Vol.2012-OS-122 No.14
2012/8/1

VMM による軽量かつセキュアなボランティアコンピューティング基盤の実現

芹川 大地[1]　表 祐志[1]　品川 高廣[2]　加藤 和彦[1]

概要：ボランティアコンピューティングとは，企業や研究機関等が自組織で所有するコンピュータの代わりに，一般の PC ユーザがボランティアで提供する計算資源を利用して分散コンピューティングを行う仕組みである．ボランティアコンピューティングには，高価なコンピュータを自組織で用意する必要がないという利点がある一方，組織側にとっては用意した計算コードや計算結果が第三者やボランティアに改竄・盗聴される可能性があるほか，ボランティア側にとっては計算資源として提供している PC のセキュリティやプライバシーが危険にさらされる可能性がある．従来の研究では，計算コードを仮想マシン内で動かすことで安全性を向上させる試みなどがなされているが，導入に手間がかかるほかオーバーヘッドが大きくなってしまう．

本研究では，準パススルー型 VMM である BitVisor を活用することにより，一般ユーザの PC に容易に導入できて，オーバーヘッドも少ないボランティアコンピューティングの枠組みを実現するとともに，組織側とボランティア側の双方のセキュリティやプライバシーを同時に保護できる仕組みを提案する．

1. はじめに

多数のコンピュータをネットワークで繋いで分散処理させる技術の一つとして，ボランティアコンピューティング [1] がある．ボランティアコンピューティングでは，一般の PC ユーザーがボランティアとして自身の所有している計算資源（CPU やメモリ等）の余剰分を提供し，企業や研究機関がその計算資源を用いて計算処理をおこなう．そのため，企業や研究機関は高価なコンピュータを持つことなく大規模計算を行うことができるという利点がある．以下では，企業や研究機関のことを**組織**と呼び，自身の PC を提供する一般の PC ユーザーのことを**ボランティア**と呼ぶ．

ボランティアコンピューティングには，組織側及びボランティア側のそれぞれにとって問題が存在する．まず組織側の問題点としては，悪意のあるボランティアが組織側の計算コードや計算結果を漏洩・改竄してしまう恐れがある点が挙げられる．計算処理はボランティア側の PC 上でおこなわれており，通常は組織側からボランティアの PC 上の動作を制御することは出来ない．従って，組織側のノウハウが詰まった計算コードが流出したり，重要な計算結果が外部に流出してしまう危険性があるほか，そもそも計算結果が正しいことを保障することが困難である．

既存研究では，計算結果の改竄を検出するために，複数のボランティアに同じ計算を送り，返ってきた計算結果を比較することにより結果の正しさを検証する Redundant Computing[2] がある．しかしこの方法では，複数の PC で同じ計算を行うので非効率になってしまうほか，結果の採択は多数決によって決まるので必ずしも計算結果が正しいことは保証できない．計算コードや計算結果の漏洩を防止する手法としては，ボランティアからはアクセスできない closed-box VM[3] と呼ばれる VM を用いて機密性を高める方法がある [4]．しかし，この手法では既存の汎用 VMM を利用するため，ホスト OS や I/O 仮想化のための VMM のオーバーヘッドが大きくなる．

ボランティア側の問題点としては，組織が送り込んだ計算コードがボランティアのマシン環境を壊してしまう恐れがある点が挙げられる．ボランティアコンピューティングでは組織は信頼できるものとして扱うが，悪意を持っていなかったとしても，組織側の計算コードのバグによってマシン環境が壊されてしまう可能性は否定できない．Entropia Virtual Machine[5] では，仮想マシン内のサンドボックスで処理をすることにより，ボランティアのマシン環境の破壊を防ぐ事が出来るが，仮想化のための VMM のオーバーヘッドが大きくなるほか，VMM 自身をインストールするために手間がかかる．

[1] 筑波大学
University of Tsukuba
[2] 東京大学
University of Tokyo

本研究では，信頼できるハイパーバイザ型 VMM を用いて，ボランティアコンピューティングにおける組織側及びボランティア側の双方のセキュリティを実現する手法を提案する．まず，(1) 組織が用意した計算コードや計算結果の漏洩・改竄を防止するために，計算処理をボランティアの PC 上で動作する VMM の内部でおこなう．VMM は OS よりも高い特権で動作しており，ボランティアの OS 側から VMM 内の計算コードやデータにアクセスすることを防止できる．組織側は，VMM と暗号通信を介して計算コードや計算結果のやりとりをおこなうほか，Intel-TXT[6] と TPM[7] を利用して VMM 自身が改竄されていないことを確認する．これにより，Redundant Computing を行わずに正しい計算を効率よく安全におこなうことが出来る．

