1J-07

準パススルー型仮想マシンモニタを用いたプロセス単位の解析インターフェース

市川遼† 並木 美太郎†
†東京農工大学

1 はじめに

ソフトウェアの挙動を解析することはパフォーマンスの向上に大きく貢献する．例えばあるソフトウェアの中で特定の関数が大量に呼び出されていた場合，その関数内の処理を最適化することで全体の実行速度の向上につながる．この関数を特定するためにはそのプロセスを監視して，全ての関数の呼び出された回数を記録することが必要になるが，解析対象となるプロセスに影響なく実行することは困難である．また，解析するためのインターフェースも OS によって異なる．

OS そのものの解析をするときはカーネルレベルの権限が必要であるが，解析のためにカーネル空間に追加した処理は当然カーネルからは無視できないため，純粋な解析結果を得ることは難しい．またカーネルはシステムに直結しているため少しでもミスがあるとクラッシュしてしまう．このため OS のデバッグを行う際は QEMU などの仮想マシンを用いるが，パフォーマンスの測定における環境としては，多くの物理デバイスが仮想化されているため不適切である．

2 課題と目標

パフォーマンス計測や挙動解析において，解析による影響を解析対象に与えないためにはそもそも解析対象と別の層で解析することが必要である．ハードウェアに解析用の機構を用意することで影響は解決できるが，物理デバイスによる解決は機器に深く依存してしまう．

また VMM からゲスト OS を解析するときには問題が生じる．通常 VMM から取得できる CPU やメモリアクセスはゲスト OS のカーネルによって発生したものであり，そこからゲスト OS のプロセスの仮想メモリなどの情報を取得することはできない．この VMM とゲスト OS 内におけるデータの意味的な隔たりをセマンティックギャップという．

そこで本研究では物理デバイスに依存せず，セマンティックギャップ問題を解決するシステムを目標とする．

3 LVisor

本研究で提案・試作するシステムを LVisor とした．LVisor の全体設計を図 3 に示す．

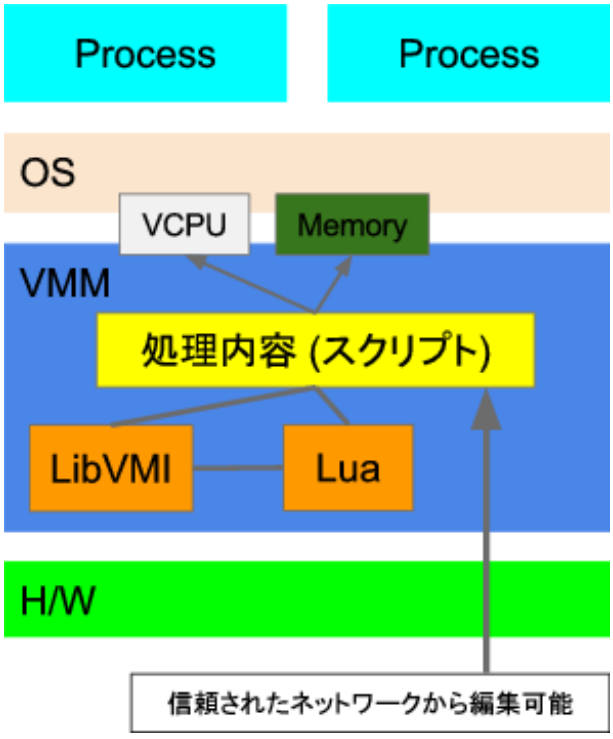

図 1: 全体構成

OS とハードウェアの間に介在する仕組みとして VMM (Virtual Machine Monitor) を用いる．VMM の層から解析を行うことでゲスト OS に影響を及ぼさず，また OS に依存しない解析を行うことができる．

Process Analysis Platform based on Para-Pass-Through Virtual Machine Monitor
Ryo ICHIKAWA†and Mitaro NAMIKI†
†Tokyo University of Agriculture and Technology
s177837r@st.go.tuat.ac.jp, namiki@cc.tuat.ac.jp

