分析）
    - ローカルでは再現しない。
    - 本番環境で一時間に一度程度発生。
    - コアダンプを見てもよくわからない。
  - PHPのトレース
    - 本番環境における詳細なトレースがほしい。
    - PHP のトレースをする。
    - mysqlのクエリも含めてトレースしたい。
- まとめと課題

eBPFを使う動機

- コーナーケースでの分析がしたい。
- アプリの改修なしにメトリックが取れる。
  - 既存の手法: datadogやnewrelic
    - 自動で取れないメトリック・スパンに対してアプリの改修が必要。
  - あらかじめログを仕込む必要がない。
    - ログを仕込むためにソフトウェアを更新したくない。
    - 本番環境で、静的な言語の再コンパイルなしに解析ができる。
- サービスへの影響が少ない。
  - uprobe（ユーザー空間の関数のトレース）
    - 一回あたり1us程度でメトリックが取れる。
    - 使いすぎは注意！（コンテキストスイッチが入る。）
  - ユーザーデータへのアクセスが容易。
    - データのパーサーを書く必要がない。
    - アプリの解析結果を利用できるため高速。
- リアルタイムな分析
  - 個別のアクセスごとの解析（キーやクエリ、個々のアクセスタイム）

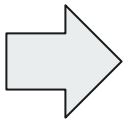
Case1: データセンターの移設

- システム
  - Ubuntu Focal
  - オンプレ、VM
  - Web
    - Apache+PHP
  - ミドルウェア
    - Cache: Memcached互換(Flare)
    - DB: MySQL
- データセンターを跨ぐとネットワーク遅延が大きい。
  - 十倍程度違う。
  - 数百us から数msに変わる。
- 無停止で移設したい。
  - 単一のシステムでないので、メンテが入れられない。
- 複数のサービスのメッシュ
- どのサービスがどのサービスをどの程度使っているか分からない。
- 移設の優先順位が立てられない。
- 移設後の遅延がどのミドルウェアへのアクセスによるものかわからない。
- DBよりCacheの方がアクセス頻度が多いので、遅延の影響を受けやすい。