また，(2) ボランティア側のマシン環境が壊されてしまうことを防止するために，組織側から送られてきた計算コードは，VMM 内部の保護ドメインを用いてアクセス権が制限された環境で実行する．保護ドメイン内では，あらかじめ割り当てられたメモリ領域へのアクセスや送信元の組織とのネットワーク通信のみが可能であり，計算コードが VMM の本体にアクセスしたり，VMM の権限を用いてゲスト OS にアクセスすることを防止する．これにより，ボランティアのマシン環境を保護する事が出来る．

本研究では，VMM のオーバーヘッドや導入コストを削減するために，準パススルー型の VMM である BitVisor[8] を利用する．ゲスト OS からハードウェアへのアクセスの大部分をパススルーとしつつ，計算コードに最低限必要な CPU・メモリなどの計算資源の確保と計算コードに対する保護のみを VMM で実現することにより，VMM が介在することによるコストを削減するほか，既存のインストール済みの環境に対する導入を容易にする．

本稿の構成は以下の通りである．2 章ではボランティアコンピューティングの関連技術について述べる．第 3 章では提案システムである，ハイパーバイザ型 VMM を用いたボランティアコンピューティングの設計について説明する．第 4 章では第 3 章で述 べたシステムの具体的な実装内容について説明する．第 5 章では実装したシステムの性能評価を行う．第 6 章では現在までの本研究の結論と，今後の課題について述べる．

2. 関連研究

2.1 BOINC

BOINC[2] とは, ボランティアコンピューティングを行うためのフレームワークである．BOINC は現在，様々な組織が様々なプロジェクトに用いており，有名なものとしては SETI@home[9] や Folding@home[10] 等がある．現在，BOINC によるボランティアコンピューティングの処理能力は，5Peta FLOPS を越えるほどになっている [11].

2.1.1 Redundant Computing

1 章でも述べた通り，ボランティアコンピューティングではボランティアから返ってきた計算結果が正しいか分からないので，検証を行わなければならない．BOINC では，計算結果の検証として，Redundant Computing という手法を用いる．Redundant Computing では，複数のボランティアに同じ計算コードを投げることにより，ある計算コードについて複数の計算結果が返ってくる．そして，返ってきた複数の計算結果で多数決を行うことによって，少数側を改竄された値，または計算ミスとして取り除く．

Redundant Computing では複数のボランティアに同じ計算コードを投げるため，効率が悪い．また，多数決で正しい計算結果の採択を行うため，改竄された値を取り除きやすいが，確実に取り除けるとは保証できない．

3. 提案システムの設計

本研究では VMM を用いてボランティアコンピューティングの問題点である，(1) 組織が用意した計算コードや計算結果の改竄・流出，(2) ボランティアのマシン環境の破壊，を防ぐ．

提案システムの概要を図 1 に示す．提案システムでは，ボランティアの PC に提案システム用の VMM を導入する．ボランティアは提案システム用の VMM を導入すれば，後は VMM が自動で処理を行う．VMM は組織のサーバから計算コードを受け取ると，図に示すように保護ドメイン内で処理を開始する．この保護ドメイン内では，ボランティアのマシン環境に影響を及ぼすような機能を制限する．また，この保護ドメインと VMM 間は図中に示すように，特別なインターフェースを通さないとお互いにアクセスできない．このインターフェースでは，ボランティアコンピューティングに必要な最低限のアクセスだけを許すことにより，ボランティアのマシン環境の破壊を防ぐ．処理が終わり次第，VMM は組織のサーバに計算結果を送り，一連の処理は終了となる．

本提案システムの VMM はボランティアコンピューティングを実現する為の最低限の機能のみを実装することにより，VMM の処理によって発生する VMM のオーバーヘッドを小さくする．また，本提案システムの VMM はゲスト OS の処理にほとんど手をつけず，パススルーすることにより，ゲスト OS の修正を必要とさせない．以上により，提案システムを用いることによって，軽量かつセキュアなボランティアコンピューティング基盤の実現を行う．

