博士論文

仮想計算機における
性能プロファイリングシステムに関する研究

山本 昌生

2019年9月

岡山大学大学院
自然科学研究科

目次

図 目 次

表 目 次

第 1 章

序論

1.1 はじめに

計算機は，企業などの組織による業務用途から個人用途まで幅広く利用されている．このため，現代では，経済基盤や社会基盤として，また個人の日常生活にとっても，計算機が必要不可欠となっている．この計算機の普及の背景として，計算機の利用形態の変遷と計算機サービスの円滑な運用を支えるためのシステムの発展がある．

図 1.1 に，計算機の利用形態の変遷を示す．世界初の汎用目的で利用可能な商用計算機 UNIVAC I（Universal Automatic Computer）[1, 2] が登場した 1951 年以降，1950 年代から 1960 年代は，メインフレームと呼ばれる大型汎用計算機による集中処理が利用されていた．1970 年代から 1980 年代にかけて，計算機の小型化（ダウンサイジング）が進み，ミニコンピュータ（ミニコン）[3] やオフィスコンピュータ（オフコン）と呼ばれる計算機による集中処理の利用も広まった．1980 年代から 1990 年代にかけて，オペレーティングシステム（OS：Operating System）[4, 5] として，UNIX を使用したワークステーションや Microsoft Corporation（MS）社の Windows を使用したパーソナルコンピュータ（パソコン）と呼ばれる処理能力の高い端末の普及とともに，クライアントサーバ [6, 7] と呼ばれる分散処理 [6] の利用が主流となった．クライアントは，端末側の計算機のことで，サーバは，クライアントから依頼された処理を行い，クライアントに結果を返す計算機のことである．1990 年代後半から 2000 年代にかけて，ネットワーク [8] の高速化やインターネット [9] の普及にともない，再び集中処理の利用が広まった．このトレンドは，Web2.0 [10] とも呼ばれる．2010 年代の現在は，ネットワーク上に分散した計算機リソースを必要な時に必要な分だけ利用するクラウドコンピューティングまたはクラウドと呼ばれる集中処理の利用形態が中心となっている．

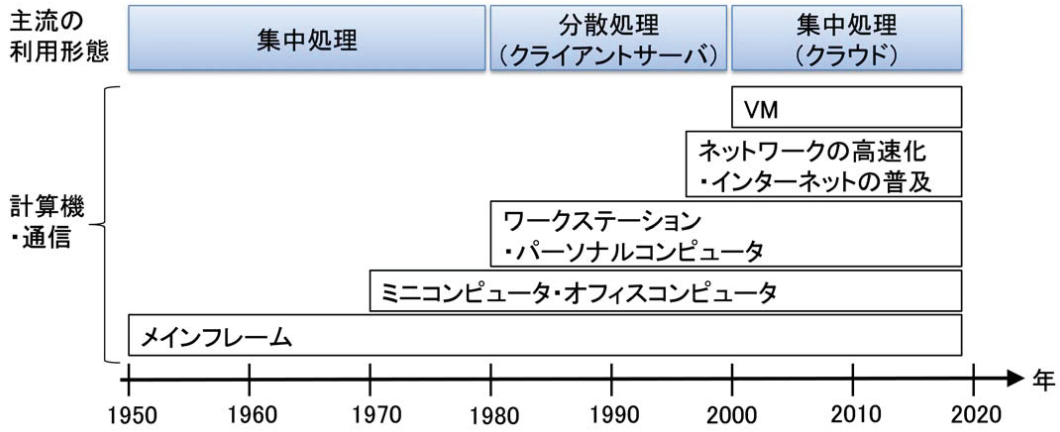

図　1.1　計算機の利用形態の変遷

クラウドとは，パソコンやスマートフォン，タブレットなどの端末から，インターネットなどのネットワークを経由して，計算機のサービスやデータを利用する利用形態のことである．クラウド利用者が利用する計算機は，クラウド上の仮想計算機（VM : Virtual Machine）[11] となる．VM は，ソフトウェア上で物理計算機を模して構築される計算機である．これにより，物理的な 1 台の計算機上に複数の計算機環境を動作させることが可能となる．

計算機サービスの円滑な運用を支えるためのシステムの一つとして，性能プロファイリングシステムがある．性能プロファイリングシステムは，計算機の性能低下異常を検出し，その要因処理を特定するシステムである．この性能プロファイリングシステムを実現するための技術も，計算機の利用形態の変遷とともに，進化する必要がある．クラウドが普及してきた現在では，クラウドが利用している VM に適した性能プロファイリングシステムが必要である．さらに，クラウドで性能異常を検出するには，VM を利用した性能プロファイリングシステムが行うデータ収集，データ格納，および解析の一連の処理を連続して継続的に実行する必要がある．

本論文では，VM 環境における継続的な性能プロファイリングシステムの実現手法の確立について述べる．先ず，既存手法の問題点と対処について述べる．次に，新たに VM 環境における継続的な性能プロファイリングシステムの実現手法について述べる．

1.2 研究背景

1.2.1 クラウド

現在，計算機サービスの利用形態としてクラウドが普及している．クラウドでは，インターネットなどのネットワーク経由で，計算機の演算能力やストレージが計算機サービスとして提供される．サービス利用者から見ると，クラウドとは，パソコンやスマートフォン，タブレットなどの端末から，インターネットなどのネットワークを介して，サービスやデータを利用する計算機の利用形態のことである．利用者からは，サービスの実体，例えば利用している表計算アプリケーションを実行している計算機や写真画像を保存しているストレージが，インターネットの向こう側のどこにあるのかが分からないため，このような形態をクラウドコンピューティングまたはクラウドと呼ぶ.

クラウドの定義

クラウドの定義としては，アメリカ国立標準技術研究所（NIST：National Institute of Standards and Technology）による定義が広く参照されている．NIST の定義は広い枠組みを示したものであり，その概要としては「クラウドコンピューティングは，共用の構成可能なコンピューティングリソース（ネットワーク，サーバー，ストレージ，アプリケーション，サービス）の集積に，どこからでも，簡便に，必要に応じて，ネットワーク経由でアクセスすることを可能とするモデルであり，最小限の利用手続きまたはサービスプロバイダとのやりとりで速やかに割当てられ提供されるものである.」 [12] とされている．つまり，ネットワーク経由で必要な時に必要なだけ計算機を利用できるサービスということになる．さらに，クラウドコンピューティングは，可用性を促進するモデルであり，五つの特徴と三つのサービスモデルと四つの配置モデルで構成されると定義されている [13].

クラウドの五つの特徴を以下に示す.

(1) On-demand self-service

(2) Broad network access

(3) Resource pooling

(4) Rapid elasticity

(5) Measured service

On-demand self-service は，必要な時に利用者自身でサービスを利用開始できることである．Broad network access は，ネットワーク経由で利用できることである．Resource pooling は，ストレージ，演算能力，メモリ，およびネットワーク帯域などのリソースを事前に用意されていることである．および複数の利用者で共有し状況に応じて使用リソース量を増減できることである．Rapid elasticity は，利用者に割り当てるリソースを利用者からの要求に応じて速やかに増減できることである．Measured service は，サービスの種類（リソース）毎に使用量が計測できることである．

クラウドの三つのサービスモデルを以下に示す．

(1) Software as a Service (SaaS)

(2) Platform as a Service (PaaS)

(3) Infrastructure as a Service (IaaS)

このサービスモデルは，五つの特徴を持ったサービスが計算機のどの範囲の機能を提供するかを定義している．図 1.2 に，三つのサービスモデルと，クラウド事業者によって提供および運用管理されるサービの範囲を示す．SaaS は，アプリケーションソフトウェアの機能をネットワーク経由で提供する形態である．PaaS は，OS やミドルウェアの機能をネットワーク経由で提供する形態である．IaaS は，物理計算機やストレージ，ネットワークなどのハードウェアリソース，および VM などの仮想化されたリソースをネットワーク経由で提供する形態である．

クラウドの四つの配置モデルを以下に示す．

(1) Private cloud（プライベートクラウド）

(2) Community cloud（コミュニティクラウド）

(3) Public cloud（パブリッククラウド）

(4) Hybrid cloud（ハイブリッドクラウド）

この配置モデルは，サービスモデルをネットワーク上のどこに構築し運用するかを定義している．プライベートクラウドは，サービスの利用者が管理しているネットワーク内に構築し

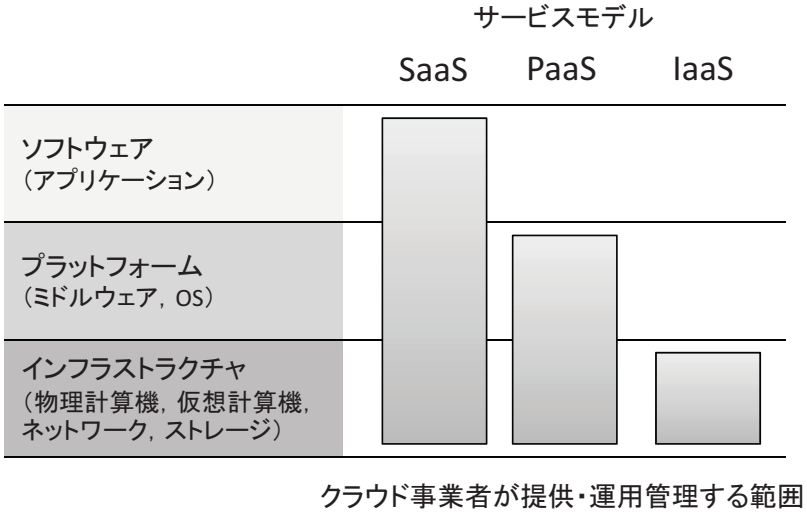

図 1.2 三つのサービスモデルとサービスの提供範囲

運用する特定の利用者専用のクラウドである．コミュニティクラウドは，特定の利用者間で管理しているネットワーク内に構築し特定の利用者間で共同運用するクラウドである．パブリッククラウドは，クラウド事業者がインターネットなどのネットワークを介して不特定多数の利用者にサービスモデルを提供するクラウドである．ハイブリッドクラウドは，プライベートクラウドとコミュニティクラウドとパブリッククラウドの中から複数のモデルを組み合わせたクラウドで，計算機リソースやアプリケーションを複数のモデル間で共有できるクラウドである．

クラウドへの今後の社会的期待

2016 年 1 月に，内閣府から，科学技術政策として第 5 期科学技術基本計画 [14] が発表され，日本が目指すべき未来社会が提唱された．提唱された未来社会で活用されるモノとして，人工知能（AI : Artificial Intelligence），ドローン，自動走行車などとともにクラウドが挙げられている．この未来社会の姿は，第 5 期科学技術基本計画の中で，Society 5.0（ソサエティ 5.0）として，「必要なもの・サービスを，必要な人に，必要な時に，必要なだけ提供し，社会のさまざまなニーズにきめ細かに対応でき，あらゆる人が質の高いサービスを受けられ，年齢，性別，地域，言語といったさまざまな違いを乗り越え，活き活きと快適に暮らすことのできる社会」と定義されている．また，内閣府では，Society 5.0 について，「サイバー空間（仮想空間）とフィジカル空間（現実空間）を高度に融合させたシステムにより，経済発展と社会的課題の解決を両立する，人間中心の社会（Society）」と述べている．さらに，「狩猟

社会（Society 1.0），農耕社会（Society 2.0），工業社会（Society 3.0），情報社会（Society 4.0）に続く，新たな社会を指すもので，第 5 期科学技術基本計画において我が国が目指すべき未来社会の姿として初めて提唱されました.」と述べている [15]．つまり，現在は，情報社会（Society 4.0）で，スマートフォンや SNS（Social Networking Service）などから発生する大量のデータ（ビッグデータ）であふれている社会である．これらのデータを活用して，経済発展と社会的課題の解決を両立する，人間中心の社会が Society 5.0 と言える．従って，Society 5.0 を実現するためには，ビッグデータを収集し分析することが必要不可となる．このためには，IoT（Internet of Things，モノのインターネット）や AI の活用が必須であり，ビッグデータと IoT と AI の連携を可能とするためにクラウドが必要とされている．これらビッグデータ，IoT，AI，クラウドを組み合わせた基盤を一般社団法人電子情報技術産業協会（JEITA）では社会 OS と呼んでいる [16]．また，実際に AI サービスを提供するクラウドとして，2018 年 8 月からサービス運用が開始された，産業技術総合研究所の AI 橋渡しクラウド（ABCI：AI Bridging Cloud Infrastructure） [17] がある．

このように，Society 5.0 を実現する仕組みとして，また社会 OS の構成要素の一つとして，クラウドは，今後ますます必要とされており，色々なハードウェアやソフトウェアやシステムのクラウド化が進む．

1.2.2 クラウドでの VM の利用

クラウドを構築し運用するために欠かせない基盤が VM である．VM は，ソフトウェア上で物理計算機を模して構築される計算機である．物理計算機を模して VM を構築し管理するソフトウェアを VM モニタ（VMM）[18] と呼ぶ．VMM には，KVM [19, 20, 21, 22]，Xen [23, 24, 25]，vSphere ESXi [26, 27, 28]，Hyper-V [29, 30, 31] などがある．VM を利用することにより，物理的な 1 台の計算機上に複数の計算機環境を動作させることが可能となる．例えば，図 1.3 に示すように，これまでは，1 台の物理計算機上に一つの OS 環境を構築していた．VM により，1 台の物理計算機上に複数の OS 環境を動作させることが可能となる．これにより，クラウドでは，高機能化で処理能力が向上したハードウェアをより効率的に動作させることができるようになった．一方で，1 台の物理計算機上に構築されるシステムの構造の複雑さが増し，計算機に異常が起こった際の要因の特定や対処が困難になった．このため，クラウドが利用している VM に適した性能プロファイリングシステムが必要である．

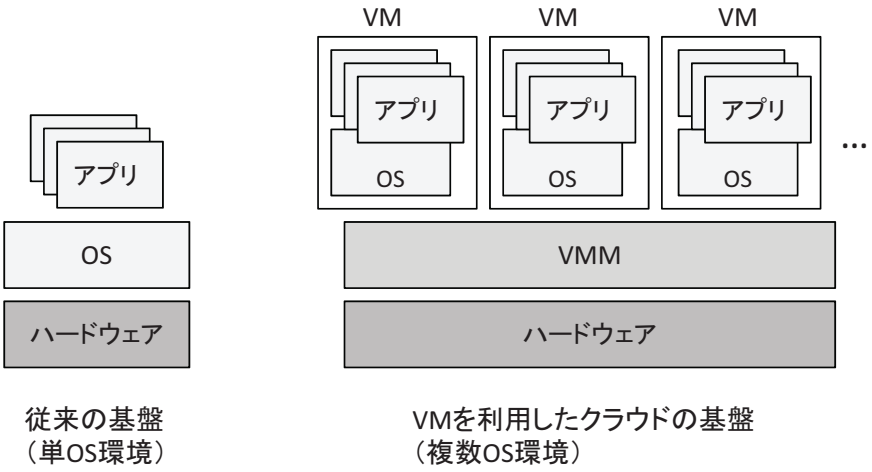

図 1.3 従来の計算機基盤と VM を利用したクラウドの計算機基盤

1.2.3 保守性向上を支える性能プロファイリングシステム

計算機の主な要件として，機能，性能，品質がある．このうち，品質要件を評価するために，RAS（ラス）という指標がよく用いられる．RAS は，1970 年に IBM 社がメインフレームの System/370 を発表した際に，評価の指標として考え，実際に商品価値の訴求のために用いたものである．RAS は，以下の三つの指標要素で構成されている．

(1) Reliability（信頼性）

(2) Availability（可用性）

(3) Serviceability（保守性）

信頼性は，システムなどの障害の起こりにくさで，障害を起こさず安定稼働し続ける平均時間（MTBF：Mean Time Between Failures）が指標値として用いられる．

可用性は，万一障害が発生しても運用中のサービスを止めずに稼働し続けられる性質で，不稼働時間や稼働率が指標値として用いられる．例えば，業務処理が途中で失敗せず完了していることを保証するトランザクション処理 [7] を行う基幹システムの中のミッションクリティカルシステムと呼ばれるシステムは，5 ナイン（99.999%）か 6 ナイン（99.9999%）の稼働率保障を必要とする．6 ナインの場合，1 年間の不稼働時間は，式 1.1 より 32 秒以内である．

$$365[\text{日}/\text{年}] \times 24[\text{時間}/\text{日}] \times 60[\text{分}/\text{時間}] \times 60[\text{秒}/\text{分}] \times (1 - 0.999999) \qquad (1.1)$$

保守性は，発生した障害の修復のしやすさで，障害発生から復旧するまでの平均時間（MTTR: Mean Time To Recovery）が指標値として用いられる．また，MTBF と MTTR を用いると，可用性の指標の稼働率は，式 1.2 で与えられる．

$$稼働率 = \frac{MTBF}{MTBF + MTTR} \tag{1.2}$$

性能プロファイリングシステムは，計算機の性能低下異常を検出しその要因となっている処理を特定することにより，MTTR を短くし，保守性の向上を支える．図 1.4 に性能プロファイリングの流れを示す．先ず，プログラムの動作情報として，命令アドレスやプロセス識別子（PID：プロセス ID）などのデータ収集を行う．データ収集契機として，CPU が備える性能監視カウンタ（PMC : Performance Monitoring Counter）のカウンタオーバフロー割込み機能を使い，データ収集契機の発生毎にプログラム動作情報をメモリに収集する．次に，データ収集終了後に，メモリからディスクにデータ格納を行う．最後に，解析処理として，ディスクに格納された収集データを基にプログラムの関数単位などで頻度集計を行う．