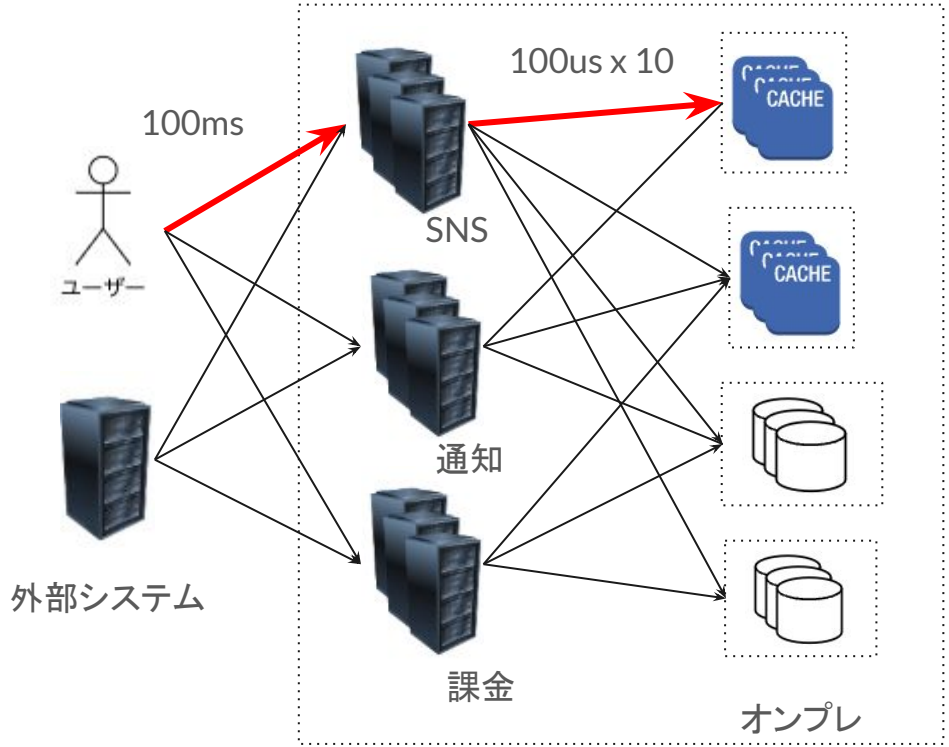
データセンター間のレイテンシー

- オンプレ間
  - 0.1ms以下
- AWSのAZ内
  - 0.2ms以下
- AWSのAZ間
  - 0.5ms～1.7ms
- AWSとオンプレ間
  - 1.7ms～3.8ms
- オンプレでのMemcachedのレスポンスタイム
  - 0.5ms
- オンプレからAWSへのMemcachedのレスポンスタイム
  - 4ms