3.1 計算内容の保護

この節では，組織側にとっての問題である，組織が用意した計算コードや計算結果の改竄・流出を防ぐ手法について説明する．

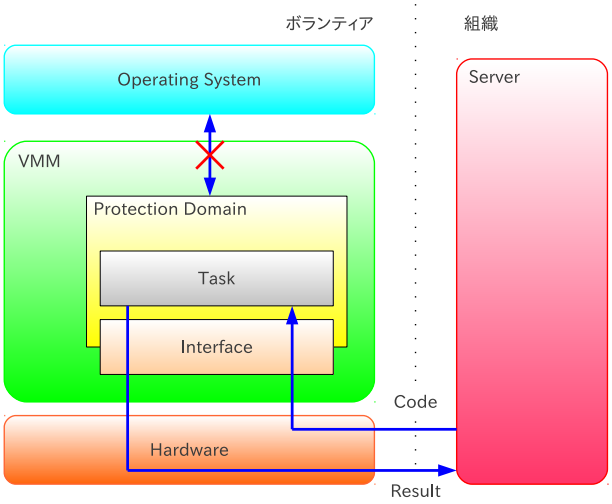

図 1 提案システムの概要
Fig. 1 Overview of the proposed system

3.1.1 VMM 内部における保護

ボランティアコンピューティングでは計算をボランティアのコンピュータで行うため，計算コードや計算結果が改竄されるのを防ぐか，改竄を検出し，改竄された計算結果を除かなくてはならない．また，組織によっては，自分たちの研究内容や計算コードのアルゴリズムを外に漏らしたくないため，自分たちの計算コードや使用されているデータを覗かれたくない組織もある．そこで提案システムでは，ボランティアが計算に一切のアクセスができないようにするために，VMM 内部で全ての処理を行う．

3.1.2 通信中における保護

ボランティアコンピューティングでは計算をボランティアのコンピュータで行うため，ボランティアのコンピュータにネットワークを介して計算コードを送り込む必要がある．そのため，通信内容を暗号化しておかなければ，第三者に盗聴・改竄を行われる可能性がある．

提案システムではボランティアによる改竄を防ぐ為に，VMM がサーバと直接通信する．VMM とサーバ間はボランティアや第三者による改竄・盗聴を防ぐために，公開鍵暗号と共通鍵暗号を用いて，セキュアな通信を行う．

3.2 マシン環境の保護

この節では，ボランティア側の問題である，ボランティアのマシン環境の保護について説明する．

3.2.1 保護ドメイン

ボランティアコンピューティングの考えでは，ボランティアは組織側のことを全面的に信頼する必要がある [2] が，その中には，組織がボランティアのコンピュータへ悪意ある計算コードを送り込まない，ということも含まれる．しかし，バグによってマシン環境が破壊されてしまう可能性は否めない．そのため，組織から送られてきた計算コードがマシン環境を破壊することを防ぐ為に，ボランティアのマシン環境に影響を及ぼすような機能を制限した環境内で計算コードを走らせ，ボランティアのマシン環境を保護する．

制限された実行環境としては，保護ドメインを用いる．メモリの一定領域を保護ドメインとして確保すると，保護ドメインと VMM 間は特定のインターフェース越しにしかアクセスができなくなる．そこで，組織から送られてきた計算コードを走らせる際，保護ドメインとしてメモリを確保する．保護ドメインと VMM の間にはボランティアコンピューティングを実現するための，最低限のインターフェースのみを提供することにより，ボランティアのマシン環境を保護する．

3.2.2 VMM とのインターフェース

3.2.1 項で述べた通り，保護ドメインと VMM の間には，ボランティアコンピューティングを実現するための最低限のインターフェースのみを提供する．提案システムでは，保護ドメインとのインターフェースとしてメッセージパッシングを用いる．VMM から特定のメッセージを保護ドメインに投げると，保護ドメイン内のプロセスはそのメッセージにしたがった挙動を取る．メッセージにはバッファを付けることができ，保護ドメイン内のプロセスとのデータのやりとりは，このバッファを用いて行う．そのため，組織側はこのメッセージパッシングを前提とした計算コードを書く必要がある．