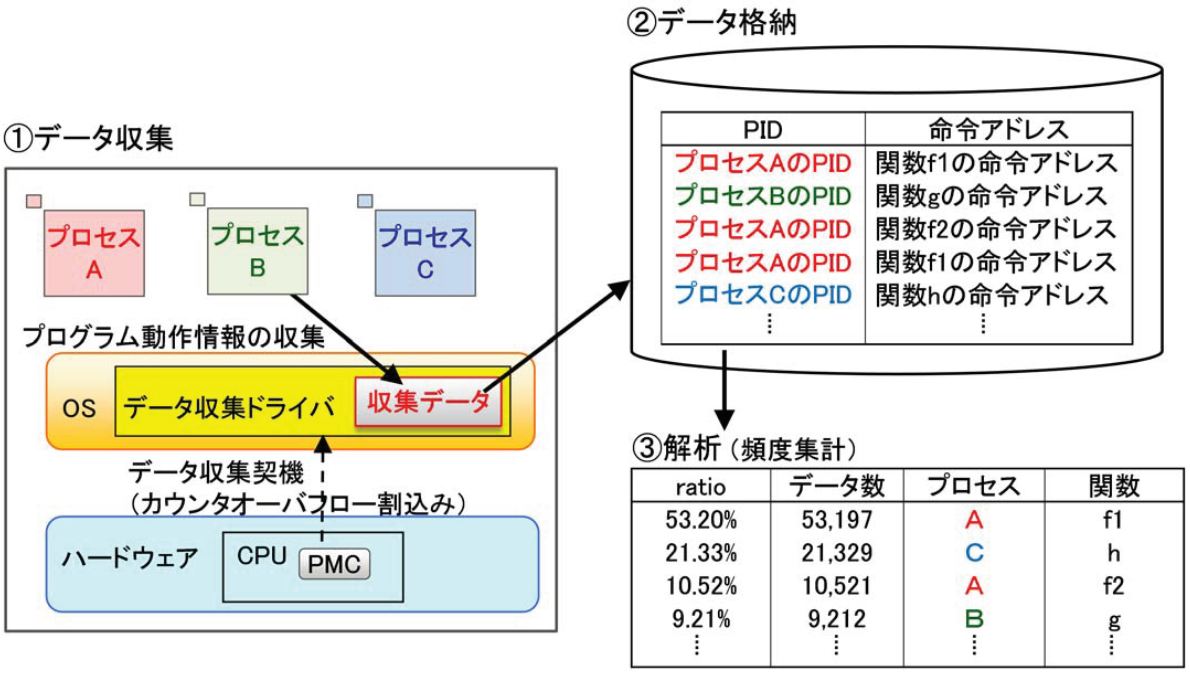

図　1.4　性能プロファイリングの流れ

1.3 性能プロファイリングシステムに関する研究状況

性能プロファイリングシステムに関する研究は，利用環境によって，

(1) 物理計算機における場合

(2) 仮想計算機における場合

に分類できる．また，性能プロファイリングシステムで行う，データ収集，データ格納，および解析の一連の処理の実行方法によって，

(a) 一度のみ実行する場合（非継続的）

(b) 繰り返し継続実行する場合（継続的）

に分類できる．そこで，本節では，これらの分類の組合せ毎に，性能プロファイリングシステムに関する研究状況について述べる．表 1.1 に，関連研究の分類を示す．

1.3.1 物理計算機における非継続的な性能プロファイリングシステム

物理計算機における非継続的な性能プロファイリングシステム（表 1.1 の (1)-(a)）に関する研究として，CPU が備える PMC ベースの性能プロファイリング手法 [49, 32, 33, 34] と PMC ベース以外の性能プロファイリング手法 [35, 36, 37, 38, 39, 40, 41] がある．また，手法に関する研究以外に，データ収集時のオーバヘッド検証 [42] がある．

PMC ベースの性能プロファイリングでは，先ず，PMC でカウントする性能イベント（例えば，CPU サイクル数や実行命令数やキャッシュミス数など）の設定を CPU に行う．次に，設定した性能イベントの一定発生回数周期でデータ収集を行う．データ収集の契機として PMC のオーバフロー割込みを利用し，オーバフロー割込み発生時に動作していたプログラムの動作情報を一定時間メモリ上に収集し続ける．プログラムの動作情報として，命令アドレスや PID などを収集する．データ収集終了後，メモリ上の収集データをディスクに格納する．解析処理では，ディスクに格納されている収集データを基に，命令ブロック単位や関数単位，プロセス単位などで頻度集計する．これにより，カウントに使用した性能イベントが，プログラム中のどの処理で多く発生したのかについて知ることができる．例えば，プログラム中のどの処理で CPU 時間の多くを消費しているか，どの処理の命令が多く実行されたか，どの処理でキャッシュミスが多く発生したかについて知ることができる．このため，PMC ベースの性能プロファイリングシステムは，計算機の性能低下異常の要因処理の調査手段として必要不可欠である．

表 1.1 利用環境，およびデータ収集，データ格納，解析の一連の処理の実行方法による関連研究の分類

	(1) 物理計算機	(2) 仮想計算機
(a) 非継続的 （一連の処理を一度のみ実行する場合）	• Oprofile [32] • Intel VTune [33] • Linux perf [34] • gprof [35] • ATOM [36] • Morph [37] • FIT [38] • Spike [39] • Etch [40] • 低負荷埋込み型 [41] • オーバヘッド検証 [42]	• Xenoprof [43] • Du らの研究 [44] • 仮想PMC方式 [45, 46, 47, 48]
(b) 継続的 （一連の処理を繰り返し実行する場合）	• DCPI [49] • GWP [50, 51]	

PMC ベース以外の性能プロファイリング手法として，データ収集コードをプログラムに埋め込む手法がある [35, 36, 37, 38, 39, 40, 41]．この手法は，性能プロファイリングによる解析対象が特定のアプリケーションや特定の OS 環境のみとなる．また，対象アプリケーションのリコンパイルが必要となる．

データ収集時のオーバヘッド検証 [42] では，Linux perf [34] を使って，PMC のカウンタオーバフロー割込みによるデータ収集時のオーバヘッドや，カウントに使用する性能イベント種を切り替えながらデータ収集する際のオーバヘッド，イベントカウントのみのオーバヘッドなどが報告されている．

1.3.2 物理計算機における継続的な性能プロファイリングシステム

物理計算機における継続的な性能プロファイリングシステム（表 1.1 の (1)-(b)）に関する研究として，プログラムの動作情報を継続的に収集する性能プロファイリングシステムの DCPI（DIGITAL Continuous Profiling Infrastructure）[49] と GWP（Google-Wide Profiling）[50, 51] がある．共通の特徴として，いずれもデータ収集を継続して行い，データベースでデータを管理し，ユーザからの要求に応じて必要な解析結果を提供するというサービスシステムとなっている．つまり，解析処理を含まない継続的な性能プロファイリングシステムである．個々の特徴として，先ず，DCPI は，1990 年代に Alpha CPU と DIGITAL Unix をベースとしたシステム向けに Digital Equipment Corporation（DEC）社が開発した性能プロファイリングシステムである．CPU の性能カウンタのオーバフロー割込みを用いたデータ収集を行う．基本仕様として 10 分周期でメモリ上の収集データをユーザが指定したディレクトリのデータベースへ格納する．データベースは，ネットワークを介して共有されうると述べられている．ベンチマーク性能の劣化による DCPI の負荷は，1〜3%で，例えば SPECint95 [52] で約 2.0%と文献 [49] で述べられている．開発とメンテナンスは，DEC 社から Compaq Computer Corporation（COMPAQ）社，Hewlett-Packard Company（HP）社と継承されたが，Alpha CPU の終焉とともに 2005 年頃を最後に今はメンテナンスされていない．なお，DCPI のサブセット機能として Oprofile [32] が派生している．DCPI は，現在多く利用されている PMC ベースのプロファイリング技術の源流といえる．Oprofile は，オープンソースで開発されているが，現在も HP 社が開発を支援している．一方，GWP は，2000 年代に Google LLC（Google）社が DCPI を参考にして，Intel CPU を用いた IA（Intel Architecture）サーバシステム向けに開発した性能プロファイリングシステムである．GWP は，Oprofile をベースに用いた自社データセンタ用の性能プロファイリングシステムで，文献 [51] では，2 万台以上の物理計算機を対象にした GWP の使用が報告されている．

1.3.3 仮想計算機における非継続的な性能プロファイリングシステム

VM における非継続的な性能プロファイリングシステム（表 1.1 の (2)-(a)）に関する研究として，デリゲーション方式 [43, 44] と仮想 PMC 方式 [45, 46, 47, 48] がある．

デリゲーション方式は，VMM と VM の両方に性能プロファイリングシステムを配置し協調動作する方式である [43, 44] ．図 1.5 に示すように，データ収集契機となる PMC のカウンタオーバーフロー割込みが発生すると，VMM 上のデータ収集ドライバが割込みを受けて，その時動作していた VM 上のデータ収集ドライバに VMM 上から仮想割込みをあげて処理

を引き渡す．処理を引き受けたVM上のデータ収集ドライバは，その時VM上で動作していたプログラムの動作情報を収集する．このため，デリゲーション方式では，データ収集毎にVMコンテキストスイッチが発生し，データ収集オーバヘッドが高くなる．また，物理CPU数が仮想CPU数より小さい場合，VMが動作している物理CPUの使用を他VMに奪われて動作できなくなる待ち時間（スチール時間）が発生する．しかし，デリゲーション方式は，VM毎にデータ収集し解析するため，スチール時間を考慮した正精度な性能プロファイリングができない．

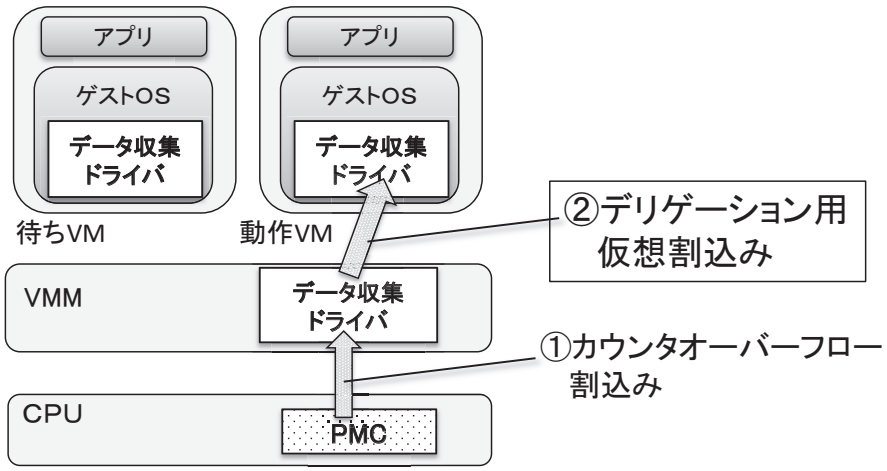

図 1.5 デリゲーション方式

仮想PMC方式は，PMCをVMMが仮想化してVM上から利用できるようにし，VM上で既存の性能プロファイリングシステムを利用可能とする方式である [45, 46, 47, 48] ．しかし，デリゲーション方式と同様に，データ収集毎にVMMとVM間で仮想割込みによるVMコンテキストスイッチが発生するため，データ収集オーバヘッドが高くなる．また，仮想PMC方式も，VM毎にデータ収集し解析するため，デリゲーション方式と同様に，物理CPU数と仮想CPU数の違いにより発生するスチール時間を考慮した高精度な性能プロファイリングができない．

1.4 本論文の研究課題

1.4.1 目的

クラウドが普及してきた現在，クラウドの保守性向上を支える性能プロファイリングシステムが必要である．しかし，クラウドで利用されている VM における性能プロファイリングシステムの既存手法には，データ収集オーバヘッドが高い問題や物理/仮想 CPU 数の違いを考慮した高精度な性能プロファイリングができない問題がある．さらに，VM における継続的な性能プロファイリングシステムの既存手法がない．そこで，本論文では，これらの問題を解消し，VM における継続的な性能プロファイリングシステムの実現手法を確立することを目的とする．

1.4.2 内容

1.4.1 項の目的を達成するために，本論文では，以下を課題とする．

（課題 1） データ収集オーバヘッドの削減

（課題 2） 物理/仮想 CPU 数の違いを考慮した測定精度の向上

（課題 3） 継続的な性能プロファイリングを可能にするシステムの分散化とデータ収集停止時間の短縮

（課題 4） 分散化した性能プロファイリングシステムの解析処理時間の短縮

（課題 1）と（課題 2）により，既存手法の問題点を解消する．さらに,（課題 3）と（課題 4）により，新たに VM 環境における継続的な性能プロファイリングの実現手法を確立する．

1.5 論文の構成

第 2 章では，VM を利用した性能プロファイリングシステムのデータ収集時のオーバヘッドを削減するための手法について述べる．

第 3 章では，物理 CPU 数と仮想 CPU 数の違いにより発生する VM のスチール時間の定義と問題を説明し，測定精度の向上手法について述べる．

第 4 章では，継続的な性能プロファイリングシステムを構成するために，VM を利用した性能プロファイリングシステムの分散化について述べる．さらに，分散化した性能プロファイリングシステムでデータ格納時に発生するデータ収集停止時間の短縮法について述べる．

第 5 章では，分散化した性能プロファイリングシステムにおいて，データ収集とデータ格納と解析の一連の処理を連続実行するためのデータ収集時間，データ収集停止時間（データ格納時間），解析処理時間の関係条件を示す．さらに，この条件を満たすために，分散化した性能プロファイリングシステムの解析処理時間の短縮法について述べる．

第 6 章では，本論文の結論について述べる．

第 2 章

データ収集オーバヘッドの削減

2.1　概要

クラウドの性能低下異常の要因処理を特定することは，非常に困難である．これは，クラウドの基盤となっている仮想計算機（VM）における性能プロファイリングシステムに未解決の問題があるためである．本章では，VM におけるデータ収集オーバヘッドを削減できる性能プロファイリングシステムについて述べる．この性能プロファイリングシステムは，仮想計算機モニタ（VMM）でのみデータ収集を行う．これにより，VM におけるデータ収集オーバヘッドを削減できる．具体的な対処として，VMM でのデータ収集手法と VM 上の動作プログラムのシンボルマップの生成手法，および VMM が持つタイムスタンプによる各 VM の動作プログラムの疑似的な収集データ生成手法について説明する．評価では，先ず，本提案手法による解析結果が正しいことを示す．次に，既存研究と比較してオーバヘッドが削減できていることを示す．さらに，本提案手法を用いて性能低下の要因処理を特定する事例を示す．

2.2　既存の性能プロファイリング手法

2.2.1　物理計算機における性能プロファイリング手法

物理計算機において，プログラムの性能最適化や性能低下問題の要因調査のために，性能プロファイリングシステムが広く使用されてきた．例えば，性能プロファイリングシステムは，指定された性能イベントが指定回数発生するたびに，その時動していたプログラムの動

作情報を収集する．次に，性能プロファイリングシステムは，収集データを解析し，様々なプログラムの実行頻度を性能イベントの発生比率によって提示する．この統計的なプロファイリング結果を使用して，プログラム中のボトルネック箇所の検出や性能低下問題の要因調査を行うことができる．このように，性能プロファイリングシステムは，プログラムの性能最適化や性能低下問題の要因調査に必要不可欠な統計解析ツールである．

図 2.1 に，性能プロファイリングの機能ブロックと機能間の処理の流れを示す．一般的に性能プロファイリングの機能処理は三つの区分に分けられる．一つ目は，データ収集部である．二つ目は，解析部である．三つ目は，解析結果の出力生成部である．データ収集部は，二つの機能ブロックで構成される．(1) プログラム動作情報を収集する機能ブロックと (2) マップ情報やオブジェクトファイルなどを収集する機能ブロックである．解析部も二つの機能ブロックで構成される．収集データに含まれるプログラムの関数名などシンボルを解決する機能ブロックと (3) シンボル毎の頻度集計を行う機能ブロックである．図 2.1 で網掛けしているこれらの機能ブロック (1) (2) (3) が本提案手法の対象部分となる．

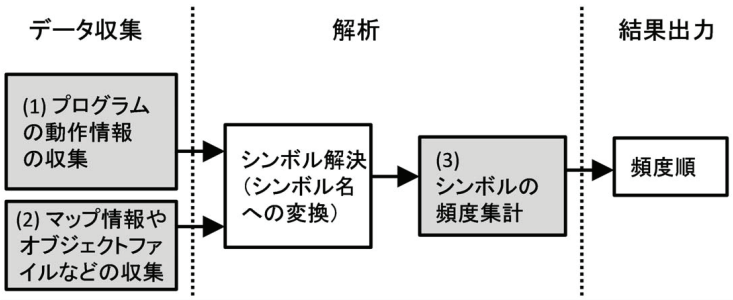

図　2.1　プロファイリングの機能ブロックと処理の流れ

データ収集ブロック (1) では，例えば，動作中のプログラムが一定時間間隔で割り込まれ，割り込まれたプログラムの動作情報を収集する．この割込みによるデータ収集契機は，ハードウェア性能監視カウンタが指定された回数のイベント発生をカウントしたらカウンタオーバフロー割込みを起こすことによって作られる．CPU サイクルイベントが一万回発生したらデータ収集割込みを生成するように設定すると，一万 CPU サイクル毎にデータ収集割込みが発生する．性能プロファイリングシステムは，割込み発生毎に，割込まれたプログラムのプロセス識別子（PID：プロセス ID）とプログラムカウンタ（PC）を動作情報として収集する．このように，性能監視カウンタのオーバフロー割込み機能を使って，PID と PC を定期的に収集する．

PID と PC からなる収集データは，人が分かるシンボルであるプロセス名と関数名に変換する必要がある．このために，プロセス情報とオブジェクトファイルを収集する必要がある．

図 2.2 に示すように，先ず PID を使用して実行プロセスを特定し，次に PC 値を使用して関数名を特定する．

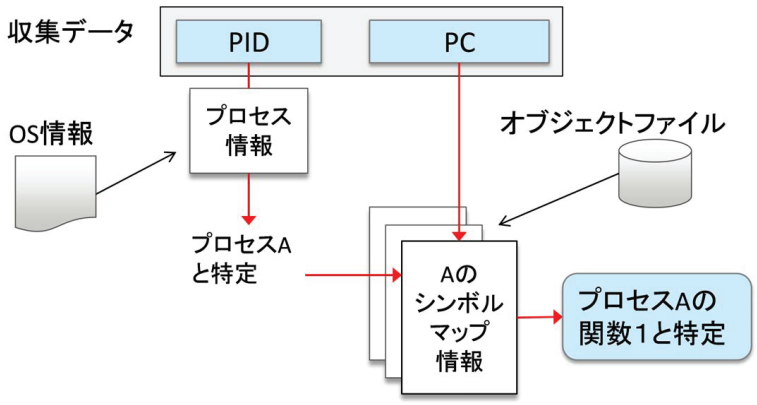

図　2.2　プロセス情報とオブジェクトファイルを使った収集データからシンボル名への変換

2.2.2　仮想計算機における性能プロファイリング手法

VM における最初の性能プロファイリングシステムである Xenoprof [43] は，2005 年に発表された．しかし，VM における性能プロファイリングシステムは，物理計算機における場合と異なり，普及していない．VM における性能プロファイリングシステムの既存の方法には，図 2.3 に示すように，VMM と VM の間の追加のコンテキストスイッチによるオーバヘッドという共通の問題がある．この追加コンテキストスイッチは，VM 上でのデータ収集のための仮想割込み注入や VM 上のプログラムのシンボル解決のために必要である．

本項では，先ず，VM 上（VM レベル）と VMM 上（VMM レベル）の両方の性能プロファイリング手法の観点から，VM における既存の性能プロファイリングシステムに関する問題を分析する．次に，性能監視カウンタベースの VM における性能プロファイリングシステムの課題として解決すべき点を説明する．

仮想性能監視カウンタを用いた VM で行う性能プロファイリング手法

ハードウェア性能監視カウンタを使う既存の性能プロファイリングシステム（例えば，Intel VTune [33]，Oprofile [32]，および PAPI [53]）が，VM においても使えるようになってきている．以前は，VMM が性能監視カウンタの仮想化には対応しておらず，VM から性能監視カウンタを使用できなかった．このため，既存の性能プロファイリングシステムは，VM 上