データセンターを跨ぐと
レイテンシーは
およそ20倍

解析対象のシステム



例

WEB

一回のリクエスト

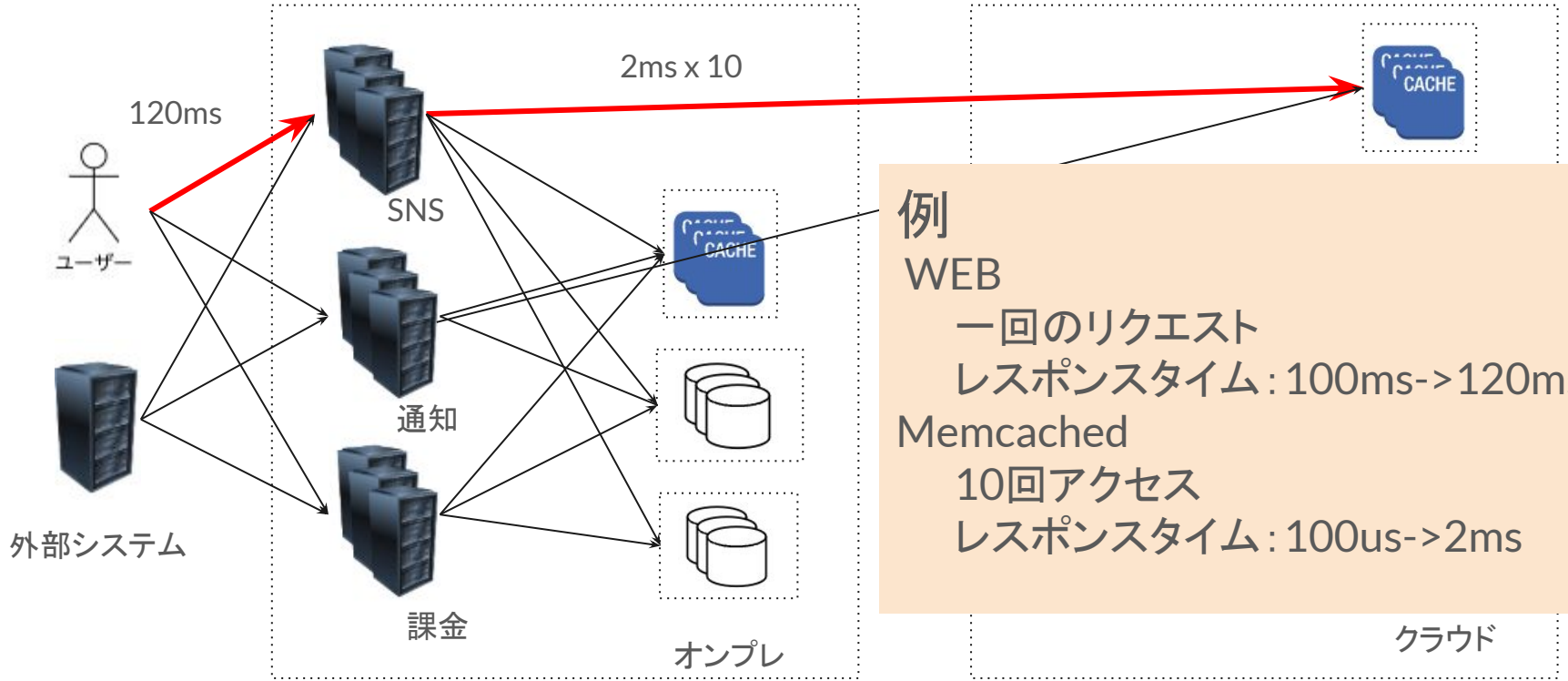
レスポンスタイム：100ms

Memcached

10回アクセス

レスポンスタイム：100us

解析対象のシステム



例

WEB

一回のリクエスト

レスポンスタイム：100ms->120ms

Memcached

10回アクセス

レスポンスタイム：100us->2ms

データセンターの移設

- やりたいこと
  - 移設におけるネットワーク遅延の影響を見積もる。
  - 移設サーバーの優先度をつける。
  - どんなリクエストで何回 Memcachedへアクセスしているか調べる。
  - どのサーバーにアクセスしているか知りたい。
  - MemcachedのIPを取得。
  - 上記の機能の bpftraceのスクリプトを作る。
- 問題
  - Apache+PHPでリクエストのたびに呼ばれる関数でメソットでパスがわかる関数がわからない。
  - PHPがMemcachedにアクセスするときの関数がわからない。
  - sockadder_inの構造体からIPを取る方法がわからない。
    - sockadder_inはCのソケット通信で使うIPを入れる構造体
    - bpftraceでlinuxのヘッダの読み込みがうまく行かない。

データセンターの移設

- 問題
  - Apache+PHPでリクエストのたびに呼ばれる関数がわからない。
  - HTTPのメソッドやパス取りたい。
- 解決方法
  - sudo bpftrace -l 'uprobe:/usr/sbin/apache2:*'でuprobeを調べる。
  - Apacheのコードリーディングをする。すでにあるものを参考に。
    - https://www.dzeta.jp/~junjis/code_reading/index.php
  - ap_process_requestを使った。

```
struct request_rec {
  int* ptrs0[6];
  char* the_request; // メソッドとパスが入る
  int dummy[4];
  char* protocol;
  char* hostname
}
uprobe:/usr/sbin/apache2:ap_process_request {
  $req=str(((struct request_rec*)arg0)->the_request);
}
```

データセンターの移設

- 問題
  - PHPがMemcachedにアクセスするときの関数がわからない。
- 解決方法
  - Memcachedにアクセスに使う共有ライブラリを調べる。
    - dpkg(パッケージツール)でphpのエクステンションを調べる。
  - Memcachedの共有ライブラリの uprobeを調べる。*をつけて検索。
    - sudo bpftrace -l ‘uprobe:xx/libmemcached.so:*’
  - 上記のうち、uprobeをつかって調べたい関数をトレース。

データセンターの移設

- 問題
  - sockadder_inの構造体からIPを取る方法がわからない。
    - sockadder_inはCのソケット通信で使うIPを入れる構造体。
    - bpftraceでlinuxのヘッダの読み込みがうまく行かない。
- 失敗した方法
  - LLMに聞く。(ハルシネーションが多くて役にたたない。)
  - linuxのヘッダーを読み込む。(bpftraceがヘッダーをパースできない。bpftraceのバグかも。)
  - ググる。
- 解決方法
  - 自前でソースを読んで sockadder_inの構造体を定義。

```
struct sockaddr_in {
  u_char  sin_len;
  u_char  sin_family;
  u_short sin_port;
  u_int32_t sin_addr;
  char    sin_zero[8];
}
```