保護ドメイン内からはメッセージに付随してくるバッファ越しにしか VMM へのアクセスを許さない．また，保護ドメインからバッファに格納するデータは計算結果のみである．VMM からはバッファにデータを格納した上で，保護ドメイン内へメッセージを投げることにより，保護ドメイン内のプロセスへデータを渡すことが可能である．

3.3 VMM の改竄検出

3.1 節では組織側の問題である計算内容の保護について，3.2 節ではボランティア側の問題であるマシン環境の保護について説明したが，両者に関わる問題として VMM の改竄が挙げられる．組織側としてはボランティアや第三者に VMM が改竄されることにより，計算コードや計算結果の改竄・盗聴が行われる危険がある．ボランティア側としては第三者に VMM が改竄されることにより，マシン環境を破壊されてしまう恐れがある．そこで，組織側がリモートで VMM の改竄検出を行う必要がある．

リモートで VMM の改竄検出を行う為に，Intel-TXT(Trusted Execution Technology) を用いる．Intel-TXT では，セキュリティチップ TPM を用いることにより，TPM を起点として対象のソフトウェアコンポーネントまでを順番に検証していく．Intel-TXT ではこの検証を動的にリモートで行うことができるため，VMM の改竄を検出することができる．

4. 実装

この章では，提案システムを実現する為に行った，実装について説明する．提案システムの VMM としては BitVisor[8] を用いる．BitVisor は単一 OS のセキュリティを強化する VMM で，セキュリティ強化のために必要な I/O のみをフックする準パススルー型アーキテクチャにより，非常に軽量な VMM となっている．

4.1 BitVisor とサーバ間のネットワーク通信

ボランティアコンピューティングでは，ボランティアのコンピュータで処理する計算コードと計算結果を，ネットワークで送受信する．計算コードの受信には，ゲスト OS と NIC 間の I/O をフックできる準パススルー型 NIC ドライバを拡張し，ボランティアコンピューティング用のパケットを選別する．計算結果の送信には，BitVisor 内部でパケットを作り，準パススルー型 NIC ドライバを通して NIC へ渡し，送信する．この節では以後，提案システムのプロトコルや通信内容の暗号化も含め，BitVisor とサーバ間のネットワーク通信について説明していく．

4.1.1 ネットワークのフック

BitVisor は準パススルー型アーキテクチャにより，特定の I/O を監視・制御できるが，ゲスト OS と NIC 間の I/O もその対象に含まれる．BitVisor は NIC からゲスト OS へパケットが流れていく過程で，準パススルー型 NIC ドライバによりパケットをフックする．本来，これはセキュリティ強化のための VPN を張るために使用されるものであるが，今回の実装ではこの機能を利用し，UDP ヘッダのポート番号で組織のサーバからのパケットを選別する．ポート番号は予め設定しておき，設定したポート番号のパケットは BitVisor 内部で処理し，それ以外のパケットはゲスト OS へ全て渡す．

4.1.2 提案システムのプロトコル

この項では，提案システムにおける現在の BitVisor とサーバ間のプロトコルについて説明する．提案システムにおける現在のプロトコルを図 2 に示す．まず，ボランティア側の BitVisor から組織側のサーバに，計算コード (バイナリコード) を要求する．要求を受け取ったサーバは計算コードを送る前に，要求を出してきた BitVisor の改竄検出を行う．BitVisor の改竄検出は 3.3 節で述べた通り，Intel-TXT を用いたリモート検証で行う．BitVisor の正当性が確認できたら，次にサーバと BitVisor は共通鍵の共有を行う．共通鍵の共有については 4.1.5 項で後述する．共通鍵の共有が行えたら，サーバ側は BitVisor にバイナリコードの受信の準備をさせるために，BitVisor に開始を伝えるためのパケット (以下，このパケットを START フラグパケットと呼ぶ) を送信する．BitVisor は START フラグパケットを受け取ると，ボランティアコンピューティングに関係する環境の初期化を行う．BitVisor は初期化を行った後，サーバに準備が出来たことを伝える．その後，サーバはバイナリコードを暗号化，分割し，分割されたバイナリコードをパケットに格納，順番に送信し続ける (以下，このパケットを CODE パケットと呼ぶ)．なお，バイナリコードの送受信中も，UDP ではパケットの到着順に差が出る可能性があるため，BitVisor はパケットを受け取ったことを毎回サーバ側に伝えることとし，パケットの損失が起きない限り，順番通りに届くようになっている．サーバは CODE パケットを全て送信し終わると，CODE パケットの終わりを伝えるパケット (以下，このパケットを END フラグパケットと呼ぶ) を送信する．BitVisor は END フラグパケットを受けとると，受け取った暗号化されたバイナリコードを復号化，処理を始める．そして処理が終了し次第，サーバに計算結果を返す．計算結果を受け取ったサーバは，BitVisor に新しく計算コードを割り当て，BitVisor に START フラグパケットを送るところから始める．また，問題が発生した際にはリセットをかけるパケット (RESET フラグパケット) を BitVisor に送信することにより，強制的に初期化させることが可能である．