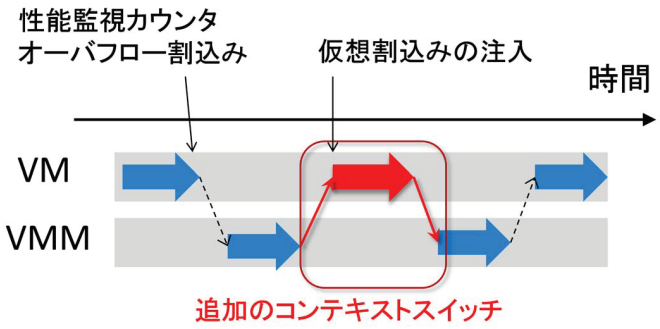

図　2.3　既存手法によるオーバヘッド

で使用できなかった．しかし，近年，一部の VMM が提供する VM は，仮想化された性能監視カウンタを備える [45, 46, 47, 48]．この仮想化した性能監視カウンタを使用することにより，既存の性能プロファイリングシステムが VM 上で使えるようになってきている．仮想化した性能監視カウンタは，VMM により，ハードウェア性能監視カウンタを VM 間でタイムシェア方式で調停されて共有されている．具体的には，VMM が，ハードウェア性能監視カウンタの VM 間のコンテキストスイッチを行い，一部のイベントのカウント値に対してはプロファイリングシステムが期待する値に一致するように調整する [45]．VMware はすでに ESXi5.1 [54] からこの機能を提供している．Xen もこの機能を提供するように拡張されている [48]．カーネルベース VM（KVM） [19] と Linux perf [34] の組合せでも同様のアプローチを採用しているが，KVM の VM では perf 以外の汎用的な性能プロファイリングシステムは利用できない [47, 55]．PAPI では，KVM の VM においても，性能プロファイリング以外では性能監視カウンタを使用することができる．KVM の VM で性能プロファイリング機能が使用できない理由は，カウンタオーバフロー割込み機能が提供されていないためである．このように，一部の VMM の VM においては，仮想化された性能監視カウンタを使用して，汎用的な性能プロファイリングシステムによる性能プロファイリングが可能になっている．

しかし，VMM システム全体の性能プロファイリングなしで，VM 上の性能プロファイリングだけでは，正確な VM 挙動を把握することは困難である．これは，VM の性能低下問題の要因が他の VM や VMM にある場合があるためである．さらに，VM 上の性能プロファイリングでは，スチール時間が把握できない．スチール時間は，物理 CPU（pCPU）を VM に割り当てることができなかったために VM が動作できなかった時間である．既存の Linux と KVM/Xen にはスチール時間のレポート機能がある．これは準仮想化機能の一種で，VMM が VM を動作スケジューリングしなかった頻度情報を VMM が提供する機能である．この機能は，例えば，KVM 環境で PAPI5 が提供する API PAPI_get_virt_usec において，スチー

ル時間の補正を実装するために使用されている．ただし，この機能はVMMシステム全体でのスチール時間を提供する．対して，PAPI5のAPIはVM上のプロセス単位の結果を返すAPIである．従って，PAPI5のAPIでは，VM上のプロセス時間の測定結果を補正することは困難である．この問題に対して，Weaverらは次の通り述べている．

「物理計算機の物理リソースを複数のVMが時分割で共有しながら使っているような状態（オーバコミット）では，スチール時間が問題になる．ほとんどのHPC（High　Performance　Computing）の場合，1台の計算機に一つのタスクしか実行させないため，これは重要な問題にはならない.」 [53]

一方，一般的なVM環境では，スチール時間は重大な問題になりうる．一般的なクラウドサービスでは，VMや仮想CPU（vCPU）が物理リソースに対してオーバーコミットされるからである．従って，本研究では，スチール時間は解決されるべき問題である．このためには，VMMで行うVMMシステム全体の性能プロファイリング手法が必要である．前述した既存の汎用性能プロファイリングシステムの中では，Oprofileだけが VMMシステム全体の性能プロファイリング手法をサポートしているが，これはXenに依存した手法で，Xenoprofと呼ばれる次に説明するVMMレベルの性能プロファイリングシステムの一つである．

デリゲーション方式による VMM で行う性能プロファイリング手法

VMMで行う性能プロファイリング手法（VMMレベルプロファイリング）は，各VMおよびVMMを統合して解析し仮想化環境全体を把握できるため，効果的な性能プロファイリング手法として期待される．実際，Xenoprof [43] やDuら [44] のVMMレベルプロファイリングの結果により，VMMで集約的に行うデータ収集とVMMから各VMへ性能プロファイリング処理を委任するデリゲーション方式がVMの性能プロファイリングシステムとして有効な手法であることが実証されている．

しかし，デリゲーション方式には問題がある．VMMで行う性能プロファイリング手法の実現の難しさは，VMで動作しているプログラムの動作情報の収集とそのシンボル解決の実現方法にあった．XenoprofとDuらは，VMMでは難しいこれらの処理をVMMの代わりにVMに処理させるデリゲーション方式で実現した．しかし，このデリゲーション方式では，性能監視カウンタのオーバフロー割込みを仮想割込みを使ってVMへ転送する必要があり，性能監視カウンタのオーバフロー割込み毎にVMMとVM間で仮想割込みによる追加のコンテキストスイッチが発生する．このため，本来の性能プロファイリング手法よりオーバヘッドが大きくなる問題がある．Duらによると，性能プロファイリング手法は性能オーバヘッドをできるだけ小さくする必要があるが，VMを利用した性能監視カウンタベースの性能プ

ロファイリングシステムでは，コンテキストスイッチの追加は不可避である [44].

対して，Linux perf [34] は，このようなデリゲーション方式を使わずに，VMM 上で vCPU のコンテキストスイッチ用のコンテキスト退避領域から VM のプログラムカウンタ（PC）と CPU 特権レベルを収集し，KVM [19] で実行されている VM 上の Linux カーネルの性能プロファイリングを行うことができる [56]．しかし，VM で動作しているプロセスの PID は収集できない．従って，VM で動作している Linux カーネルの収集データをシンボル名に変換することはできるが，VM で動作しているユーザアプリケーションや Linux 以外のゲスト OS の性能プロファイリングはできない問題がある．対して，Xenoprof [43] や Du ら [44] は，デリゲーション方式によりこの問題を解決している．つまり，ユーザーアプリケーションの分析と，VMM とゲスト間の追加のコンテキストスイッチに伴うオーバヘッドの削減との間にはトレードオフがある．

そこで，本章では，VMM システム全体の性能プロファイリングシステムにおいて，これら両方の問題を同時に解決することを目指す．

課題: VMM で行う性能プロファイリングで，追加のコンテキストスイッチは使わずに，VMM および VM で動作している全ての OS やユーザーアプリケーションプログラムの関数レベルの解析を可能生とすること．

2.3　CPU が備える仮想化支援機能を使用したデータ収集オーバヘッドの削減手法

2.3.1　手法概要

本項では，2.2.2 項で述べた課題への対処として，VMM で行う一元的な性能プロファイリングシステムを提案する．次の三つの新しい基本手法が提案手法のポイントとなる．これら三つの手法により，性能プロファイリングのオーバヘッドを増やすことなく，VMM で収集したデータからでも VM で動作するアプリケーションプログラムのシンボル名に変換する手法を確立し課題を解決する．

基本手法（1）データ収集方法：VMM で一元的に行うデータ収集
VM 上および VMM 上の動作プログラムを特定するためのプログラム動作情報を VMM だけで一元的に収集する．

基本手法（2）VM 上アプリのシンボル解決方法 : VM 上アプリのシンボルマップ情報の生成とマップ情報を用いた VM 収集データからのシンボル解決
先ず，この手法は VM 上および VMM 上で動作しているプログラムのシンボルマップ情報を生成し回収する．これらの操作は，VMM で一元的に行うデータ収集が完了した後に，各 VM 上および VMM 上で行う．その結果，データ収集中に行う VM でのデリゲーション [44, 57] は VMM システム全体での性能プロファイリングだとしても不要となる．次に，VM 上アプリのシンボルマップ情報を参照し，VM に関連する収集データをシンボル名に変換する

基本手法（3）収集データの解析方法．VMM 上の時間を用いた一つの共通ベース時間軸による解析
三つ目は，VM 上アプリも含んだ VM 環境全体の一元的な解析手法で，共通ベース時間として VMM の時間を用いる．この手法により，VM 上で実行されたアプリケーションの正確な挙動を理解できるようになる．

各基本手法の詳細は，2.3.3 項〜2.3.5 項で説明する．これらの説明では，VMM として KVM を想定した環境での一元的な性能プロファイリングシステムの実装を述べる．KVM を用いた理由はは，VMM レベルのモジュールドライバの開発が他の VMM より容易であると判断したためである．

KVM は完全仮想化を実現するソフトウェアで，Linux kernel をベースとしており，カーネルモジュールとして実装されている．従って，VMM レベルは Linux カーネルレベルと同じであり，VMM レベルのモジュールドライバは Linux カーネルモジュールとして実装すればよい．このように，データ収集ドライバの実装は KVM 固有となる．言い換えると，データ収集ドライバの実装とソフトウェア上へのロード位置は，仮想化タイプと VMM の実装に依存する．従って，基本手法（1）は VMM 依存となる．他の基本手法は VMM に非依存な手法となる．

本章で利用したハードウェア基盤は，Intel 64 Nehalem アーキテクチャである．性能監視カウンタによるカウンタオーバフロー割込みの割込みレベルはマスク不可割込み（NMI）が良い．なぜなら，データ収集契機として NMI を使えば，OS やドライバや VM エミュレータまで含めた全ソフトウェアのデータ収集が可能となるからである．データ収集時の収集データは，統合せずに全て収集形式のまま時系列順で VMM のメモリバッファに記録するものとする．

2.3.2　CPU が備える仮想化支援機能と仮想計算機管理構造体

本提案手法では，ハードウェアによるいくつかの仮想化支援機能が重要となる．本提案手法がこれらの機能に基づいているためである．本項では，Intel CPU が提供する仮想化支援機能の特徴を説明する．この仮想化支援機能は，Intel 社以外の CPU でも同じような仕組みで実装されている一般的な機能である．この CPU が備える仮想化支援機能を使って，後述の三つの新しい手法のうちの二つの手法を実装する．Intel 社は，2005 年の Intel Pentium 4 プロセッサ 662/672 から，CPU 機能として仮想化支援機能の第一世代を導入した．これは，Intel Virtualization Technology (Intel VT) [58, 59] として知られており，VT-x とも表記される．第一世代 VT-x では，命令セットを仮想化することにより CPU の仮想化がサポートされた．これにより，ソフトウェアによる CPU の仮想化（例えば，ソフトウェアによる実行命令のエミュレーションや準仮想化）が不要となった．2008 年の Nehalem プロセッサからは，第二世代 VT-x が導入され，拡張ページテーブル（EPT）と仮想プロセッサ ID（VPID）により，メモリ管理ユニット（MMU）の仮想化機能も追加された．EPT により，ソフトウェアによるページテーブルの仮想化（例えば，シャドーページテーブル）が不要となった．VM 環境において，VMM で行う一元的な性能プロファイリングシステムを実現するため，Intel VT-x のこれらのハードウェア支援機能を使用する．第一の手法の VMM 上でのデータ収集手法では，CPU の仮想化のために第一世代 VT-x で導入された VM 管理構造体を利用する．さらに，第二の手法において，VM 上のプログラムのシンボルマップを生成するために，第二世代 VT-x で導入された EPT を利用する．

ハードウェア支援による CPU 内部の新しい実行モードと VM 管理構造体（VMCS）を使ったモード遷移を図 2.4 に示す．この新しい実行モードは，VM 拡張モード（VMX）と呼ばれており，第一世代 VT-x で導入された．VMX は，二つの実行モード（VMX モード）を提供する．一つは，VMM やホスト OS が実行される root モードである．もう一つは，VM や VM 上のゲスト OS が実行される non-root モードである．各実行モードとも，四つの特権リングモード（リング 0 から 3）をサポートしている．VMM と VM はどちらも四つの特権リングモードを全て使うことができる．しかし，non-root モード時に，特権命令が実行されたり割込みが発生すると，実行モードが non-root モードから root モードに遷移する．特権命令を実行したり割込みを処理するため，VM は実行制御を VMM に渡す．この VM から VMM への制御移行は，VM-Exit と呼ばれる．特権命令や割込みの処理が完了したら，VMM は VM に制御を戻す．これを VM-Enter と呼ぶ．Intel VT は，メモリ上のデータ構造体である VMCS を使って，VM-Exit や VM-Enter の高速化を助ける．VMCS は，VM やホスト計算機の CPU のレジスタ状態（例えば，PC や制御レジスタの状態）を保持するデータ構造体である．こ

れらの状態は，VM-Exit や VM-Enter 発生時に，CPU により自動的にレジスタから VMCS へ退避，または VMCS からレジスタへ復元される．さらに，VMCS は VM の vCPU 毎に存在し，CPU ハードウェアだけでなく VMM レベルのソフトウェアからも読み書きできる．このため，VMM で行う性能プロファイリングシステムのデータ収集は，VMM レベルのドライバーとして実装し行うこととする．

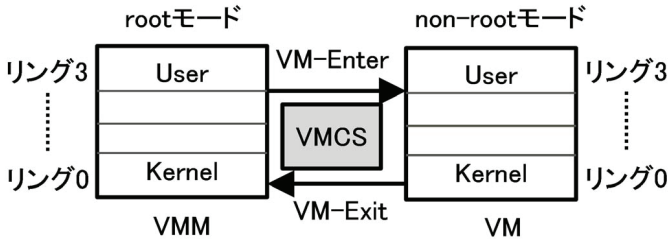

図　2.4　VMCS を使った VMX モード遷移

ハードウェアによる MMU の仮想化支援機構を図 2.5 に示す．これは，SLAT（Second Level Address Translation）と呼ばれ，拡張ページテーブル（EPT）を使って実現している．ネストティッドページテーブルとも呼ばれるこの追加ページテーブルは，第二世代 VT-x から導入された．これにより，VM 上のプロセスの場合でも，物理計算機上のプロセスの場合と同じように，ゲスト OS 自身が管理する CR3（制御レジスタ 3）とページテーブルにより，プロセスのページマッピング管理を行い続けられるようになる．CR3 は，動作中プロセスのページテーブルの先頭アドレス値を保持している．CPU からメモリへアクセスする時，MMU がページテーブルを参照して論理アドレスを物理アドレスへ変換する．物理計算機では，このアドレス変換を行うために使われるページテーブルを OS が管理している．一方，ハードウェアによる MMU の仮想化支援がない VM では，VMM が VM の物理アドレスから物理計算機の物理アドレス（マシンアドレス）へのページマッピングを VMM 内部のデータ構造を使って維持管理している．また，VMM は，シャドウページテーブルと呼ばれるページテーブルも維持管理している．これは，MMU からもアクセス可能で，VM の論理アドレスからマシンアドレスへ変換する際に参照されるページテーブルである．つまり，シャドウページテーブルは，VM の論理アドレスを物理計算機の物理アドレスに直接変換するページテーブルとなる．VMM は，シャドウページテーブルをゲスト OS が管理しているページテーブルと同期し続ける必要がある．このために，VMM は，ゲスト OS による CR3 への書き込み（ページテーブルの変更）を検知し，関連するシャドウページテーブルのアドレス値で CR3 を書き換える必要があった．対して，EPT を使ったハードウェア仮想化支援により，シャドウページテーブルが不要になる．これにより，ゲスト OS による CR3 への書き込みを検知する必要

も，VMM が CR3 を書き換える必要もなくなった．従って，性能プロファイリングシステムはゲスト OS が直接セットした CR3 値を収集することができるようになった．

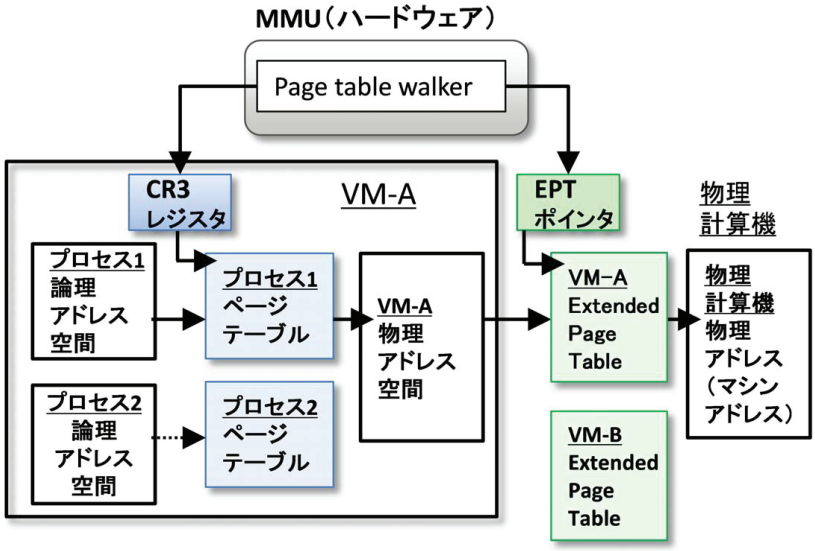

図　2.5　Intel CPU での EPT を使った SLAT

2.3.3　仮想計算機管理構造体を利用した VMM 上でのデータ収集

VMM と VM 上で動作しているプログラムの動作情報データを VMM のタイムスタンプデータとともに VMM で一元的に収集する．データ収集時の収集データは，統合せずに全て収集形式のまま時系列順で VMM のメモリバッファに，仮想 CPU（vCPU）毎ではなく物理 CPU（pCPU）毎に記録する．収集すべきデータは，以下の通りである．これらのデータをオーバフロー割込み毎に pCPU 単位で収集する．

VMM 動作情報（ホストコンテキスト）

(1) Time Stamp Counter (TSC)

(2) Process ID (PID)

(3) Program Counter (PC)

VM 上プログラム動作情報 （ゲストコンテキスト）

(4) VPID (VIRTUAL_PROCESSOR_ID)

(5) Guest PC (Program Counter)

(6) Guest CR3 (Control Register 3)

(7) VM-Exit reason number

収集すべきゲストコンテキストデータは，全て VMCS から収集できる．図 2.6 に示す通り，VMCS から Guest PC，Guest CR3，VPID，および VM-Exit reason number を収集する．

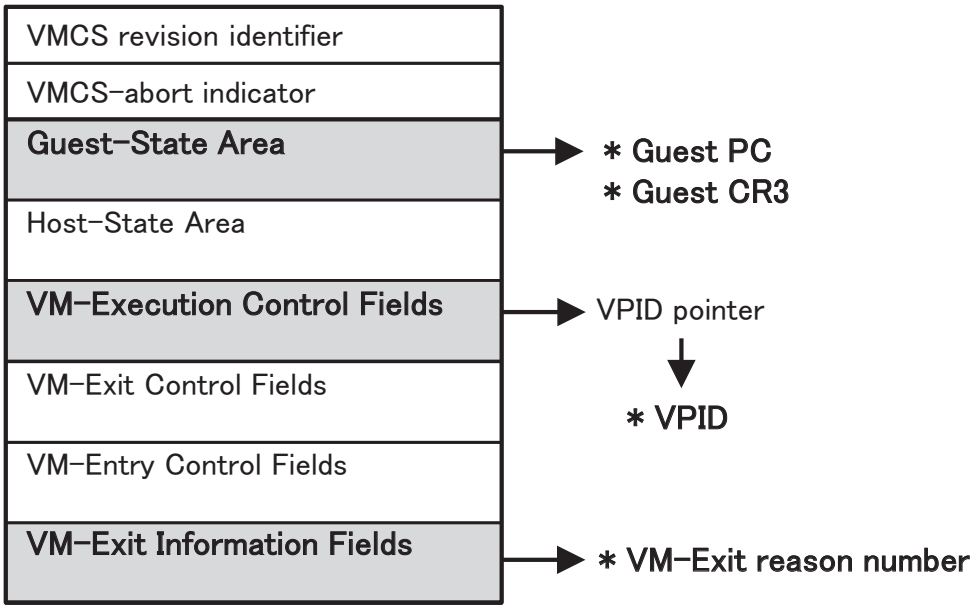

図　2.6　VMCS のフィールド内容

一元的な解析処理のための共通時間となる VMM の時間情報として，物理タイムスタンプカウンタ（TSC）を用いる．TSC を選択する理由は二つある．一つは，TSC が正確かつ高精細な時間情報源であること．もう一つは，システムに及ぼす読み出しオーバヘッドが低いことである．一方で，TSC を使用する場合，注意すべき点もある．TSC は CPU 毎に存在するが，通常では，マルチ CPU システムにおいて全 CPU の全 TSC が同じタイムスタンプ値を持っていると暗に想定して使われることが多い．TSC 間のタイムスタンプ値に差異があっても多くの場合は無視できる程度である．実際に，本評価環境において，TSC 間のタイムスタンプ値の最大差異は 1 マイクロ秒以下であった．これはデータ収集周期として使用する 1 ミリ秒に比べて無視できる．