3.1 仮想環境と実機環境の差異

多くの VMM は複数の VMM を管理可能なためソフトウェアが複雑かつ巨大になりがちであるが，LVisor では一つの VM しか動作させない軽い VMM として BitVisor を用いる．BitVisor は準パススルー型 VMM であり，CPU とメモリ以外一切の仮想化を行わないことによってより実機に近い環境を実現する．

3.2 セマンティックギャップ

セマンティックギャップを解決するための機構として，Xen をベースとしたサンドボックス DRAKVUF[1] で用いられた LibVMI[2] を用いる．VMI はゲスト OS のリソースを監視・制御する技術であり，ゲスト OS 内部の情報を取得することを目的としている．

3.3 VMM の拡張性

LVisor では Lua で解析処理の内容を記述し実行時にスクリプトを評価することによって，VMM をビルドし直すことなく解析処理の内容を変更することができる．

3.4 コード例

LVisor で `mysql` のファイル書き込みを監視する例を示す．

```
1  handler = function(args)
2    proc = lvisor.getproc()
3    procname = proc.procname
4    if procname == "mysql" then
5      fd = args[0]
6      buf = args[1]
7      count = args[2]
8
9      lvisor.log("write to "..fd..":0x"
           ..string.format("%x", buf).."("
           ..count..")")
10   end
11 end
12 mvmon.hook.syscall("write", handler)
```

まずファイル書き込みが発生するシステムコール `write` が呼ばれたときの処理を記述する．`lvisor.getproc` は現在実行しているプロセスオブジェクトを返す API である．引数にはシステムコールの引数が直接渡されるため `fd,buf,count` は `args` から取得することができる．`lvisor.log` で外部のログサーバーまたは LVisor 上のログに記録する．

3.5 実装

BitVisor に Lua と LibVMI を組み込み VMM 内で Lua を動かすことに成功し，BitVisor で LibVMI を使うための LVisor API(表 1) を設計した．なお BitVisor 上で動作させる LibVMI は現在デバッグ中である．

表 1: LVisor API

関数名	機能
`lv_get_pmem`	物理メモリ読み出し
`lv_set_pmem`	物理メモリ書き込み
`lv_get_proc`	プロセス取得
`lv_get_vmem`	仮想メモリ読み出し
`lv_set_vmem`	仮想メモリ書き込み
`lv_hook`	フック生成 (メモリアクセス，システムコール)
`lv_add_hook`	フック処置追加
`lv_del_hook`	フック処理削除

4 おわりに

BitVisor と LibVMI を組み合わせることによって，OS の環境を壊すことなくリソース監視・制御が可能なシステムを提案した．このシステムを用いることでほぼ実機に近い環境で動作するアプリケーションの様々なデータを取ることができる．

また LVisor では監視だけでなく制御することもできるため，メモリの内容を書き換えたりすることが可能である．これを利用して，外側から OS の設定を書き換えたり，コマンドを実行することでサービスのデプロイが行える．LVisor は PXE boot を用いてネットワークから配信することができるため，実機においてもエージェント無しで自動展開を行うことができる．LVisor は VMM からゲスト OS 内部を覗き見ることができるシステムであるため，様々な解析が可能になる．

今後，LibVMI の移植を完成させるのが課題である．

謝辞

本研究は、科研費基盤研究 (B)17H01708 の助成を受けたものである。

参考文献

[1] Lengyel, Tamas K and Maresca, Steve and Payne, Bryan D and Webster, George D and Vogl, Sebastian and Kiayias, Aggelos: "Scalability, fidelity and stealth in the drakvuf dynamic malware analysis system", Proceedings of the 30th Annual Computer Security Applications Conference, pp. 386-395, ACM, 2014.

[2] "LibVMI", `http://libvmi.com/`