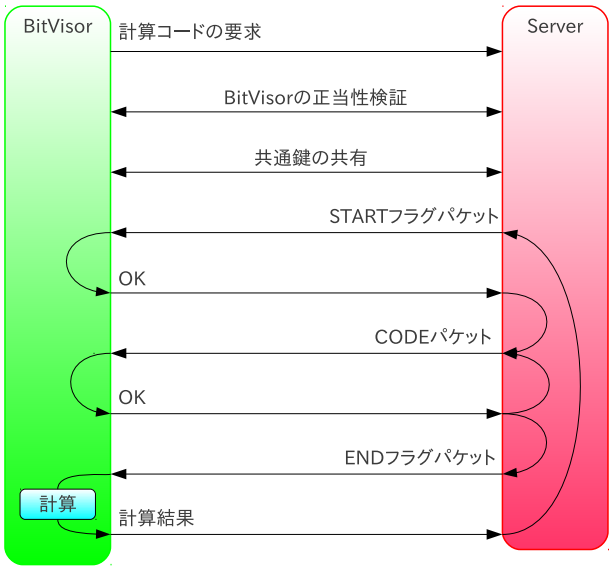

図 2 提案システムにおけるプロトコル
Fig. 2 Protocol of the proposed system

提案システムで用いるプロトコルにおける全てのパケットは，現在 UDP ペイロードに格納し，UDP でラップされている．

4.1.3 バイナリコードの受信

組織のサーバからボランティアの PC 上の BitVisor に送られる計算コードの実体は，バイナリコードである．このバイナリコードは 1 つのパケットに収めることはまず不可能であるため，サーバ側で分割，BitVisor 側で結合を行う必要がある．

まず，サーバ側は対象のバイナリコードを暗号化する．

次に暗号化されたバイナリコードをパケットに収まる大きさに分割し，ヘッダを付与する．その後，BitVisor にパケットを送信する．

START フラグパケットのヘッダにはバイナリコードのサイズが記されており，BitVisor 側ではバイナリコードのサイズ分のメモリ領域を予め確保しておく．次に，サーバから送られて来る CODE パケットを受信し，分割されたバイナリコードをメモリ上に確保された領域へ順番に格納していく．CODE パケットのヘッダには分割されたバイナリコードのサイズが記されており，BitVisor は CODE パケットを受信する間，このサイズの値を加算していく．全てのバイナリコードを受信した後，分割されたバイナリコードの合計サイズと，START フラグパケットに記されていたサイズを比較し，受信した暗号化されたバイナリコードのサイズの正当性を検証する．サイズの正当性が確認されたら，複合化し，バイナリコードの受信完了となる．

4.1.4 計算結果の送信

ボランティアの PC 上の BitVisor は，組織のサーバから送られてきた計算コードを処理した後，その計算結果をサーバに返す．BitVisor には現在，セキュリティ強化のために VPN を接続する機能と，デバッグログを NIC から出力する機能がある．そのため，BitVisor の準パススルー型 NIC ドライバには，BitVisor 内部から UDP パケットを送信するインターフェースが存在する．そこで，BitVisor の内部で計算結果送信用のパケットを作り，このインターフェースを通して，計算結果を送信する拡張を行った．

まず，UDP パケットのペイロードに計算結果を入れて UDP パケットを作成する．この UDP パケットに IP ヘッダ，Ethernet ヘッダを順番に付与する．IP アドレスは，BitVisor を組織側で配布するので，予め設定しておく．以上により，作成した Ethernet パケットを準パススルー型 NIC ドライバが NIC に渡すことにより，計算結果を送信することができる．