PID 値は，メモリ上にある OS の管理情報構造体エリアから収集する．例えば，Linux では，タスク構造体として知られているエリアである．PC 値は，カウンタオーバフローによ

り割り込まれたプロセスのレジスタ状態などの実行コンテキストデータが退避されているメモリエリアから収集する．

一方，VM 上で動作しているプログラムの動作情報は，メモリ上の VMCS エリアから収集する．Guest CR3 値は，VM 上のプロセスの PID にマッピングできるため，本手法にとって特に重要な情報となる．PID 値がレジスタに設定されている情報ではないため，VM 上のプログラムの PID 値は，VMCS に退避されない．そこで，PID の代わりに，VM 上のプロセスを特定するための情報として，VMCS から Guest CR3 値を収集する．図 2.5 に示す通り，CR3 値は，プロセスのアドレス空間を特定するためのページテーブルの先頭アドレスである．従って，CR3 値は，プロセス毎に唯一の値が設定されており，PID の代わりに動作中のプロセスを示す情報として使用できる．

データ収集終了後に，VM 毎のマップ情報を生成しなければならない．特に，Guest CR3 値を PID に変換するためのマップ情報が必要である．PID 値と CR3 値は，ともに OS が管理しているので，各 VM 上でゲスト OS の管理情報構造体エリアから採取できる．また，マップ情報として，関連する PID 値と CR3 値の組合せ情報を作成する．例えば，Linux カーネルは，PID 値をタスク構造体に，CR3 値をタスク構造体からリンクされているメモリ管理構造体に，各々保持している．従って，ゲスト OS 上で全プロセスに関する PID 値と CR3 値を採取し組合せ情報を作成することができる．さらに，関数レベルのシンボル解決を行うため，カーネルのシステムマップファイルやユーザオブジェクトファイルも採取する．

2.3.4　仮想計算機上のプログラムのシンボル解決

図 2.7 の破線囲み部分が，既存処理（図 2.2）に対して本研究で新しく追加したシンボル解決手法の処理部分となる．

先ず，収集データに含まれているプログラムを特定するため，データ収集時に PID のようなプロセス識別子も同時に収集しておく必要がある．しかし，VMM レベルのデータ収集ドライバから VM 上の PID は収集できないため，VM 上のプログラの PID の代わりに，VMCS から Guest CR3 値（図 2.7 中の (i)）を収集することとした．VM 上のプログラムの PID は，VMM から直接アクセスできない．そこで，VMM から収集可能な，VM 上プログラムの PID 以外のプロセス識別子が必要となる．この目的のため，ページテーブル先頭アドレスを使用する．ページテーブル先頭アドレスは，アドレス空間 ID（ASID : Address Space IDentifier）として使われることがある [60, 61]．最近の OS では，ページテーブルはプロセス毎の専用テーブルとなる．従って，ASID は，各プロセスのための一意のグローバルアドレスを形成するために使われる．さらに，制御レジスタがページテーブルの先頭アドレスを保持する．

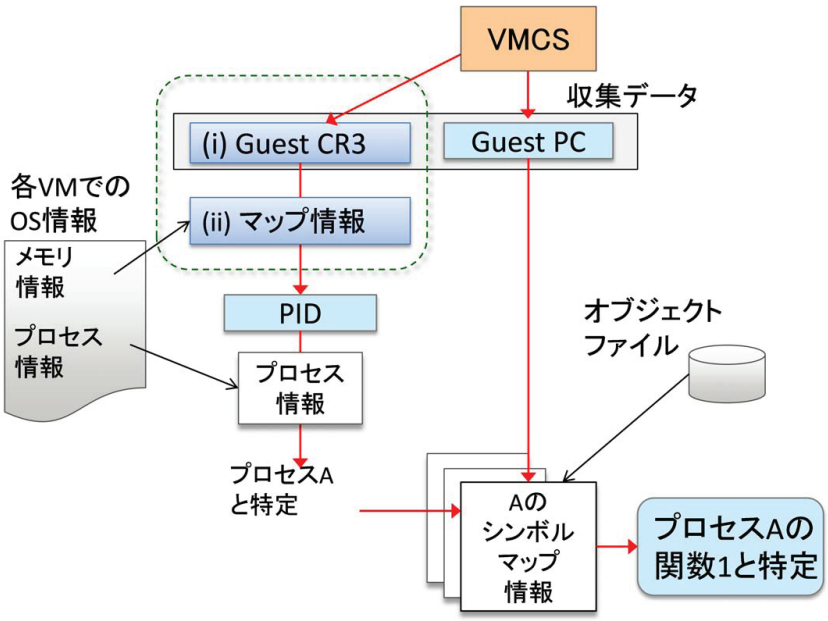

図　2.7　VM 上アプリのシンボル解決処理の流れ

Intel CPU では，CR3 レジスタが OS の制御下でページテーブル先頭アドレスを保持している．VM の各 vCPU の CR3 値は，VMCS に保存される．従って，Guest CR3 値は VMM から見え，VMM レベルの性能プロファイリングシステムは，図 2.7 に示すように，VMCS から Guest CR3 値を収集することができる．

次に，CR3 値から PID 値に変換するために参照するマップ情報が必要となる．CR3 値と PID 値は，ともに OS によって管理されている．各 VM 上で OS の管理情報構造体から CR3 値と PID 値を採取し，この対の情報をマップ情報として使用する（図 2.7 中の (ii)）．これにより，CR3 値は，CR3-PID 対のマップ情報を介して，実行されたプログラムを識別するために使用することができる．また，各 VM 上でのデータ採取は，データ収集後に 1 回だけ実行すればよい．この結果，VM 上プログラムの動作情報を収集するために，各 VM に仮想割込み注入しプロファイリング処理をデリゲーションする必要がなく，余計なオーバヘッドを発生させずにデータ収集することに成功した．さらに，OS が Linux の場合は，データ収集中に終了するプロセスのマップ情報も，Linux カーネル機能を使用してプロセス終結処理をフックすることにより，採取できる．

このように，基本手法（２）により，2.2.2 項で述べたトレードオフを解消する．これにより，クラウド向け性能プロファイリングシステムとして求められる課題を解決する．

2.3.5　仮想計算機の疑似収集データの生成

VM には，pCPU が割り当てられなかったために動作できなかった時間（空白時間）がある．空白時間帯の VM のデータ収集は，行われない．このため，空白時間は，VM に対応する有効な収集データが無い時間である．従って，頻度集計前に各 VM の空白時間を割り出し，収集データを補間する必要がある．各 VM の空白時間帯を割り出すために，VM 環境全体で共通の一つの時間軸を使用する．この共通の時間軸として，VMM の時間（物理 TSC）を使う．これにより，本提案手法（VMM で行う一元的な性能プロファイリングシステム）は，各 VM の空白時間も含めた性能プロファイリングシステムである．これは，VM 上で動作しているプログラムの正確な挙動の理解に役立つ．

VM の空白時間の割り出しは，vCPU 毎に行う．図 2.8 に，VM の空白時間を割り出し，空白時間で収集データを補間した疑似収集データの生成方法を示す．VM の性能プロファイリングは，この疑似収集データを使って行う．図 2.8 は，データ収集対象環境として，二つの pCPU があり，二つの VM（VM0 と VM1）が実行されている環境を想定した図である．VM0 には一つの vCPU があり，VM1 には二つの vCPU があるとする．vCPU は，pCPU に固定割り当てされていないとする．時間軸は，VMM の時間による経過時間とする．この図 2.8 では，t1 から t4 までの時間に収集された収集データを示す．図 2.8 の上図（a）は，VMM で収集された収集データを示す．この収集データから，VM の疑似収集データを生成する．図 2.8 の下図（b）は，生成後の各 VM の疑似収集データを示す．生成は，以下のように行う．

(1) 先ず，VMM で収集したデータを vCPU 単位でグルーピングする．

(2) 次に，グルーピングしたデータを vCPU 毎に VMM 時間軸上に並べる．

(3) 空白部分を空白時間とする．

これにより，各 vCPU の空白時間を割り出すことができる．

さらに，空白時間は，ゲスト OS がホルト命令を発行することにより発生する自発的な空白時間と，スチールによる空白時間の二つの状態に分類できる．この分類には，VM-Exit reason number を使用する．VM-Exit reason number が 12 の場合は，空白時間をホルトに分類する．12 以外の場合は，スチールに分類する．vCPU がスチール状態にある場合，VM は，実行できる状態にあるが，使用可能な pCPU が無いため，pCPU に割り当てられていない状態である．

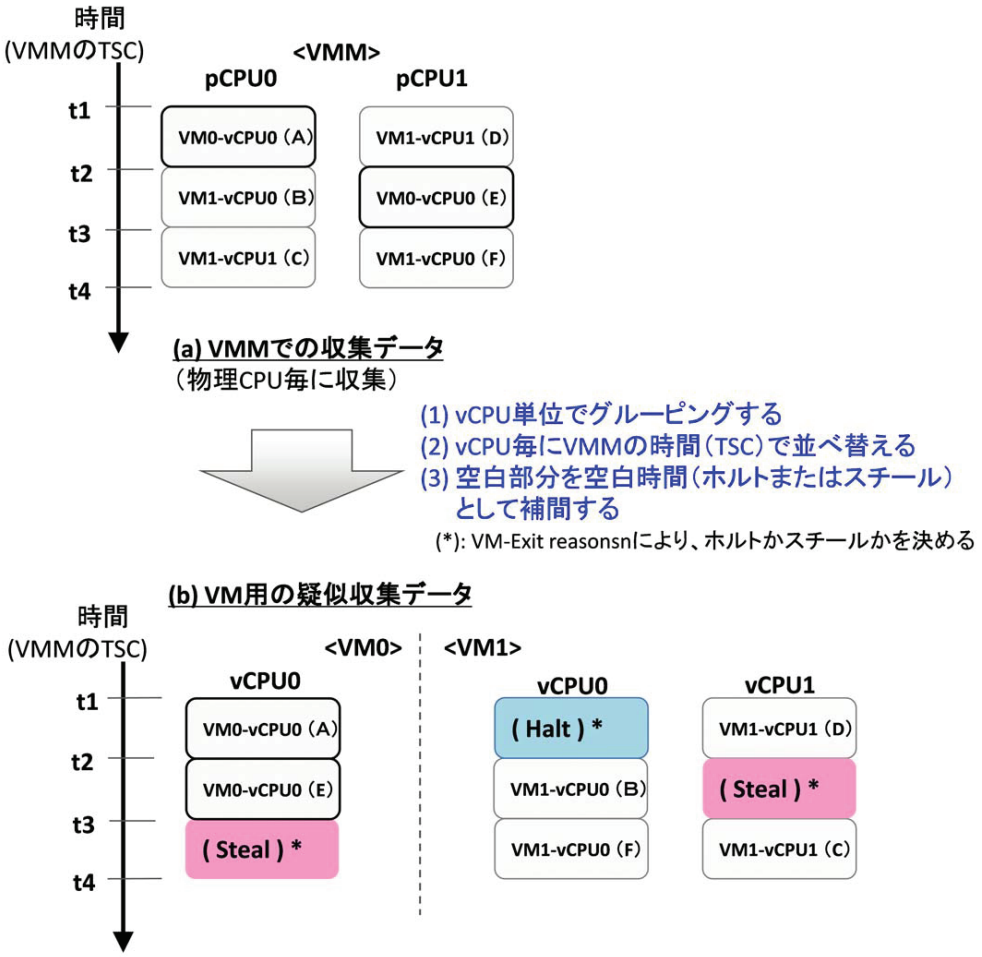

図　2.8　VM の疑似収集データの生成

2.4　評価

2.4.1　評価の目的と評価環境

VMM が KVM の環境において，VMM で行う一元的な性能プロファイリングシステムのデータ収集プログラムを Linux カーネルモジュールドライバとして実装した．KVM 環境では，カーネルレベルのプログラムが VMM レベルのプログラムとなる．OS，アプリケーション，VMM は全て変更していない．データ収集は VMM のみで行う．

2.4 節では，VMM で行う一元的な性能プロファイリングシステムの評価について述べる．先ず，VM で動作するアプリケーションも関数単位で解析できることを示す．次に，低オーバヘッドを達成することで効果を示す.（オーバヘッドを定量的に評価し，関連研究のオーバヘッドと比較する.）最後に，性能低下の要因処理を特定し改善する事例を示す.

表 2.1 に評価環境を示す．物理計算機は，Fujitsu Primergy BX900 blade server を用いる．物理環境の OS（ホスト OS）は，Linux kernel 2.6.32 を用いる．VMM は，KVM（qemu-kvm）[19, 62] を用いる．VM 数は，1VM または 3VM で評価を行う．pCPU 数は，1，2，または 4CPU で評価を行う．vCPU 数は，全ての VM で，1，2，または 4CPU で評価を行う．VM 名は，guestOS1，guestOS2，および guestOS3 とする．VM 上のゲスト OS は，ホスト OS と同じ種類かつ同じバージョンである．表 2.2 に性能プロファイリングのデータ収集条件を示す．データ収集時に動作している負荷プログラム（ワークロード）として，libquantum [63]，MySQL [64]，および Java VM （JVM）[65] を用いる．これらが性能プロファイリングの対象となる．先ず，libquantum は，基礎評価として，プロファイリング結果の正しさの評価に用いた．評価に用いた Libquantum-0.9.1 は，SPEC CPU2006 [66] に含まれるアプリケーションプログラムの一つである．このプログラムは，量子計算機のためのアルゴリズムによる量子力学シミュレーションの C ライブラリプログラムである．例えば，shor コマンドを発行すると，ショアの因数分解アルゴリズムが実行される．次に，本手法のオーバヘッドを評価するために，MySQL を用いたオンライントランザクション処理（OLTP）に対するオーバヘッドを評価する．MySQL は，Web やデータベース（DB）などのシステムを構築するために広く利用されている LAMP（Linux, Apache, MySQL, and PHP/Perl/Python）のコンポーネントの一つで，Web アプリケーションのための DB としても普及している．MySQL の OLTP 性能の測定には Sysbench [67] を用いる．最後に，VMM で行う一元的な性能プロファイリングによる性能改善事例を示すため，性能改善対象として JVM を用いる．この JVM での java トランザクション処理性能の測定には，SPECjbb2013 [68] を用いる．

評価結果を示す前に，性能プロファイリング結果を分類するための五つの用語の内容を明

表 2.1 実験環境

ホスト環境	
CPU	Intel Xeon X5570, 2.93 GHz
Num. of booted CPU	1 or 2 (§4.1 to §4.3), 4 (§4.4)
Memory	24 GB
OS	RHEL Server 6.3[1] 64 bit (kernel 2.6.32-279 x86_64)
VMM	qemu-kvm-0.12.1.2
ゲスト環境	
Num. of vCPU	1 or 2 (§4.1 to §4.3), 4 (§4.4)
Memory	4 GB (§4.1 to §4.3), 20 GB (§4.4)
OS	RHEL Server 6.3[1] 64 bit (kernel 2.6.32-279 x86_64)
Domain names	guestOS1, guestOS2, guestOS3

[1]Red Hat Enterprise Linux Server release 6.3

表 2.2 測定条件

データ収集条件	
Sampling rate	1 ミリ秒
Duration	30 秒 (§4.1 to §4.3), 60 秒 (§4.4)
Sampling base event	CPU_CLK_UNHALTED.REF_P
ワークロード 1 (§4.1 to §4.2)	
Program	libquantum 0.9.1
Compiler	gcc version 4.4.6 20120305
Optimization	-O2
Invocation	./shor 1397 8
ワークロード 2 (§4.3)	
Program	MySQL 5.1.61-4.el6 for x86_64
Benchmark	SysBench 0.4.12-5.el6 for x86_64
ワークロード 3 (§4.4)	
Program	Java HotSpot 64-Bit Server VM, version 1.7.0_79
Benchmark	SPECjbb2013-Composite:Single JVM

確にしておく．以下の五つの用語は，性能プロファイリング手法を特定するための記述である．この中で，VMM で行う一元的な性能プロファイリングシステムは，5 番目で示される性能プロファイリング手法となる．この手法は，Host-level profiling と Guest-level profiling の両方の特徴を含む手法である．

1. Native profiling: 物理計算機における既存の性能プロファイリング手法
(a) プロファイリング対象プログラムは，HOST 上の動作プログラム
(b) データ収集は，HOST 上で行う
(c) プロファイリング結果は，HOST 上の動作プログラムを示す（VM 動作は識別不可）

2. Guest-inside profiling: VM における既存の性能プロファイリング手法
(a) プロファイリング対象プログラムは，GUEST 上の動作プログラム
(b) データ収集は，GUEST 上のみで行うか，HOST と GUEST の両方で行う
(c) プロファイリング結果は，GUEST 上の動作プログラムを示す

3. Host-level profiling: Unified VM profiling の一部
（“Host-level” は “VMM-level” と同じ意味である．）
(a) プロファイリング対象プログラムと VM は，HOST 上の動作プログラム
(b) データ収集は，HOST 上で行う．
(c) プロファイリング結果は，HOST 上の動作プログラムと動作 VM（VM 名）を示す

4. Guest-level profiling: Unified VM profiling の一部
(a) プロファイリング対象プログラムは，GUEST で動作している．
(b) データ収集は，HOST 上のみで行う
(c) プロファイリング結果は，GUEST 上の動作プログラムを示す．

5. Unified VM profiling: 提案手法．VM における新たな性能プロファイリング手法で，上記 3 と 4 から構成される
(a) プロファイリング対象プログラムは，HOST 上の動作プログラムと GUEST 上の動作プログラムのどちらかで動作している．
(b) データ収集は，HOST 上のみで行う（VMM で行う）
(c) プロファイリング結果は，HOST 上の動作プログラムと GUEST 上の動作プログラム，および VM 名を示す

2.4.2　プロファイリング結果

先ず，本手法による性能プロファイリングシステムの結果の正しさを関数レベルのプロファイリング結果で検証した．このために，物理計算機と VM の両方で，プロファイリング対象として libquantum を動作させた．物理計算機の pCPU は一つである．VM は vCPU を一つ持つ VM を 1VM（guestOS1）使用した．表 2.3 に，物理計算機での関数レベルの性能プロファイリング結果を示す．表 2.4 に，本手法を使った VM の関数レベルの性能プロファイリング結果を示す．これらの比較結果を図 2.9 に示す．libquantum の関数の VM における性能プロファイリング結果と物理計算機における結果が，ほぼ同じになることを期待していた．ただし，経験上，非決定性要因（例えば，VMM の処理や他 VM による影響）のため，両者の結果にはわずかな違いがあることも予測していた．図 2.9 の libquantum の関数の結果を見ると，実験結果は期待通りである．両者の libquantum の関数の傾向はほぼ同じであるが，この場合，データ収集回数（サンプル数）は 3 %から 5 %の差を確認することができる．さらに，この差異要因として，本手法では，スチールの発生頻度を示すことができることも分かる．これは，VMM 時間を共通時間とした本提案手法の効果である．スチールは，解決すべき非決定性要因を含んでいなければならない．実際，収集データの VM-Exit reason number を調べると，スチールの要因として，外部割込み（VM-Exit-reason number 1）と VM からの割込みコントローラ（APIC）アクセス（VM-Exit-reason number 44）があることが分かった．スチールのサンプル数の内訳は，外部割込みによるスチールが 82 %，APIC アクセスによるスチールが 18 % であった．この結果から，同じ物理計算機上で他 VM が同時に存在していなくても，スチールの影響があることが分かった．このようなスチールは，既存の OS 標準ツールでは把握できない．