データセンターの移設

- やりたいこと
  - ネットワーク遅延の影響を見積もる。
  - 移設の優先度をつける。
- やったこと
  - どんなリクエストで何回 Memcachedへアクセスしているか調べる。
  - データセンター間の遅延を含めて移設の影響を予測。
  - eBPF(bpftrace)で計測
    - apacheのリクエストを受ける関数を調べる。
      - コードリーディングでホスト名とクエリーパラメータが取れる関数を選定する。
      - 現在の環境で取れるかどうか確認する。
      - 今回はap_process_requestを使用した。
    - memcachedへリクエストを出す関数を調べる。
      - `sudo bpftrace -l 'uprobe:/usr/lib/x86_64-linux-gnu/libmemcached.so:*'`
    - bpftraceでスクリプトを書く。
  - 集計処理
    - LLMでpythonコードのテンプレートを生成して頑張る。
- 結果
  - あるサービスのwebのレスポンスタイム変化の予測
    i. 平均: 0.151 秒 -> 0.195秒
    ii. 最小: 0.009 秒 -> 0.014秒
    iii. 最大: 1.997 秒 -> 2.206秒

データセンターの移設：bpftraceのスクリプト

 # uprobeとuretprobeを使って処理時間を計測。

```
uprobe:/usr/sbin/apache2:ap_process_request {
  @start[pid] = nsecs;
  printf("%d 0 %s %s %s\n",pid, probe,str(((struct request_rec*)arg0)->hostname), str(((struct
request_rec*)arg0)->the_request));
}
uretprobe:/usr/sbin/apache2:ap_process_request {
  printf("%d %d %s\n",pid, nsecs-@start[pid], probe);
}

uprobe:/usr/lib/x86_64-linux-gnu/libmemcached.so:memcached_get,
uprobe:/usr/lib/x86_64-linux-gnu/libmemcached.so:memcached_set,
....
以下省略
```

Case2: ミドルウェアへのアクセスのスパイク分析

- 問題
  - ミドルウェアに特定の時間にアクセスが集中する。
  - どこのWEBサーバーからアクセスがきているのかわからない。
- やりたいこと
  - ミドルウェアへのアクセスがスパイクした時に、 IPとアクセスの種類を調べる。
  - ミドルウェアがC++で書かれていて、メンバ変数に IPアドレスが書かれているので、それを読みたい。
- 解決方法
  - C++のABIを理解して、メンバ変数にアクセスする。
    - C++のvtable、string型、name manglingの理解が必要。
    - blog書きました。https://labs.gree.jp/blog/2024/02/22855/
  - ミドルウェアの内部のメトリックス（ QPS）をみて、スパイクしているかどうか判断する。

ミドルウェアへのアクセスのスパイクの原因分析

```
uprobe:/usr/bin/flared:_ZN4gree5flare6op_get11_run_serverEv / @enb == 1  && @cnt > 0 / {
  printf("%d %s %s\n",nsecs,probe,ntop(
              ((struct op*)arg0)->connection.ptr->_addr_inet.sin_addr
              ));
  @cnt = @cnt -1;
}
uretprobe:/usr/bin/flared:_ZN4gree5flare5stats11get_cmd_getEv{
  @cmd_get_rps = (retval - @prev_cmd_get) / ((nsecs - @prev_time)/1000000000);
  @prev_cmd_get = retval;
  @prev_time = nsecs;
  if (@cmd_get_rps > 6000) {
    @enb = 1;
  } else {
    if (@cmd_get_rps < 5000) {
      @enb = 0;
      @cnt = 10;
    }
  }
}
```

Case3: セグフォの分析

- 問題
  - 本番環境でApache+PHPでセグフォが起こりコアダンプが出る。
  - コアダンプPHPのスタックトレースをみて問題の関数が見つかる。
    - しかし、その関数は単なるバッファにアクセスする関数（memcached_io_read)。
- やりたいこと
  - セグフォをしないように修正したい。
  - セグフォを起こしている関数と発生の条件を特定したい。
- 解決方法
  - eBPFで次のスクリプトを作って検査
    - memcached_io_readを呼び出している関数を列挙して uprobeで監視
    - バッファとそのポインタをダンプして、バッファの状態を監視
    - セグフォルトしたことも記録に残す
      - `tracepoint:signal:signal_deliver`を監視