4.1.5 通信路の暗号化

通信路の暗号化には RSA 公開鍵暗号と AES 共通鍵暗号を用いた．RSA により AES 共通鍵を共有し，以後の通信は全て AES により暗号化される．今回の実装では，RSA と AES の利用に OpenSSL[12] を使用した．

4.2 バイナリコードの実行

ボランティアの PC 上の BitVisor が組織のサーバから受け取る計算コードの実体は，バイナリコードである．この節では，バイナリコードの実行に必要な保護ドメインと，BitVisor と保護ドメイン間のメッセージパッシングの実装について説明する．

4.2.1 保護ドメインの確保

BitVisor の保護ドメイン機能は本来，ビルド時に BitVisor に付属されるバイナリコード用の機能である．この付属されるバイナリコードは BitVisor をビルドする際に登録され，BitVisor 起動時に自動的に保護ドメインとして確保し，プロセスとして動き始める．提案システムで保護ドメインとして動かしたいバイナリコードは，BitVisor のビルド時に登録することはできない．そのため，登録されていないバイナリコードでも保護ドメインとして動かせるように拡張を行った．

4.2.2 メッセージパッシング

プロセスと BitVisor の間は，メッセージパッシングで通信する．現在の実装では

- 計算開始用のメッセージ
- 計算結果をサーバに送信する用のメッセージ

の 2 つが実装されている．

プロセスは確保された直後は計算を行わず，BitVisor からの計算開始用のメッセージを待つ．計算開始用のメッセージが BitVisor から投げられると，プロセスはボランティアコンピューティング用の計算を開始する．現在の実装では，計算を行う手法を 3 つ用意しており，それぞれにおいて計算開始用のメッセージを送るタイミングが異なる．詳しくは 5 章で後述する．計算が終了すると，BitVisor は計算結果をサーバに送信する用のメッセージを投げ，プロセスは組織側のサーバに計算結果を送信する．

4.3 プロセスの強制終了

BitVisor にはプロセスをタイムスライス等でスケジューリングする機能は存在しない．そのため，プロセスがバグ等により無限ループに入ると，ゲスト OS が停止してしまう．この節では，無限ループを検知し，プロセスを強制終了するための実装について述べる．

4.3.1 無限ループの検知

無限ループを正しく検知するのは非常に難しいため，現在の実装では一定時間計算が続くと，無限ループに入っていると判定する単純な実装を行っている．

4.3.2 IPI

IPI(Inter-Processor Interrupt) とは，プロセッサが別のプロセッサにかける割り込みのことである．IPI は割り込みコマンドレジスタと呼ばれるレジスタに書き込むことにより，発生させることができる．また，割り込みコマンドレジスタにはメモリマップド I/O を通して書き込むことができる．提案システムでは IPI を用いて無限ループに入っているプロセッサに割り込みをかけ，割り込みハンドラでプロセスの強制終了を行う．

プロセスの強制終了に用いる割り込みは，NMI(Non Maskable Interrupt) を用いる．これは BitVisor は通常，割り込みを禁止しており，普通の割り込みを用いても割り込みハンドラが呼ばれないが，NMI は割り込み禁止時にも受け付けられる割り込みであるためである．

プロセスが走っていないプロセッサが，4.3.1 項で述べた

方法で無限ループを検知すると，IPI を用いて無限ループに入っているプロセッサに NMI をかける．そして，NMI の割り込みハンドラを利用して，プロセスを強制終了する．

5. 実験

この章では，提案システムの現在までの実装における性能評価を示す．

5.1 実験環境

今回の実験に用いたマシンの環境は以下の通りである．

- Lenovo ThinkPad x61
  - OS: Ubuntu 11.04
  - CPU: Intel Core 2 Duo T7300 2.0GHz
  - Memory: PC2-5300 1GB
  - HDD: 320GB

なお，実験用マシンと，組織側のサーバを担う PC はスイッチングハブを経由して接続されている．

5.2 実験内容

今回の実験では，モンテカルロ法による円周率計算，を行う．モンテカルロ法は，二次元平面上において X 軸，Y 軸の値をランダムに取り，座標 (X,Y) で表される点が円の内側に位置する個数を試行回数で除算することにより，円周率を求めるアルゴリズムである．モンテカルロ法では，通常 X 軸，Y 軸に取る乱数は 0 以上 1 未満を取り，円の半径が 1 の単円を用いる．以後，X 軸，Y 軸の値をランダムに取り，座標 (X,Y) に点を打つことを打点と呼ぶ．