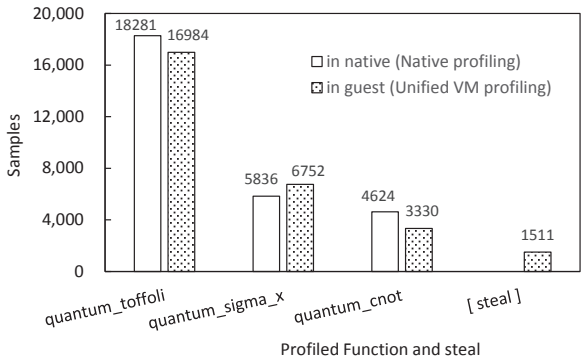

図　2.9　Native profiling 結果と Unified VM profiling 結果の比較

表 2.3 Native profiling 結果

Total samples:29938　OS:USER = 1.17%:98.83%

Samples	%Ratio	Function	Module
18281	61.06	quantum_toffoli	libquantum
5836	19.49	quantum_sigma_x	libquantum
4624	15.45	quantum_cnot	libquantum
674	2.25	quantum_swaptheleads	libquantum
171	0.57	quantum_objcode_put	libquantum
14	0.05	rb_reserve_next_event	vmlinux
12	0.04	update_wall_time	vmlinux
12	0.04	rb_end_commit	vmlinux
11	0.04	run_timer_softirq	vmlinux
10	0.03	ring_buffer_lock_reserve	vmlinux

表 2.4 Unified VM profiling 結果（Guest-level profiling 結果）

Total samples:29996　OS:USER:steal = 1.49%:93.47%:5.04%

Samples	%Ratio	Function	Module
16984	56.62	quantum_toffoli	libquantum
6752	22.51	quantum_sigma_x	libquantum
3330	11.10	quantum_cnot	libquantum
1511	5.04	[steal]	(outside)
766	2.56	quantum_swaptheleads	libquantum
198	0.66	quantum_objcode_put	libquantum
28	0.09	apic_timer_interrupt	vmlinux
22	0.07	native_apic_mem_write	vmlinux
17	0.06	__run_hrtimer	vmlinux
17	0.06	pvclock_clocksource_read	vmlinux

表 2.5 に，VMM 内の性能プロファイリング結果（Host-level profiling）を示す．この VMM 内の結果と表 2.4 に示す VM における結果（Guest-level profiling）を比較すると，guestOS1 を除いた VMM 内のサンプル数（29914-28402=1512）と VM におけるスチールのサンプル数（1511）がほぼ一致する．このように，本手法による性能プロファイリングシステムの結果が正しいことが分かる．

表 2.5 Unified VM profiling 結果（Host-level profiling 結果）

Total samples:29914　OS:USER:VM = 4.71%:0.34%:94.95%

Samples	%Ratio	Function	Module
28402	94.95	[guestOS1]	(VM1)
82	0.27	vmx_vcpu_run	kvm_intel
41	0.14	update_curr	vmlinux
34	0.11	copy_user_generic_string	vmlinux
32	0.11	kvm_arch_vcpu_ioctl_run	kvm
32	0.11	rb_reserve_next_event	vmlinux
31	0.10	(stripped local_functions)	qemu-kvm
29	0.10	ring_buffer_lock_reserve	vmlinux
24	0.08	update_wall_time	vmlinux
24	0.08	x86_decode_insn	kvm

次に，物理計算機と VM の両方で，libquantum を二つの異なるプロセス（process 1 と process 2）として実行した．表 2.6 に，物理計算機における，これら二つのプロセスの性能プロファイリング（Native profiling）結果を示す．また，図 2.10 に，物理計算機における，これら異なるプロセスの同じ関数の性能プロファイリング結果の比較を示す．対して，表 2.7 に，VM における，性能プロファイリング（Guest-level profiling）結果を示す．また，図 2.11 に，VM における，これら異なるプロセスの同じ関数の性能プロファイリング結果の比較を示す．先ず，物理計算機における性能プロファイリング結果について，図 2.10 から，各プロセスの同じ関数の CPU 使用率（%Ratio）は，ほぼ同じであることが分かる．このデータ収集時に，top コマンドで各プロセスの CPU 使用率を計測していた結果でも，49.9 % : 49.9 %で一定で，ほぼ同じであった．VM においても，二つのプロセスが同じ CPU 使用率なら，二つのプロセスの同じ関数はほぼ同じサンプル数と同じ CPU 使用率であることが期待できる．実際，VM においても，二つのプロセスの CPU 使用率は，top コマンドで 49.9 % : 49.9 %で一定で，ほぼ同じであった．対して，図 2.11 から，VM における性能プロファイリングも期待通りであることが確認できる．従って，VMM で行う一元的な性能プロファイリングシステムは，VM 上のユーザプロセスを正確に区別できることが分かる．

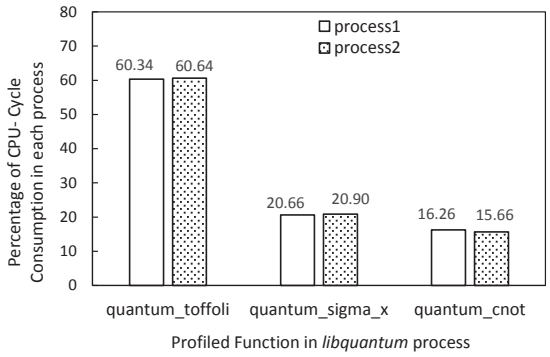

図 2.10 物理計算機での二つの異なる libquantum プロセスの同じ関数のプロファイリング結果の比較

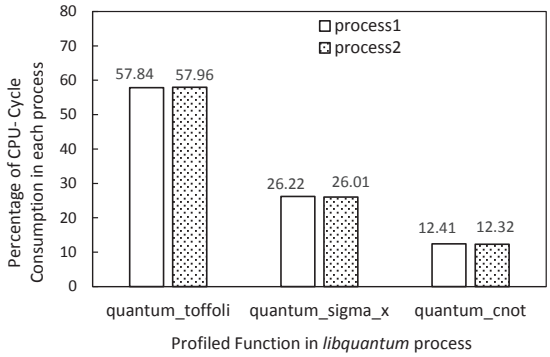

図 2.11 VM での二つの異なる libquantum プロセスの同じ関数のプロファイリング結果の比較

表 2.6 物理計算機上で二つの同じワークロード実行時の Native profiling 結果

Total samples:29938 OS:USER = 1.27%:98.73%

Samples	%Ratio	Function	Module
8963	29.94	quantum_toffoli	libquantum(2)
8915	29.78	quantum_toffoli	libquantum(1)
3089	10.32	quantum_sigma_x	libquantum(2)
3052	10.19	quantum_sigma_x	libquantum(1)
2402	8.02	quantum_cnot	libquantum(1)
2314	7.73	quantum_cnot	libquantum(2)
324	1.08	quantum_swaptheleads	libquantum(2)
323	1.08	quantum_swaptheleads	libquantum(1)
91	0.30	quantum_objcode_put	libquantum(2)
78	0.26	quantum_objcode_put	libquantum(1)
15	0.05	__rb_reserve_next	vmlinux
12	0.04	ring_buffer_lock_reserve	vmlinux

表 2.7 VM 上で二つの同じワークロード実行時の Unified VM profiling 結果

Total samples:30000 OS:USER:steal = 1.75%:93.27%:4.98%

Samples	%Ratio	Function	Module
8103	27.01	quantum_toffoli	libquantum(1)
8094	26.98	quantum_toffoli	libquantum(2)
3673	12.24	quantum_sigma_x	libquantum(1)
3633	12.11	quantum_sigma_x	libquantum(2)
1739	5.80	quantum_cnot	libquantum(1)
1721	5.74	quantum_cnot	libquantum(2)
1494	4.98	[steal]	(outside)
392	1.31	quantum_swaptheleads	libquantum(1)
384	1.28	quantum_swaptheleads	libquantum(2)
128	0.43	quantum_objcode_put	libquantum(2)
98	0.33	quantum_objcode_put	libquantum(1)
37	0.12	apic_timer_interrupt	vmlinux

さらに，2vCPU の VM において，各 vCPU に一つの libquantum プロセスを固定して実行した．この時に VMM で行った性能プロファイリング結果を表 2.8 と表 2.9 に示す．表 2.8 は，vCPU0 上のみにおけるプロファイリング解析結果である．表 2.9 は，vCPU1 上のみにおけるプロファイリング解析結果である．libquantum の関数のプロファイリング結果が，図 2.11 と似たような傾向になることが期待される．実際，図 2.12 から，期待通りであることが確認できる．また，図 2.13 に示すように，1vCPU の場合と 2vCPU の場合のサンプル数は，ほぼ一致している．このように，VMM で行う一元的な性能プロファイリングシステムは，一つの pCPU に複数の vCPU が割り当てられ動作している環境でも良好な結果を得ることができる．

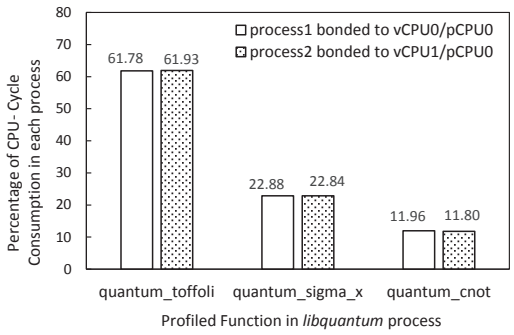

図 2.12 各 vCPU 上の二つの異なる libquantum プロセスの同じ関数のプロファイリング結果の比較

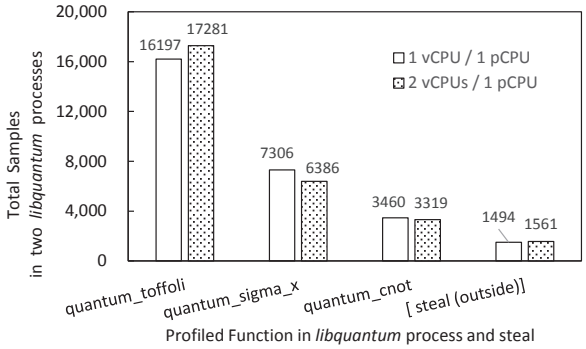

図 2.13 1vCPU と 2vCPU の場合の libquantum プロセスの同じ関数のサンプル数の比較

表 2.8 1pCPU 上で 2vCPU 持った VM における vCPU0 上の Unified VM profiling 結果

Total samples:30000 OS:USER:steal = 0.75%:46.77%:52.48%

Samples	%Ratio	Function	Module
14994	49.98	[steal]	(on vcpu1)
8669	28.90	quantum_toffoli	libquantum(1)
3210	10.70	quantum_sigma_x	libquantum(1)
1678	5.59	quantum_cnot	libquantum(1)
750	2.50	[steal]	(outside)
345	1.15	quantum_swaptheleads	libquantum(1)
122	0.41	quantum_objcode_put	libquantum(1)
26	0.09	apic_timer_interrupt	vmlinux
18	0.06	pvclock_clocksource_read	vmlinux
8	0.03	_spin_lock	vmlinux

表 2.9 1pCPU 上で 2vCPU 持った VM における vCPU1 上の Unified VM profiling 結果

Total samples:30000 OS:USER:steal = 0.90%:46.37%:52.72%

Samples	%Ratio	Function	Module
15006	50.02	[steal]	(on vcpu0)
8612	28.71	quantum_toffoli	libquantum(2)
3176	10.59	quantum_sigma_x	libquantum(2)
1641	5.47	quantum_cnot	libquantum(2)
811	2.70	[steal]	(outside)
353	1.18	quantum_swaptheleads	libquantum(2)
121	0.40	quantum_objcode_put	libquantum(2)
38	0.13	apic_timer_interrupt	vmlinux
28	0.09	pvclock_clocksource_read	vmlinux
14	0.05	native_apic_mem_write	vmlinux

最後に，2pCPU の物理計算機において，2vCPU の VM を一台用意し起動した．vCPU は，いずれかの pCPU に固定されている．VM 上では，libquantum を二つ実行した．各 libquantum プロセスは，2vCPU あるうちの指定されたどちらかの vCPU に固定割り当てされ動作する．表 2.10 と表 2.11 は，これら二つのプロセスを含む，VMM で行う一元的な性能プロファイリング結果（Guest-level profiling）である．表 2.10 は，pCPU0 に固定割り当てされた vCPU0 上のみにおけるプロファイリング解析結果である．表 2.11 は，pCPU1 に固定割り当てされた vCPU1 上のみにおけるプロファイリング解析結果である．libquantum の関数のプロファイリング結果が，図 2.12 と似たような結果になることが期待される．実際，図 2.14 から，期待通りであることが確認できる．また，pCPU あたりのサンプル数はこれまでの結果と同じであることも期待される．図 2.15 から，実験結果は期待通りであることが分かる．このように，VMM で行う一元的な性能プロファイリングシステムは，複数の pCPU 環境でもうまく動作する．

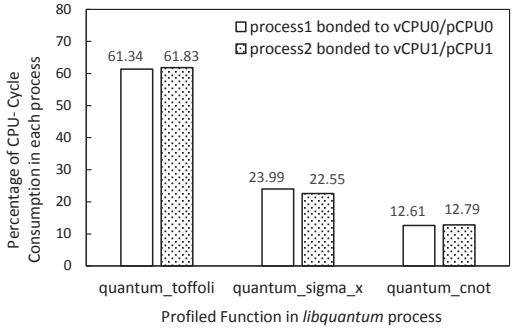

図 2.14 各 vCPU 上の二つの異なる libquantum プロセスの同じ関数のプロファイリング結果の比較

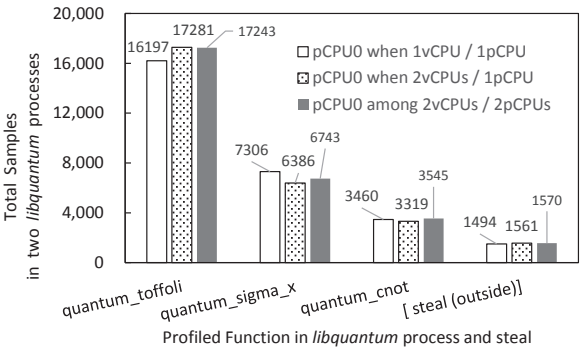

図 2.15 vCPU も pCPU も複数ある場合の pCPU0 上の同じ関数のサンプル数の比較

表 2.10 2pCPU 上で 2vCPU 持った VM における vCPU0 上の Unified VM profiling 結果

Total samples:29998 OS:USER:steal = 1.05%:93.71%:5.23%

Samples	%Ratio	Function	Module
17243	57.48	quantum_toffoli	libquantum(1)
6743	22.48	quantum_sigma_x	libquantum(1)
3545	11.82	quantum_cnot	libquantum(1)
1570	5.23	[steal]	(outside)
458	1.53	quantum_swaptheleads	libquantum(1)
114	0.38	quantum_objcode_put	libquantum(1)
38	0.13	pvclock_clocksource_read	vmlinux
26	0.09	apic_timer_interrupt	vmlinux
24	0.08	native_apic_mem_write	vmlinux
9	0.03	rb_reserve_next_event	vmlinux

表 2.11 2pCPU 上で 2vCPU 持った VM における vCPU1 上の Unified VM profiling 結果

Total samples:30000 OS:USER:steal = 1.05%:96.36%:2.59%

Samples	%Ratio	Function	Module
17873	59.58	quantum_toffoli	libquantum(2)
6520	21.73	quantum_sigma_x	libquantum(2)
3697	12.32	quantum_cnot	libquantum(2)
777	2.59	[steal]	(outside)
653	2.18	quantum_swaptheleads	libquantum(2)
155	0.52	quantum_objcode_put	libquantum(2)
25	0.08	native_apic_mem_write	vmlinux
23	0.08	pvclock_clocksource_read	vmlinux
21	0.07	apic_timer_interrupt	vmlinux
16	0.05	rb_reserve_next_event	vmlinux

2.4.3　データ収集時のオーバヘッド

提案手法を実際の運用サービスで使用するためには，性能プロファイリング結果の正確さに加えて，低オーバヘッドでなければならない．一方で，性能プロファイリングシステムは統計解析手法であるため，正確さを増すためには収集データを増やす必要がある．この目的のためには，細粒度のデータ収集を行うか長期間のデータ収集を行う必要がある．一般には，前者の方法が採用される．しかし，データ収集周期が短くなればなるほど，オーバヘッドは高くなる．即ち，オーバヘッドと結果の正確さはトレードオフの関係にある．従って，許容可能なオーバヘッド以下でのデータ収集周期の最短周期を把握しておくことが必要である．一般に，サービス運用中に実際に使用されている性能管理ツールのオーバヘッドは，5 % から 10 % である．対して，性能プロファイリングシステムは，性能が重要視される場合の利用も考慮する必要があり，許容可能なオーバヘッドは 5 % 未満である．性能プロファイリングシステムのデータ収集による被測定計算機へのオーバヘッドは，性能監視カウンタオーバフロー割込み処理とデータ収集処理により発生し，不可避である．しかし，本研究では，理想として，1 % 以下のオーバヘッドを目指している．

そこで，本項では，VMM で行う一元的な性能プロファイリングシステムが運用中の計算機サービスで利用できるか判断するために，オーバヘッドを評価する．先ず，本項では，先行研究の Du ら [44] と同じような評価ワークロードを用いた．このワークロードは，浮動小数点演算を行う二つの関数で構成されている．実行すると，一つの関数で CPU の実行時間の約 80 %，もう一つの関数で約 20 %消費する．このようなプログラムを固定回数実行した時の，性能プロファイリングシステムのデータ収集の有無による実行時間比をオーバヘッドとする．ここで，物理計算機と VM のそれぞれにおいて，Du らのオーバヘッドと本提案手法によるオーバヘッドを比較し，本提案手法の低オーバヘッドを示す．なお，Du らや Xenoprof [43] などの既存手法と本提案手法とのデータ収集手法の大きな違いは，デリゲーションの有無となる．本評価測定での，データ収集周期は，Du らにあわせて 2.5 ミリ秒周期を用い，評価ワークロードの実行は VM 上で行った．

物理計算機では，図 2.16 に示すように，Du らのオーバヘッドは 0.048 %，本提案手法のオーバヘッドは 0.041 % で，ほぼ同程度のオーバヘッドある．従って，評価環境条件としてほぼ同等と言える．Du らの分析によれば，VM において，物理計算機の場合から追加されるデータ収集オーバヘッドは，デリゲーション無しなら 0.077 % となる．従って，デリゲーションがなければ，VM におけるデータ収集オーバヘッドは 0.125 % と予測される．これに対して，Du らの VM でのデータ収集オーバヘッドは 0.441 % と予測値より大きい．一方，VMM で行う一元的な性能プロファイリングシステムによるデータ収集オーバヘッドは 0.133