セグフォの分析のコード

```
uprobe:/opt/gree/libmemcached/lib/libmemcached.so:memcached_io_read {
 $read_buffer_length = ((struct memcached_server_st*)arg0)->read_buffer_length;
 $read_data_length = ((struct memcached_server_st*)arg0)->read_data_length;
 $read_ptr = ((struct memcached_server_st*)arg0)->read_ptr;
 printf("%-7d uprobe:memcached_io_read:buf buflen:%d len:%d ptr:%p\n",pid, $read_buffer_length,
$read_data_length, $read_ptr);
}
tracepoint:signal:signal_deliver
/args->sig == 11/
{
   printf("SIGSEGV detected!\n");
   printf("# %s\n%s\n", kstack(perf), ustack(perf));
}
```

Case4: PHPのエクステンションの挙動の調査

- 問題
  - 本番環境でMemcachedへの接続の永続化ができていないような挙動が見られた。
  - しかし、コード上はちゃんと設定されているように見えるので原因が分からない。
- やりたいこと
  - 実際にいつ接続して、いつ切断がされているか調べたい。
  - ミドルウェアへのコネクションの作成と削除の関数が知りたい。
- 解決方法
  - PHPのエクステンションのコードリーディング。
  - ミドルウェアへのコネクションの作成と削除の関数を uprobeでトレース。
  - apacheのリクエスト（は ap_process_request）も監視。
- 結果
  - リクエストごとにコネクションを切っていることが確認。
  - zend_register_list_destructors_exの使い方の問題。
    - PHPのリソースのデストラクタを設定する関数。

Case5: PHPのトレース

- 問題
  - 移設中に特定のリクエストが遅い。
  - 10分に一度来るようなリクエストが対象。
- やりたいこと
  - 移設によるものか、どうなのか原因の分析をしたい。
  - 特定のリクエストだけトレースしたい。(ap_process_requestの引数のチェックで対応できる。)
  - PHPの関数のトレースがしたい。
  - PHPの変数の中身を表示したい。
- PHPのトレースをとる前提条件
  - dtraceのコードを使う。
  - 環境変数 USE_ZEND_DTRACE=1を設定する。
    - 最適化を切ることで execute_exが関数の呼び出しごとに呼ばれる。
- Apache+PHPでの問題
  - ubuntu:focalでusdtがうまく動かない。(多分bpftraceのバグ)
    - $ sudo bpftrace -l 'usdt:/usr/lib/apache2/modules/libphp7.4.so:*' | grep function__entry
    - usdt:/usr/lib/apache2/modules/libphp7.4.so:php:function__entry
    - $ sudo bpftrace -e 'usdt:/usr/lib/apache2/modules/libphp7.4.so:php:execute__return {printf ("%s",probe)}'
    - Attaching 1 probe...
    - Error finding or enabling probe: usdt:/usr/lib/apache2/modules/libphp7.4.so:php:execute__return

PHPのトレース

- 問題
  - PHPの変数の中身を表示したい。
- 解決方法
  - PHP のランタイムのデータ構造を理解して表示。
    - https://www.npopov.com/2017/04/14/PHP-7-Virtual-machine.html
    - https://www.phpinternalsbook.com/
- 重要なこと
  - zval変数
    - PHPの変数。色々な種類のデータ (数字とか文字列とか)が入っている。
    - データの種類を表す変数がある。
  - zend_string型
    - zvalが文字列型の時に文字列の値が入る型。
  - execute_ex関数
    - PHPの関数の呼び出しと対応 : 最適化されると呼ばれないことがある。
  - zend_execute_data変数
    - execute_ex関数の引数
    - PHPのスタックフレーム
    - 関数の引数が積まれている。