先に 4.2.2 項で触れたが，現在は計算を行う手法を 3 つ用意している．(1)1 つ目はバイナリコードを受け取った直後に計算を開始し，終了するまで BitVisor がコアを占有する手法である．この手法はコアを占有し，計算が終わるまでコアに他の処理を割り当てないため，最も早い手法となっている．但し，計算が終わるまでゲスト OS が止まってしまい，ボランティアが一切の操作をできなくなってしまう．(2)2 つ目はゲスト OS から BitVisor に遷移した際に，少しずつ計算を行う手法である．今回の実験内容では，ボランティアの操作の負担にならない程度に，一度の遷移時に連続で打点を行う．(3)3 つ目は BitVisor のタイマー機能を用い，定期的に少しずつ計算を行う手法である．但し，タイマーに設定した時間を確認するタイミングは遷移時のため，この手法も遷移に依存する．この手法でも，一度のタイマー発動時に連続で打点を行う．

5.3 実験結果

始めに，(1) の手法の結果を図 3 に示す．ここで横軸はモンテカルロ法による打った点の数，縦軸は実行にかかった秒数を表す．当然，円周率の精度は横軸右側ほど高くなる．

ここで，10 万点と 100 万点の実行時間に注目すると，それぞれ 0.01 秒と，0.04 秒であることが分かる．そこで，(2),(3) の手法において，それぞれ遷移時，タイマー発動時に 10 万点，100 万点程度の打点ではボランティアの負担にならないのはないかと予想した．そのため，(2),(3) の手法では，一度に 10 万点と 100 万点を連続で打点した際の結果を取った．また，全打点数は 10 億点で設定した．

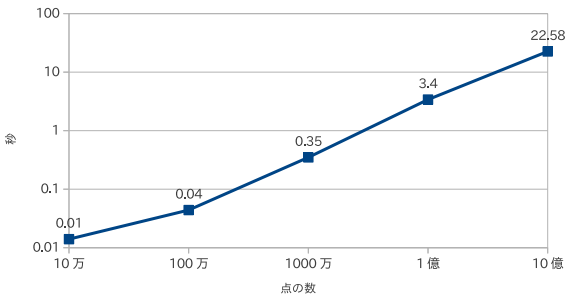

図 3 (1) の手法による実行結果
Fig. 3 Execution result by method of (1)

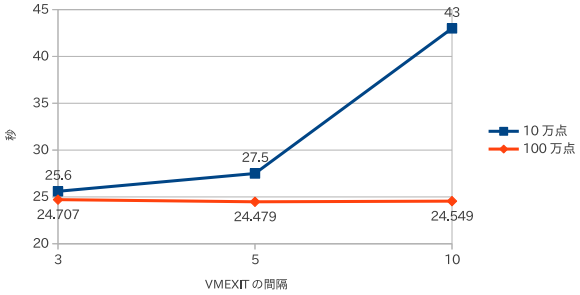

図 4 (2) の手法による実行結果
Fig. 4 Execution result by method of (2)

(2) の手法の結果を図 4 に示す．この手法では遷移時に打点を行うが，遷移 1 回毎に打点を行うと負荷が大きくなるため，遷移数回毎に打点を行う．今回の実験では遷移 3,5,10 回毎に打点を行った．遷移の間隔が図 4 中の横軸に対応する．

図中の下の線が 100 万点連続で打点を行った結果である．結果から分かる通り，遷移の間隔に依らず結果がほぼ等しい．これは打点にかかる時間に対して遷移間の時間が非常に小さい為であると思われる．

本手法においてボランティアがゲスト OS を操作できたのは，打点数 10 万点，遷移の間隔 10 回の時のみであった．他の条件では負荷が大きく，使用に耐えられるものではなかった．

(3) の手法の結果を図 5 に示す．この手法では BitVisor のタイマー機能を用いて，指定時間になると打点を行う．但し，BitVisor のタイマー機能は遷移時に確認を行うので，遷移に依存する．今回の実験ではタイマーの設定時間を 100 μ 秒，1m 秒，10m 秒に設定した．タイマーの設定