% となり，先行研究手法よりも低オーバヘッド，かつ予測通りデリゲーション排除効果が確認できた．

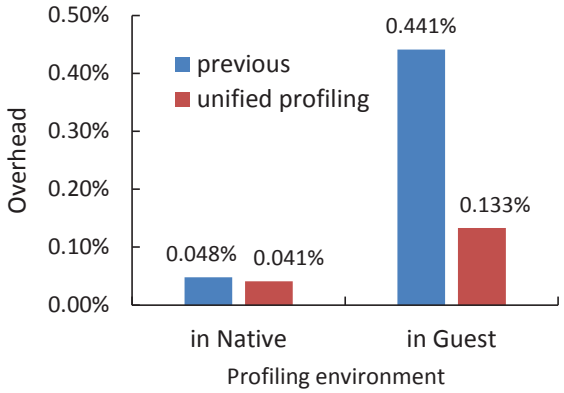

図 2.16 関連研究（previous）と提案手法（unified profiling）のオーバヘッドの比較（2.5 ミリ秒データ収集周期）

図 2.17 は，VMM で行う一元的な性能プロファイリングシステムによる，収集周期の変化に対する，オーバヘッドの変化（青線）とオーバヘッドの許容ライン（赤線）を示す．この結果から，VMM で行う一元的な性能プロファイリングシステムでは，VM 環境でも許容可能なデータ収集周期として 1 ミリ秒を使用できることが分かる．データ収集周期が 1.0 ミリ秒の時の VM 上ワークロードに対するオーバヘッドは 0.302 %，データ収集周期が 0.1 ミリ秒の時の VM 上ワークロードに対するオーバヘッドは 3.084 % である．このように，1 ミリ秒のデータ収集周期の結果は， 1 % より十分に低いオーバーヘッドである．

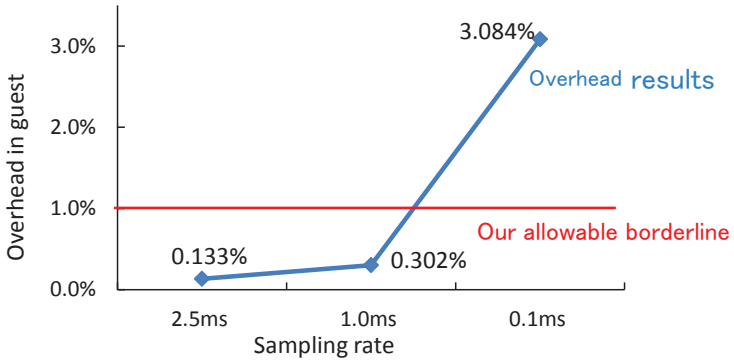

図 2.17 データ収集周期によるオーバヘッドの変化

次に，MySQL [64] を用いた，オンライントランザクション処理（OLTP）性能に対するオーバヘッド評価を行った．MySQL は，オープンソースソフトウェアのリレーショナルデー

タベース管理システム（RDBMS）として広く利用されている．また，MySQL は，Web やデータベース（DB）などのシステムを構築するために広く利用されている LAMP（Linux, Apache, MySQL, and PHP/Perl/Python）のコンポーネントの一つでもある．MySQL の OLTP 性能の測定には Sysbench [67] を用いた．この評価では，pCPU が二つの物理計算機と，その上で動作させる vCPU が二つの VM を用いた．各 vCPU は，特定の pCPU に固定割り当てして動作させた．一つの vCPU には MySQL デーモンを固定し，もう一つの vCPU には SysBench プロセスを固定し OLTP ベンチマークを実行した．ベンチマーク測定は，5 回行った．図 2.18 に，データ収集周期 1.0 ミリ秒の場合と 0.1 ミリ秒の場合の，OLTP スループット性能でのオーバヘッド結果を示す．この結果より，1.0 ミリ秒周期で十分許容可能であることが確認できる．さらに，図 2.17 の評価結果よりもオーバヘッドが低いことも確認できる．これは，オーバヘッドが CPU 使用率に依存するためと考えられる．図 2.19 に，OLTP スループットの平均性能を示す．図 2.20 に，性能プロファイリングシステムによるデータ収集無しの場合の OLTP 実行時の平均 CPU 使用率を示す．図 2.17 の測定で用いたワークロード実行時の CPU 使用率はほぼ 100 % であった．対して，MySQL での本評価時の CPU 使用率は，図 2.20 に示す通り，93 % 以下であった．従って，CPU 使用率が高い方が，データ収集による影響が大きくなりオーバヘッドが高くなると考えられる．

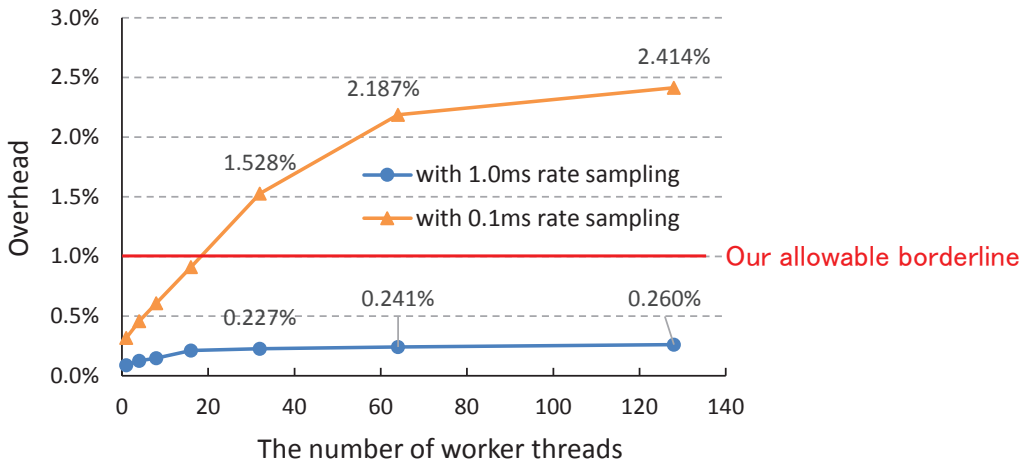

図　2.18　MySQL の OLTP スループット性能でのオーバヘッド

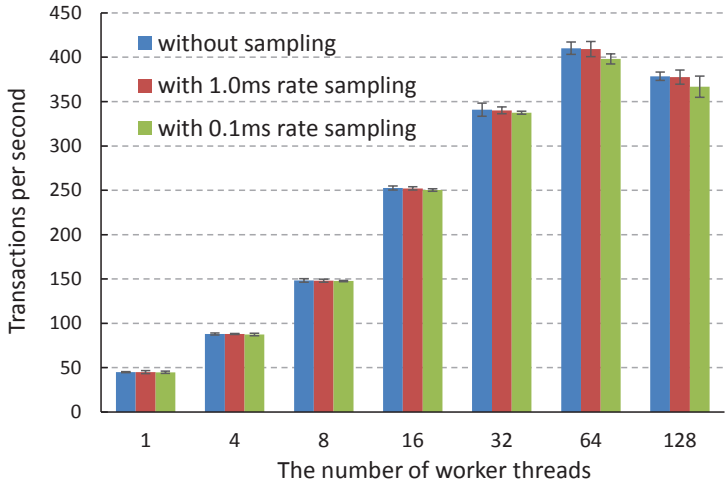

図　2.19　MySQL の OLTP スループット性能

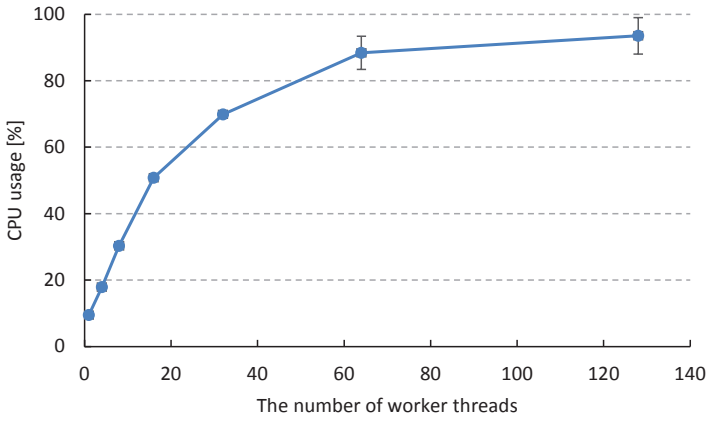

図　2.20　OLTP 実行時の CPU 使用率

2.4.4　提案手法による性能改善事例

本項では，VMM で行う一元的な性能プロファイリングシステムを用いて，VM 上で測定する SPECjbb2013 [68] の性能問題の要因を特定する．SPECjbb2013 ベンチマークは，最新の Java アプリケーション機能に基づいた性能測定を行うために開発された．Java サーバーの性能に関心を持つすべてのユーザーに関係するベンチマークである．本項では，4 つの pCPU と 24 GB の物理メモリーを搭載した物理計算機を用いる．VM には，4 つの vCPU と 20 GB のメモリを割り当てる．各 vCPU は，それぞれ別の特定の pCPU に固定割り当てしておく．VM 上の JVM の動作の特性を理解するために，SPECjbb2013 が実行されている VM の性能プロファイリングを行う．データ収集は，収集周期 1 ミリ秒で 60 秒間行った．この 60 秒間のデータ収集中は，ほぼ最大性能（max-jOPS）でベンチマークが実行されているタイミングで行った．

表 2.12 に，性能プロファイリング結果を示す．この結果から，JVM の処理関数としては，SpinPause 関数の CPU 使用率が 11.10% ，ParMarkBitMap::live_words_in_range 関数の CPU 使用率が 10.05% で，この二つの処理が上位で確認できる．しかし，これらの処理は，通常状態の SPECjbb2013. の性能プロファイリング結果では，観測されない処理である．ParMarkBitMap::live_words_in_range 処理は，ガーベージコレクション（GC）の実装に含まれる処理である．従って，GC 処理によるロック衝突（SpinPause）が発生し，性能が低下したと推測できる．

表　2.12　VM 上の SPECjbb2013 の性能プロファイリング結果

Total samples:240007　　OS:USER:IDLE(halt):steal = 7.43%:81.83%:0.01%:10.73%

Samples	%Ratio	Function	Module
74427	**31.01**	**(jvm internal functions)**	**java**
26637	**11.10**	**SpinPause**	**libjvm.so**
25749	10.73	[steal]	(outside)
24123	**10.05**	**ParMarkBitMap::live_words_in_range**	**libjvm.so**
9174	3.82	ParallelTaskTerminator::offer_termination	libjvm.so

そこで，GC 処理でのロック衝突を防ぐため，十分なヒープメモリを設定することとする．この時の JVM ヒープ・メモリサイズが 8 GB だったので，16 GB に設定変更して，再測定した．その結果，図 2.21 に示すように，SPECjbb2013 のスループット性能が改善された．また，表 2.13 に，性能改善後に収集したデータによる性能プロファイリング結果を示す．こ

の結果から，SpinPause や ParMarkBitMap::live_words_in_range が消えていることと，JVM の動作比率が向上していることが確認できる．

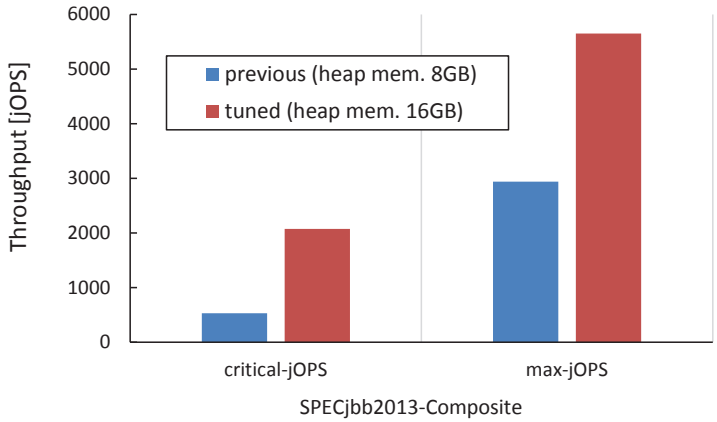

図　2.21　性能改善前後の性能比較

表　2.13　性能改善後の SPECjbb2013 の性能プロファイリング結果

Total samples:240009　　OS:USER:IDLE(halt):steal = 5.44%:90.08%:0.02%:4.46%

Samples	%Ratio	Function	Module
180019	**75.01**	**(jvm internal functions)**	**java**
10715	4.46	[steal]	(outside)
3376	1.41	JVM_LatestUserDefinedLoader	libjvm.so
2265	0.94	PSPromotionManager::copy_to_survivor_space	libjvm.so
1628	0.68	pvclock_clocksource_read	vmlinux

2.5　まとめ

クラウドの基盤として，VM 環境が普及してきた．しかし，VM における性能プロファイリングシステムは，未解決の問題があるため，物理計算機における性能プロファイリングシステムのように普及していない．従って，クラウドの性能低下異常の要因処理を特定することは，非常に困難である．

本章では，先ず，VM における低オーバヘッドな性能プロファイリングシステムに向けた問題点と課題を提示し，VMM で行う一元的な性能プロファイリングシステムを提案した．

具体的には，本提案手法を実現するための三つの手法を説明した．次に，VMM で行う一元的な性能プロファイリングシステムを開発し，VM においても，期待通りの正確な性能プロファイリング結果が得られることを実証した．さらに，既存手法と比較して，データ収集時の低オーバヘッドを示し，許容可能なオーバヘッドでのデータ収集周期を提示した．最後に，本提案手法を用いた性能改善事例を説明し，VMM で行う一元的な性能プロファイリングシステムの有用性を示した．

第 3 章

物理/仮想CPU数の違いを考慮した測定精度の向上

3.1　概要

計算機において，性能低下異常の検出やサービスの使用時間に基づく課金システムのために，プログラムの実行時間を正確に測定することが必要不可欠である．しかし，仮想計算機（VM：Virtual Machine）上で動作するプログラムの実行時間には，**スチール時間** と呼ばれる時間が含まれるため，VM を基盤とするクラウド上でのアプリケーションの実行時間を正確に測定することは困難である．スチール時間は，VM が動作している物理 CPU（pCPU）の実行制御を他 VM に奪われて自 VM が動作できなくなる待ち時間である．VM の各プログラムのスチール時間は，既存のオペレーティングシステム（OS：Operating System）の標準ツールでは認識できない．従って，パフォーマンスエンジニアが，VM での各プログラムの正確な実行時間を把握することは非常に困難である．

本章では，新たな広義のスチール時間の定義を行い，2 章で述べた性能プロファイリングシステムにより，広義のスチール時間も含め，スチール時間を正確に反映した解析が可能であることを示す．さらに，VM の各プログラムの関数レベルの実行時間を補正する手法を説明する．この補正手法は，VM モニタ（VMM）で行う一元的な性能プロファイリングシステムによる関数レベルの時系列収集データに基づき，測定時間（見かけ上の実行時間）からスチール時間を差し引くことによって実現される．本手法の実装は，CPU 性能監視カウンタを利用する VMM レベルのカーネルモジュールとユーザーレベルの分析プログラムに対して行う．従って，VMM や OS，ユーザアプリケーションでの変更は不要である．本手法の評

価では，本手法により，1 % 以下のオーバヘッドで，VM 上の実行関数の正確な実行時間を測定できることを示す．

3.2　広義のスチール

3.2.1　スチールの定義

一部の VM では，sysstat のような OS 標準の統計情報ツールを使用して，VM 全体または仮想 CPU（vCPU）単位でのスチールによる CPU 消費率を表示できる．これは，VMM が，vCPU 単位でスチールに関する統計情報を収集し，VM 上のゲスト OS に提供することにより可能となっている．*the Linux man-pages proc(5)* [69], では，スチール時間は，他の VM で費やされた時間として定義されている．しかし，VM のスチール時間には，他の VM で費やされた時間だけではなく，VMM で費やされた時間も含まれる．本論文では，他の VM に実行制御を奪われることを狭義のスチール，他の VM で消費された時間を狭義のスチール時間としタイプ 1 と定義し，他の VM および VMM に実行制御を奪われることを広義のスチール，他の VM および VMM で消費された時間を広義のスチール時間としタイプ 2 と定義する．以降で示すように，既存の標準ツールでは，タイプ 2 のスチールを把握することができない．

3.2.2　スチールの問題

表 3.1 に，実験環境を示す．物理計算機は，Fujitsu Primergy CX400 M1 server を用いた．実験結果から不確定要素を排除し安定した実験結果が得られるようにするため，CPU ソケットは 1 ソケットだけを有効にし，CPU の Hyper-Threading 機能と Enhanced SpeedStep 機能と Turbo Boost 機能を無効にした．

この実験環境で，PostgreSQL 9.2.7-1 [70] を使用し，タイプ 2 のスチールを評価した．PostgreSQL は，広く使用されているオープンソースのリレーショナルデータベース管理システム（RDBMS）で，Web アプリケーション用のデータベース（DB）としても普及している．PostgreSQL のオンライントランザクション処理（OLTP）の性能測定には，SysBench 0.4.12-12 [67] を使用した．本項では，VM を一つだけ起動し測定に使用した．VM には vCPU が二つあり，各 vCPU を特定の pCPU に固定割り当てした．PostgreSQL プロセスを一つの vCPU（vCPU0）に固定割り当てし，SysBench プロセスをもう一つの vCPU（vCPU1）に固定割り当てした．SysBench から OLTP スレッドを同時に 16 スレッド実行し，CPU 使用率をほぼ 100 %の状態にした．

表 3.1 実験環境

ホスト環境	
CPU	Intel Xeon E5-2697 v3, 2.60 GHz
Num. of pCPU	14
Memory	128 GB
OS	RHEL Server 7.1[1] 64 bit
	(kernel 3.10.0-229.el7.x86_64)
VMM	qemu-kvm-1.5.3-86.el7.x86_64
ゲスト環境	
Num. of vCPU	2
Memory	8 GB
OS	RHEL Server 7.1[1] 64 bit
	(kernel 3.10.0-229.el7.x86_64)

[1]Red Hat Enterprise Linux Server release 7.1

表 3.2 Unified VM profiling 結果

Total samples:29932

Samples	%Ratio	Function	Module
2366	7.90	[steal]	(outside)
1098	3.67	rb_iterate	postgres
855	2.86	ip6t_do_table	ip6_tables
655	2.19	hash_search_with_hash_value	postgres
575	1.92	SearchCatCache	postgres
532	1.78	PostgresMain	postgres
(中略)			
9	0.03	[idle]	(halt_exit)
(後略)			

先ず，OS 標準の統計情報ツールである ***mpstat*** を用いて，vCPU0 での CPU 使用率を測定した結果は，**%usr was 53.27, %sys 23.33, %soft 23.36, %idle 0.03 and %steal 0.00** であった．対して，VMM で行う一元的な性能プロファイリングシステム [71]（2 章）による結果からは，表 3.2 に示す通り，vCPU0 上でのタイプ 2 のスチールによる空白時間比率は，7.90 % であった．このスチールの要因内訳を VM-Exit reason number から調べると，VM からの WRMSR 命令の実行（VM-Exit reason number 32）が 90.25 %，外部割込み（VM-Exit reason number 1）が 9.62 %，EPT 違反（VM-Exit reason number 48）が 0.08%，VM からの IO 命令の実行（VM-Exit reason number 30）が 0.04 % であった．他の収集データでは，タイプ 2 のスチール要因として，VM からの CPUID 命令の実行（VM-Exit reason number 10）や VM からの PAUSE 命令の実行（VM-Exit reason number 40），EPT 設定ミス（VM-Exit reason number 49）も確認できた．これらの結果から，同じ物理計算機上で動作する他 VM がなくても，スチールの影響が発生することが分かった．さらに，VMM からの統計情報を用いる既存のツールでは，このようなタイプ 2 のスチールが認識できないことが分かった．