PHPのトレース

- 問題
  - PHPの関数のトレースがしたい。
  - bpftraceでUSDTが動かない。(Ubuntu Focal)
    - USDTにfunction__entryがある。
- 解決方法
  - USDTの代わりに、uprobeとuretprobeでexecute_exをトレース。
  - execute_exの引数のzend_execute_dataの構造体から関数名やクラス名を取り出す。
    - zend_execute_dataはPHPのスタックフレームへのポインタになっている。
  - uprobeとuretprobeの対応関係は後処理で頑張る。
    - uretprobeはどのuprobeに対応しているかわからない。
    - uretprobeでuprobeの引数にアクセスできない。
- デメリット
  - 実行負荷が大きい。
    - 1万回PHP の呼び出しがあると、uprobeでは10ms程度?のオーバーヘッドがある。
    - 負荷を軽減できない。サブサンプリングできない。
- メリット
  - dtraceでは取れなかった関数の引数の値が取れる。

PHPのトレース:コード例

```
struct zend_string {
  uint32_t refcount;
  uint32_t type;
  ulong h;
  size_t len;
  char str[64];
}
uprobe:/usr/lib/php/20190902/apcu.so:apc_cache_fetch{
  @func[pid] = ((struct zend_string*)arg1)->str;
}
uretprobe:/usr/lib/php/20190902/apcu.so:apc_cache_fetch /retval==0/{
  printf("Function returned 0, arg1: %s\n", @func[pid]);
  delete(@func[pid]);
}
```

PHPのトレース :コード例

```
uprobe:/usr/lib/apache2/modules/libphp7.4.so:execute_ex {
  $zv = (uint64) 16;
  $zed = (uint64) (8 * 8 + $zv);
  printf("%s::%s\n", ((struct zend_execute_data*)arg0)->func->scope->name->str ,
                          ((struct zend_execute_data*)arg0)->func->function_name->str);
  if (((struct zval*)(arg0+$zed))->type==6 && ((struct zend_execute_data*)arg0)->func->num_args >=1){
    printf("arg0 %s\n",  ((struct zval*)(arg0+$zed))->str->str);
  }
}
```

Case6: PHPのmysqlの呼び出しのトレース

- 問題
  - PHP からmysqlへのクエリーに時間がかかっているようだが SQLがわからない。
  - mysqli_real_queryの引数のSQLをトレースをしたいが、そのシンボルがなくて uprobeが立てられない。
- やりたいこと
  - mysqli_real_queryの引数のSQLをトレースをしたい。
- 解決方法
  - PHPのmysqlのエクステンション達
    - mysqlnd MySQL Client Library (libmysqlclient) の後継
    - mysqli
    - pdo_mysql
    - 実はどれもmysqlndのmysql_real_queryを呼び出している。
  - PHPのデバッグシンボル(`php7.4-mysql-dbgsym_7.4.3-4ubuntu2.19_amd64.ddeb`)を入れる。
  - uprobeでmysqlnd_mysqlnd_conn_data_query_pubをトレースする。
    - mysql_real_queryの関数はただのマクロで関数ポインタを追っていくと上記の関数に行き着く。

まとめと課題

- eBPFを利用しています。
  - データセンター間のネットワーク遅延の影響をアプリレベルで分析
  - PHPのパフォーマンス分析やデバッグに利用。
- 解決した課題
  - apache・HTTPの呼び出しの関数のトレース。
  - IPアドレスをとる。(`sockaddr_in`のパース)
  - セグフォを起こした時を検知する。
  - C++のトレースをとる。
  - PHPの関数呼び出しとその引数のトレースをとる。
- 課題
  - 本番環境にあまりツールを入れたくないので bpftraceを使うが、バグをふみ、騙し騙し使う。
    - 制約：OSのバージョン、サイズの制限、USDTが動かない、includeできない。
  - トレースすべきuprobeを見つけるのが大変。コードリーディングが必要。
  - LLMのサポートがいまいち。
  - uretprobeがどのuprobeに対応するかわからない。
  - uprobeのコストが高い。PHPのトレースを取るには重すぎる。
    - カーネル空間とユーザー空間を跨ぐ必要がある。(ユーザー空間の eBPF(bpftime)に期待。)
    - eBPFの良さを台無しにしている。
  - uprobeのサブサンプリングができない。(uprobeのコストの負荷の軽減ができない。)
  - セグフォを起こした時の PHPのトレースが取れない。(core dumpの分析をしたくない。軽量な方法が欲しい)

END