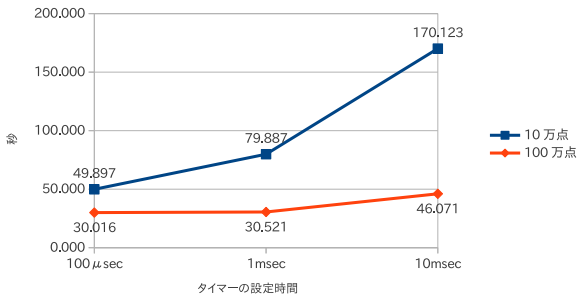

図 5 (3) の手法による実行結果
Fig. 5 Execution result by method of (3)

時間が図 5 中の横軸に対応する．

(3) も (2) と同じように，100 万点連続で打点を行った場合は，変化が小さい．これは (2) と同じように，打点にかかる時間に対してタイマー発動間の時間が非常に小さい為であると思われる．

(3) では，打点数 10 万点，タイマー 1msec と，打点数 10 万点，タイマー 10msec の時にボランティアがゲスト OS を操作できた．

5.4 考察

(1) の手法は最速ではあるものの，ボランティアが操作できなくなってしまうため手法としては採用できない．(2) の手法は遷移の間隔を完全に把握できないため，調節が難しく，現実的ではない．なお，未実装ではあるが，指定時間で強制的に遷移を起こす Preemption Timer という機能がある．Preemption Timer では最低限の性能は保証できるものの，遷移の間隔が短いときの負荷増大に対応できない．(3) の手法では，遷移に依存するものの，タイマー発動の間隔は保証できる．そのため，一見最も性能が低そうに見えるが，今回の手法の中では最も適していると言える．

但し，(2),(3) の手法は共に，ボランティアに負荷を感じさせない計算量の調節が難しく，プログラミングも非常に限定的になってしまう．現在より良い手法を思案中であるが，詳しくは 6 章で後述する．

6. 結論と今後の課題

本研究では信頼できるハイパーバイザ型 VMM を用いて，ボランティアコンピューティングにおける組織側及びボランティア側の双方のセキュリティを実現する手法を提案した．また，提案に基づき，設計と実装を行い，実際にリモートから送ったバイナリコードが実行できることを確認した．また，実行の手法についての検討，比較を行った．

今後の課題としては，パケットの欠損を防ぐ為に再送の行える TCP 通信の実装や，Intel-VT による VMM の改竄検出のリモート検証等が挙げられるが，最大の課題は計算の実行手法である．5 章では 3 つの手法を挙げたが，(1) は実行中にゲスト OS が止まってしまい，(2),(3) は実行のタイミングの調節とボランティアに負荷を感じさせない計算量の調節が難しく，組織側にも限定的なプログラミングを強いてしまう．

そこで現在は，最近の CPU のほぼ全てがマルチコアであり，コアの全てを使い切れていないことに注目し，コアの 1 つを OS から隠蔽，ボランティアコンピューティング用に割り当てることを考えている．

参考文献

[1] http://boinc.oocp.org/volunteer.php
[2] David P. Anderson. BOINC: A System for Public-Resource Computing and Storage. GRID '04 Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing
[3] Tal Garfinkel, Mendel Rosenblum, Dan Boneh. Flexible OS support and applications for Trusted Computing. : The 9th Hot Topics in Operating Systems, HOTOS-IX(2003)
[4] Wenbo Mao, Andrew Martin, Hai Jin, Huanguo Zhang. Innovations for Grid Security from Trusted Computing. Security Protocols 2006, LNCS 5087, pp. 132 149, 2009
[5] Brad Calder, Andrew A Chien, Ju Wang, DonYang. The Entropia Virtual Machine for Desktop Grids. VEE '05 Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments
[6] http://download.intel.com/technology/security/downloads/315168.pdf
[7] http://www.trustedcomputinggroup.org/resources/tpm_main_specification
[8] Takahiro Shinagawa, Hideki Eiraku, Kouichi Tanimoto, Kazumasa Omote, Shoichi Hasegawa, Takashi Horie, Manabu Hirano, Kenichi Kourai, Yoshihiro Oyama, Eiji Kawai, Kenji Kono, Shigeru Chiba, Yasushi Shinjo, Kazuhiko Kato. BitVisor: a thin hypervisor for enforcing i/o device security. VEE '09 Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments
[9] http://setiathome.berkeley.edu/
[10] http://folding.stanford.edu/
[11] http://boinc.berkeley.edu/
[12] http://www.openssl.org/