3.3　スチール時間を反映した解析

VM が pCPU にスケジューリングされていなかったために動作できなかった時間を空白時間と呼ぶ．VM における性能低下異常の検出のためには，空白時間が，スチールによるものかホルトによるものかを区別する必要がある．スチールは，VMM の操作で発生する．要因は，VMM 内か他 VM にある．つまり，スチール要因は，自 OS 環境（自 VM 環境）の外部にある．対して，ホルトは，OS が CPU に halt 命令を発行して発生する空白時間で，OS のスケジューリング制御下にある．従って，ホルト要因は自 OS 環境内（自 VM 環境内）にある．

本節では，先ず，2 章の表 2.1 の評価環境を使用して，VMM で行う一元的な性能プロファイリングシステムによるスチール時間の解析結果が正しいことを検証する．このために，一つの pCPU 上に三つの VM（guestOS1，guestOS2，および guestOS3）を起動する．各 VM は vCPU を一つだけ持つ．各 VM では，libquantum を同じように動作させる．表 3.3 に，guestOS1 の性能プロファイリング結果を示す．表 3.4 に，VMM の性能プロファイリング結果を示す．表 3.3 より，VM1 におけるスチールは，69.39% であることが分かる．また，表 3.4 から，VMM において，VM1 を除いた CPU 使用率が 69.39% であることが分かる．これらの結果の一致から，VMM で行う一元的な性能プロファイリングシステムによる解析結果の中のスチールの結果が正確であることが分かる．

表 3.3 guestOS2 と guestOS3 によるスチール発生時の guestOS1 の Unified VM profiling 結果（Guest-level profiling 結果）

Total samples:29999　OS:USER:steal = 0.79%:29.82%:69.39%

Samples	%Ratio	Function	Module
20816	69.39	[steal]	(outside)
5156	17.19	quantum_toffoli	libquantum
2286	7.62	quantum_sigma_x	libquantum
1160	3.87	quantum_cnot	libquantum
265	0.89	quantum_swaptheleads	libquantum
73	0.24	quantum_objcode_put	libquantum
21	0.07	apic_timer_interrupt	vmlinux
11	0.04	pvclock_clocksource_read	vmlinux
10	0.03	ring_buffer_lock_reserve	vmlinux
9	0.03	native_apic_mem_write	vmlinux

表 3.4 guestOS1〜guestOS3 の 3VM 動作中の Unified VM profiling 結果（Host-level profiling 結果）

Total samples:29932　OS:USER:VM = 6.81%:0.68%:92.51%

Samples	%Ratio	Function	Module
9402	31.41	[guestOS3]	(VM3)
9161	30.61	[guestOS1]	(VM1)
9125	30.49	[guestOS2]	(VM2)
137	0.46	vmx_vcpu_run	kvm_intel
67	0.22	update_curr	vmlinux
59	0.20	rb_reserve_next_event	vmlinux
59	0.20	(stripped local functions)	qemu-kvm
50	0.17	copy_user_generic_string	vmlinux
49	0.16	kvm_arch_vcpu_ioctl_run	kvm
39	0.13	ring_buffer_lock_reserve	vmlinux

次に，guestOS2 と guestOS3 の二つの VM を停止する．VMM で行う一元的な性能プロファイリングシステムによる解析結果の中のホルトの結果（idle）の正しさを検証するため，1VM（guestOS1）のみを起動状態とする．この検証では，データ収集時に libquantum を 10 秒間だけ動作させる．データ収集時間は，表 2.2 にある通り，30 秒間である．残りの 20 秒間，guestOS1 は，自発的に idle 状態となる．従って，VM 上でも VMM 上でも，idle の%Ratio が，約 67 %（20 秒/30 秒 = 66.7 % ）と期待する．

表 3.5 に，VM 上の性能プロファイリング結果（Guest-level profiling）を示す．表 3.6 に，VMM 上の性能プロファイリング結果（Host-level profiling）を示す．実際に，表 3.5 の結果から，VM 上では，idle が 67.40% であることが分かる．一方，VMM 上での idle（poll_idle 関数）は，表 3.6 より，66.73% であることが分かる．これらの結果は互いに近い値であり，かつ期待通り 67 % である．従って，VMM で行う一元的な性能プロファイリングシステムは，VM のホルトを正確に把握できる．さらに，本節の結果から，VMM で行う一元的な性能プロファイリングシステムでは，VM の空白時間をスチールかホルトの二つの状態に分類できることが分かった．これは，VM-Exit reason number の収集と VMM 時間による共通時間ベースの解析手法の両方の効果である．

表 3.5 ホルト状態を含む guestOS1 の Unified VM profiling 結果（Guest-level profiling 結果）

Total samples:30000　OS:USER:IDLE:steal = 0.54%:30.42%:67.40%:1.64%

Samples	%Ratio	Function	Module
20220	67.40	[idle]	(halt_exit)
4987	16.63	quantum_toffoli	libquantum
2326	7.76	quantum_sigma_x	libquantum
1063	3.54	quantum_cnot	libquantum
492	1.64	[steal]	(outside)
392	1.31	quantum_gate1	libquantum
237	0.79	quantum_swaptheleads	libquantum
61	0.20	quantum_objcode_put	libquantum
50	0.17	__mulsc3	libquantum
13	0.04	apic_timer_interrupt	vmlinux

表 3.6 1VM（guestOS1）のみ動作中の Unified VM profiling 結果（Host-level profiling 結果）

Total samples:29943　OS:USER:IDLE:VM = 2.17%:0.17%:66.73%:30.93%

Samples	%Ratio	Function	Module
19982	66.73	poll_idle	vmlinux
9260	30.93	[guestOS1]	(VM1)
25	0.08	vmx_vcpu_run	kvm_intel
21	0.07	ring_buffer_lock_reserve	vmlinux
17	0.06	update_curr	vmlinux
17	0.06	do_select	vmlinux
15	0.05	rb_reserve_next_event	vmlinux
15	0.05	(stripped local functions)	qemu-kvm
13	0.04	trace_clock_local	vmlinux
11	0.04	apic_timer_interrupt	vmlinux

3.4　仮想計算機のプログラムの実行時間補正法

3.4.1　対処

先ず，本提案手法の前提条件として二つあげる．一つは，図 3.1 に示すように，VMM で行う一元的な性能プロファイリングシステムによるデータ収集周期（図中の上矢印の間隔）が VM 切り替え周期（図中の空白時間）より短いことである．例えば，VM 切り替え周期が 20 ミリ秒でデータ収集周期が 1 ミリ秒である．もう一つは，スチールによる空白時間の前後では実行プログラムが同じであることである．これにより，あるスチール時間がどのプログラムの実行中に発生したスチール時間かが分かる．このことを使って，実行プログラムの実行時間の補正を行う．補正対象とするプログラムの粒度は，プロセスレベルから関数やスレッドまで含む．補正について，例えば，図 3.1 において，スチール時間（空白時間）の前後は，同じ関数 Func A が実行されている．これにより，図 3.1 に示すように，このスチール時間は関数 Func A に属するスチール時間であると認知できる．このように，関数 Func A の見かけ上の実行時間は，実際に CPU 上で動作した実行時間より大きくなる．しかし，VM 上で動作している OS はスチール時間の発生に気づけない．これは，スチール発生時，VM は非自発的に実行制御を VMM に奪われるためである．なお，VMM で行う一元的な性能プロファイリングシステムによる時系列の収集データにおいて，スチール時間の前後で動作プログラムが異なる場合，このようなスチール時間は実行時間の補正には用いないこととする．このようなスチール時間は，対応付けられるプログラムが特定できないためである．

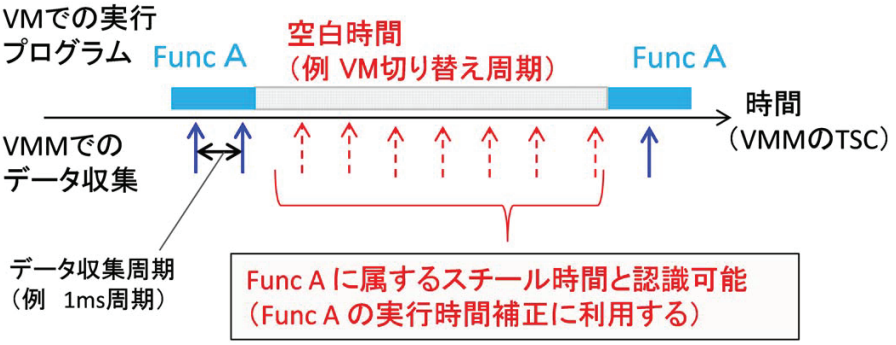

図　3.1　VMM でのデータ収集周期と VM の切り替え周期の関係

提案手法のポイントは以下の通りである．第一項目の具体的な方法については，次項で説明する．

- 先ず，VM の疑似収集データの中のスチールに関するデータに対し，どのプログラムの実行中にスチールしたかを表すスチールの属性情報を生成付与する．(3.4.2 項)

- 次に，属性情報に基づき補正対象プログラに属するスチールのデータを抽出し，抽出したスチールのデータ数とデータ収集周期を乗算することで，補正対象プログラムのスチール時間の合計時間 T を算出する．

- 最後に，VM 上で測定したあるプログラム（補正対象プログラム）の実行時間（見かけ上の実行時間）から，補正対象プログラムに属するスチールの合計時間 T を差し引くことにより，実際に CPU を使用した正確な実行時間を得る．

3.4.2　スチールの属性情報の生成方法

VM の疑似収集データの生成

VM には，pCPU が割り当てられなかったために動作できなかった時間（空白時間）がある．空白時間帯の VM のデータ収集は，行われない．このため，空白時間は，VM に対応する有効な収集データが無い時間である．従って，頻度集計前に各 VM の空白時間を割り出し，収集データを補間する必要がある．各 VM の空白時間帯を割り出すために，VM 環境全体で共通の一つの時間軸を使用する．この共通の時間軸として，VMM の時間（物理 TSC）を使う．これにより，VMM で行う一元的な性能プロファイリングシステムは，各 VM の空白時間も含めた性能プロファイリングシステムである．

VM の空白時間の割り出しは，vCPU 毎に行う．図 3.2 に，VM の空白時間を割り出し，空白時間で収集データを補間する疑似収集データの生成方法を示す．図 3.2 は，データ収集対象環境として，二つの pCPU（pCPU0 と pCPU1）があり，二つの VM（VM0 と VM1）が実行されている環境を想定した図である．VM0 には一つの vCPU（vCPU0）があり，VM1 には二つの vCPU（vCPU0 と vCPU1）があるとする．vCPU は pCPU に固定割り当てされていないとする．時間軸は，VMM の時間による経過時間とする．この図 3.2 では，t1 から t4 までの時間に収集された収集データを示す．VM の疑似収集データは，VMM の時間を基準時間として，VMM で収集したデータから vCPU 単位に生成する．図 3.2 の上図（a）は，VMM で収集された収集データを示す．この収集データから，VM の疑似収集データを生成する．図 3.2 の下図（b）は，生成後の各 VM の疑似収集データを示す．生成は，以下のように行う．

(1) 先ず，VMM で収集したデータを vCPU 単位でグルーピングする．

(2) 次に，グルーピングしたデータを vCPU 毎に VMM 時間軸上に並べる．

(3) 空白部分を空白時間とする．

これにより，各 vCPU の空白時間を割り出すことができる．

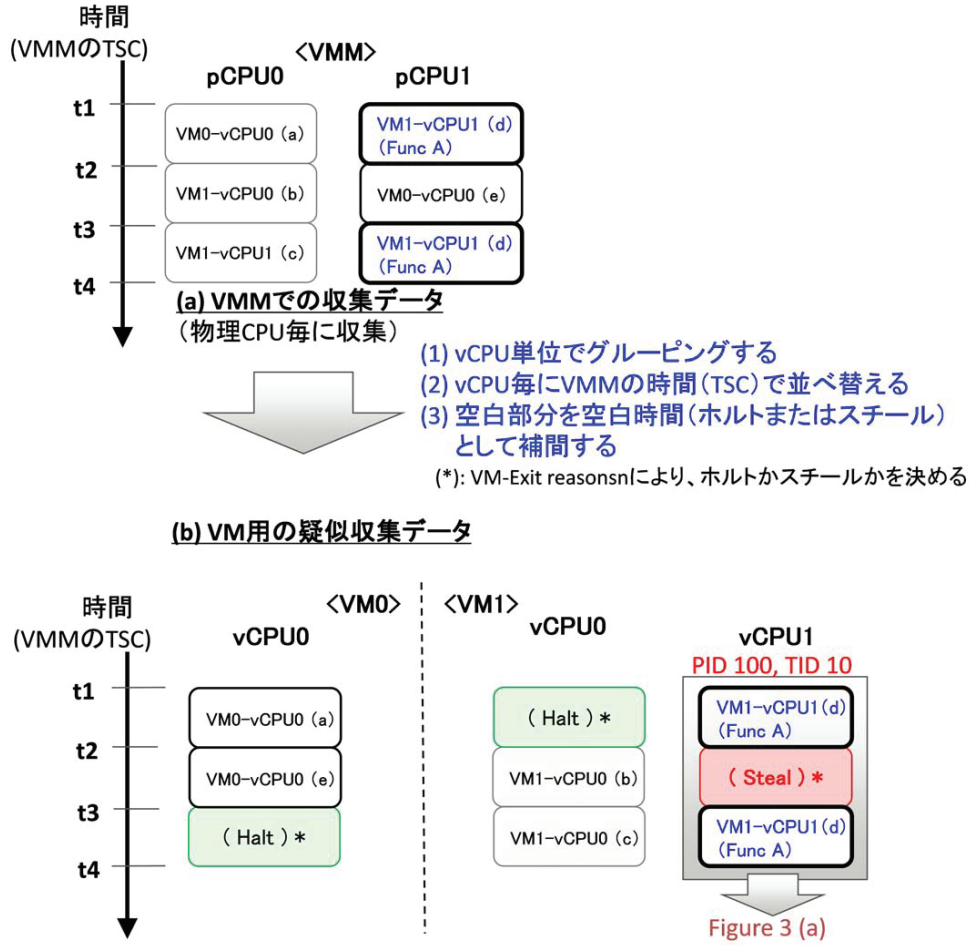

図　3.2　VM の疑似収集データの生成

さらに，空白時間は，ゲスト OS がホルト命令を発行することにより発生する自発的な空白時間と，スチールによる空白時間の二つの状態に分類できる．この分類には，VM-Exit reason number を使用する．VM-Exit reason number が 12 の場合は，空白時間をホルトに分類する．12 以外の場合は，スチールに分類する．vCPU がスチール状態にある場合，VM は，実行できる状態にあるが，使用可能な pCPU が無いため，pCPU に割り当てられていない状態である．図 3.2 の下図（b）より，VM1 の vCPU1 では，スチールが発生していることが分かる．具体的には，プロセス ID（PID) が 100，スレッド ID（TID）が 10 の実行スレッドで関数 A の実行中の t2 から t3 の時間にスチールが発生している．このため，この関数 A は，t2 から t3 の間は動作していない．しかし，既存の時間計測では，見かけ上この時間も

関数 A の実行時間に含まれる．

収集データへのスチール属性情報の付与

図 3.3 に，VMM で行う一元的な性能プロファイリングシステムの収集データの内容と保持形式の例を示す．一行が一収集データで，収集順に時系列で並んでいる．PID はプロセス ID，TID はスレッド ID である．また，既にシンボル解決は終わっている状態で，図中の A や [Steal] のある列が関数または関数相当のシンボルの列となる．図 3.3 の上図（a）は，スチールに対して，属性情報（どのプログラムの実行中に発生したスチールなのか，スチールが属するプログラムや関数に関する情報）がまだ無い状態の収集データである．また，この例では，図 3.2 の VM1 の vCPU1 と同じ状態を模している．対して，Figure 3.3 の下図（b）は，スチールの属性情報として，PID と TID と関数名を付与した収集データとなる．スチールの直前と直後のデータが，同じ PID と TID と関数名だった場合に，その PID と TID と関数名を属性情報としてスチールデータに付与する．

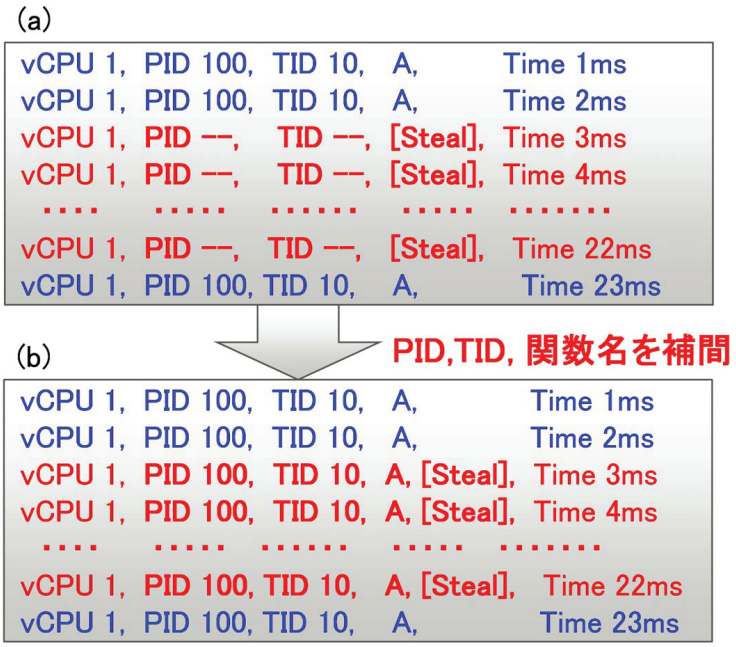

図　3.3　疑似収集データ中のスチールデータへの属性情報の付与

3.5 評価

3.5.1 VM 上でのプログラム実行時間の測定結果と補正結果の比較

本項では，測定した実行時間を正しく補正できるか評価する．評価環境は，3.2.2 項の実験環境（表 3.1 ）と同じである．本項の評価では，vCPU を二つ（vCPU0 と vCPU1）持った VM を三つ起動し，全 VM の vCPU0 を同じ pCPU（pCPU0）に固定割り当てした．これにより，pCPU0 はオーバコミット状態となる．また，各 VM の vCPU1 を pCPU0 以外の pCPU に固定割り当てした．実行時間の測定対象となるワークロードとして，浮動小数点演算を行う関数を繰り返し実行するプログラムを用いた．繰り返し実行する関数は二つ用意した．一つを関数名 compute_a，もう一つを compute_b とし，compute_a と compute_b の CPU 使用率が 80 % と 20 % になるように実行した．従って，このワークロードだけで，CPU 使用率は 100 % となる．このワークロードを三つある VM のうちの一つの VM（実行時間測定 VM）の vCPU0 上に固定割り当てし，関数の繰り返し回数を固定回数として，実行した．他の二つの VM（外乱負荷生成 VM）の vCPU0 にも，このワークロードを用いて，関数の繰り返しを無制限とし負荷をかけ続けた．実行時間測定 VM で，関数の開始時刻と終了時刻による経過時間で各関数の実行時間を測定した．この測定時に，同時に，VMM で行う性能プロファイリングシステムにより，収集周期 1 ミリ秒でデータ収集を行った．データ収集契機は，CPU が備える性能監視カウンタのオーバフロー割込み機能を使って，一定間隔で発生させることができる．割込み毎にプログラムの動作情報を収集し，収集データは全てそのまま，VMM のメモリバッファに，時系列で保持される [71].

図 3.4 に，各関数の見かけ上の実行時間を示す．ベース時間（100 % ）が，実際に CPU を使用した実行時間を示す．これらの結果を見ると，各関数の見かけ上の実行時間は，ベース時間の 2 倍または 3 倍になることがわかる．これは，CPU の演算能力を必要とするワークロードの場合，他 VM によって生成された CPU 負荷に比例して，見かけ上の実行時間が増加することを示している．これに対し，図 3.5 に，実行時間の補正結果を示す．これらの補正後の実行時間は，ほぼベース時間と同じであることが分かる．この結果から，提案手法により，正確な実行時間が得られることが確認できた．また，本提案手法でスチール時間を除いた効果は，図 3.4 と図 3.5 を比較することによって確認できる．

さらに，提案手法を実アプリケーションに適用するための予備評価を行った．評価環境は本項の評価環境と同じである．予備評価のために，実アプリケーションとして PostgreSQL を使用した．評価方法を説明する．先ず，外乱負荷生成 VM で負荷を継続的にかけ続け，次に，実行時間測定 VM において，3.2.2 項と同じように OLTP スレッドを 16 スレッド実行さ

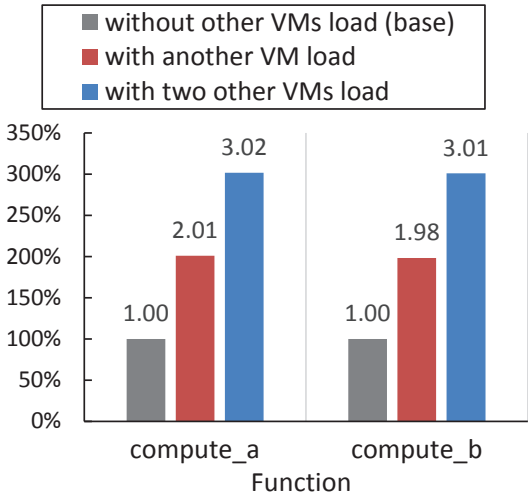

図　3.4　スチール時間を含む見かけ上の実行時間

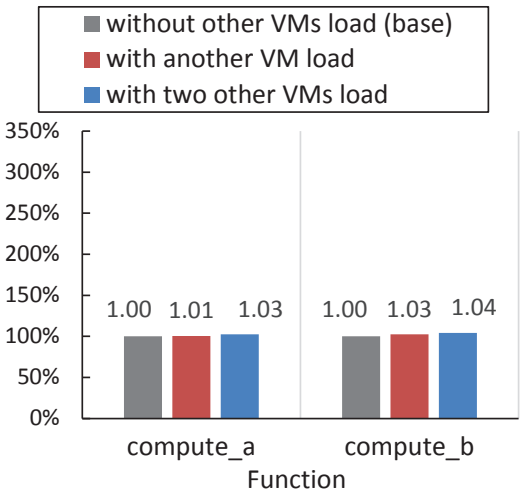

図　3.5　スチール時間を除いた補正後の実行時間

せた．OLTP の実行時間測定中に，同時に，VMM で行う一元的な性能プロファイリングシステムにより，1 ミリ秒の収集周期で 60 秒間データ収集を行った．結果，見かけ上の実行時間は 213.34 秒であった．また，vCPU0 で 60 秒間に 35788 個のスチールデータを収集した．このスチールの収集データ数から，見かけ上の実行時間の中でスチールが占めるスチール時間を式 3.1 で算出すると，127.25 秒である．

$$35788[\text{スチールの収集データ数}] \times 0.001[\text{秒}] \div 60[\text{秒}] \times 213.34[\text{秒}] \tag{3.1}$$

見かけの実行時間に対しスチール時間で差分をとって補正すると，補正後の実行時間は 86.09 秒 である．実際に，他 VM による外乱負荷が無い状態で測定したリファレンス用の実行時間は 89.51 秒であった．このようにして得た実行時間の結果を正規化して，図 3.6 と図 3.7 に示す．これらの図の結果から，提案手法は，実アプリケーションの実行時間を補正するために使用できそうである．一方，OS 標準の統計情報ツール sysstat に含まれる mpstat コマンドによる予備評価も行った．その結果，mpstat が報告する%steal の平均結果は 56.27 %であった．この結果を用いて補正した実行時間は 94.01 秒となる．本提案手法の結果の方が，リファレンス結果により近く正確であることが分かる．この差は，タイプ 2 のスチールによるものと推測される．

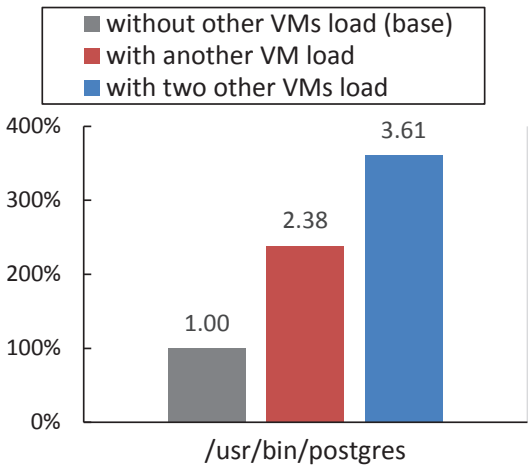

図　3.6　スチール時間を含む見かけ上の PostgreSQL 実行時間

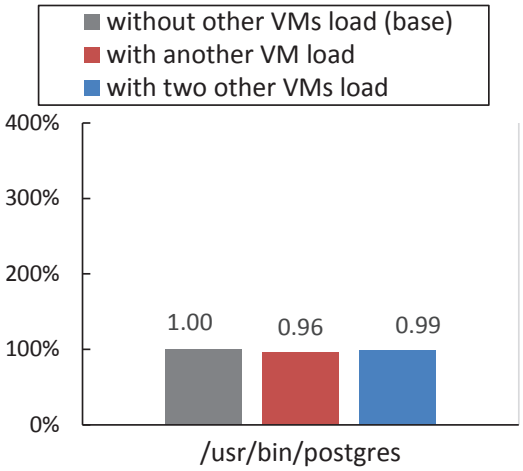

図　3.7　スチール時間を除いた補正後の PostgreSQL 実行時間

3.5.2　性能プロファイリングシステムによるデータ収集時のオーバヘッド

VMM で行う一元的な性能プロファイリングシステムのデータ収集の有無による，OLTP 性能の差異でオーバヘッドを測定した．OLTP の性能測定は，3.2.2 項と同じように OLTP スレッドを 16 スレッド実行させて行った．図 3.8 に，オーバヘッド結果（青線）と，性能要件の厳しいサービスのシステムにも適用できる許容可能なオーバヘッド（赤線）を示す．この結果から，VM 環境でも，1 ミリ秒の収集周期でデータ収集を行えば，提案手法の実サービスへの適用も可能である．

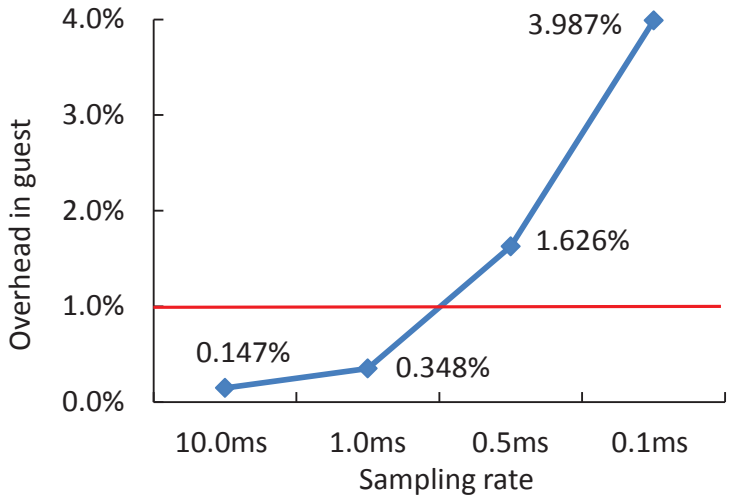

図　3.8　16 スレッドで実行中の PostgreSQL に対する，提案手法のオーバヘッド

3.6　関連研究

Weaver らは，VM 上でのスチール時間の補正を行った時間を返す時間計測関数を PAPI に実装した [53]．この実装は，スチール時間のレポート機能を持った VMM 環境（例えば，Linux での KVM/Xen 環境）に依存している．具体的には，VM が実行スケジュールから外された頻度情報を VMM から受け取り，PAPI が提供する時間計測関数である PAPI_get_virt_usec 関数に対してスチール時間補正を行っている．しかし，この時間計測関数は，VM 単位のスチール時間しか提供しない．従って，VM 上のプロセスや関数の実行時間を補正することは，未解決の問題となっていた．この問題に対して，Weaver らは,「スチール時間は，一台の物理計算機が複数の VM でオーバサブスクライブされている場合の問題である．ほとんどの HPC の状況では，一台のノードにつき一つのタスクしか走行していないため，この問題は重大な制限とはならないかもしれない.」と述べている．一方，一般的なクラウド環境では，これは重大な制限になる．一般的なクラウド環境では，一台の物理計算機が複数の VM でオーバサブスクライブされるからである．これに加えて，3.2 章に示したように，物理計算機が VM でオーバサブスクライブされていなくても，スチール時間は発生する．従って，本研究では，このスチール時間問題を解決することを目的としている．

Hofer らは，VMM から提供されるスチール時間情報を監視対象のアプリケーションスレッド単位に分解した [54]．Hofer らの手法は，最も多くの CPU 時間を消費する（または最も多くの CPU 時間が割り当てられている）スレッドがスチールによる影響を最も受けるスレッドであるという分析結果に基づいている．従って，Hofer らの手法では，各実行スレッドが消費した CPU 時間に比例してスチール時間を実行スレッド間で配分する．Hofer らは，この手法を実際に Java VM に実装した．従って，この実装では，VM 上の Java アプリケーションのみへの対応となる．さらに，この手法は，VMM から提供されるスチール時間情報に依存しているため，タイプ 2 のスチール時間を含めることができない．

3.7　まとめ

クラウド上のアプリケーションの正確な実行時間が必要とされている．しかし，クラウド基盤となる VM 環境では，プログラム単位でスチール時間を把握する手段がないため，各プログラムの正確な実行時間を得ることが困難である．本章では，先ず，pCPU がオーバコミット状態でなくてもスチール時間が発生しうることを明らかにし，VMM 統計情報を使ってもそのようなスチール時間は認識できないことを示した．また，VMM で行う一元的な性能プロファイリングシステムにより，スチール時間を正確に反映した解析ができることを示

した．次に，この性能プロファイリングシステムの収集データに基づいて，プログラム毎のスチール時間を抽出し，見かけ上の実行時間から差し引くことにより，クラウド上のアプリケーションの実行時間を補正する手法について述べた．さらに，この手法を実装し，VM で動作するプログラムに対し，期待通りの正確な実行時間が得られることを実証した．

第 4 章

継続的な性能プロファイリングを可能にするシステムの分散化とデータ収集停止時間の短縮

4.1 概要

計算機の性能異常発生時の要因調査手段として，性能プロファイリングシステムが広く利用されている．仮想計算機（VM：Virtual Machine）における性能プロファイリングシステムとして，データ収集，データ格納，および解析を VM モニタ（VMM）で行う手法が提案されている．この手法は，この一連の処理を一度のみ実行する．一方，計算機の性能異常を検出するには，収集データを定期的に格納し解析して，継続的に性能プロファイリングを行う必要がある．また，収集データを格納する際に発生するデータ収集の停止時間を短縮することも重要である．そこで，本章では，継続的な性能プロファイリングを行う構成として性能プロファイリングシステムの分散化，およびデータ格納処理の短縮法として優先処理方式および並行動作処理方式を述べ，評価結果を報告する．

4.2 仮想計算機を利用した性能プロファイリングシステム

4.2.1 構成

VM を利用した性能プロファイリングシステム [71] について，図 4.1 にプログラム構成とデータの流れを示す．これは，一台の物理計算機（ホスト）と一台の VM（ゲスト）で構成

自然科学研究科
産業創成工学専攻

された仮想化環境の例である．プログラムは，以下の四つからなる．

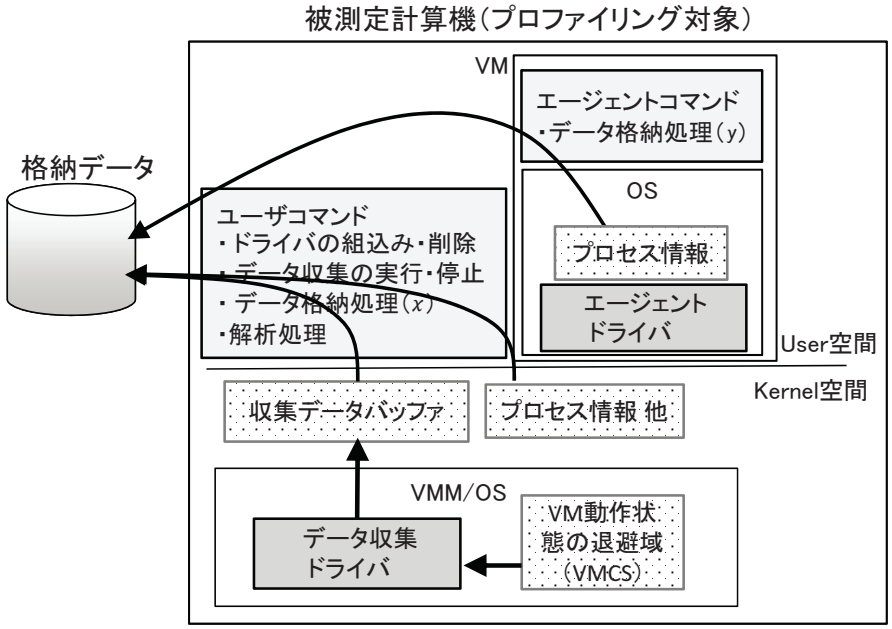

図 4.1 プログラム構成とデータの流れ

ユーザコマンドは，データ収集ドライバの VMM への動的な組込み（ロード）や VMM からの動的な削除（アンロード），およびデータ収集の実行開始の指示を行う．データ収集の開始指示時に，収集周期や収集時間の指定も行う．また，データ収集終了時に，VM 上のエージェントコマンドに対して，プロセス情報やオブジェクトファイルなどの各データのディスクへの格納指示を発行し，かつ VMM 上でも収集データや VMM 上のプロセス情報やオブジェクトファイルなどの各データのディスクへの格納処理および解析処理を行う．これらの処理のうち，VMM 上のデータ格納処理を行うプログラムをデータ格納処理プログラム x とする．

データ収集ドライバは，VMM 内にあり，指定されたデータ収集周期で，測定対象となるアプリケーションプログラムの動作情報のデータ収集を行う．VM 上で動作しているアプリケーションプログラムの動作情報は VM 動作状態の退避域から入手する．この退避域は，動作中の VM と物理 CPU（pCPU）の割り当てを待っている VM との間または VM と VMM との間で実行制御が切り替わる際に，CPU のレジスタ値などハードウェア情報を退避復元するためのメモリ領域で，Intel CPU では Virtual Machine Control Data Structure (VMCS) [72] と呼ばれている．周期的なデータ収集は，データ収集開始から，指定されたデータ収集時間が経過するまで行われる．また，データ収集開始直前には，初期設定として，指定バッファサイズに基づいた収集データバッファの確保や，収集周期トリガを生成するための CPU の

性能監視カウンタ（PMC）へのカウントベースイベントと初期カウンタ値の設定や，収集時間に基づいたタイマの設定を行う．

エージェントコマンドは，データ収集終了時に，ホスト上のユーザコマンドからの指示により，ゲスト内のプロセス情報やオブジェクトファイルなどの各データをホスト上のディスクへの格納処理を行う．このゲスト上のデータ格納処理を行うプログラムをデータ格納処理プログラム y とする．また，エージェントドライバに対し，CR3-PID のマッピング情報格納を依頼する．CR3 は Intel CPU のコントロールレジスタ 3 で，ページテーブルアドレスを保持している．

エージェントドライバは，Linux カーネルの各プロセスのタスク構造体から CR3 と PID の値を対にして格納する．

4.2.2 データ

データ（A）は VMM 上のみから，データ（B）～（E）は VMM 上と VM 上から格納する．

（A）プログラム動作情報

（B）プロセス情報

（C）オブジェクトファイル

（D）カーネルのシンボルマップ

（E）実行条件や環境情報など付帯情報

プログラム動作情報（A）は，データ収集ドライバが周期的に収集する情報で，命令アドレス（IP）や PID などを含む．プログラム動作情報の具体的な内容を表 4.1 に示す．以降，プログラム動作情報（A）のことを収集データとも呼ぶ．

プロセス情報（B）は，データ収集終了時に存在する各プロセスの情報で，PID やプロセス名，ロードされているプログラムのメモリマップ情報やオブジェクトファイルのファイルシステム上のパス情報である．

オブジェクトファイル（C）は，プロセス情報（B）に含まれる各プロセスのオブジェクトファイルである．即ち，データ収集中に動作していたプログラムの実行バイナリファイルで，プロセス情報（B）から得たパス情報を使ってファイルシステム上からコピーしたものである．

カーネルのシンボルマップ（D）は，動作中のカーネルのシンボルマップファイルで，カーネルの関数シンボル名とメモリマップ情報である．

付帯情報（E）は，収集周期や収集時間などの実行条件や動作環境の情報など参考情報である．

表 4.1 プログラム動作情報（収集データ）

名前	size (bytes)	説明
Host_IP	8	ホスト上の命令アドレス
Host_Thread_ID	4	カレント・スレッド ID
Host_PID	4	カレント・プロセス ID
Host_Return_IP1	8	戻りアドレス 1
Host_Return_IP2	8	戻りアドレス 2
TSC	8	タイムスタンプカウンタ
vPROCESSOR_ID	8	CPU 管理の仮想 CPUID
Guest_IP	8	VM 上の命令アドレス
Guest_CR3	8	VM 上のページテーブルアドレス
VMEXIT_REASON	8	VM_EXIT 要因番号
VMEXIT_INTRINFO	8	VM_EXIT 割込み情報
(reserved)	16	予備

4.2.3 処理流れ

既性能プロファイリングシステム [71] の処理流れを以下に示す．

（1）ユーザから指定された収集周期と収集時間でメモリバッファ上にデータ収集を行う．

（2）収集後にデータ（A）～（E）をディスクに格納する．

上記の処理後，サービスを停止させ，格納したデータを基に解析処理を行う．

4.3 性能プロファイリングシステムの分散化

4.3.1 既性能プロファイリングシステムの問題と対処

既性能プロファイリングシステム [71] は，計算機の性能異常発生時の要因調査手段である．このため，このシステムは，データ収集，データ格納，および解析の処理を一度しか実行しない．つまり，サービス実行中にデータ収集とデータ格納を行い，後に解析を行う．従って，継続的に動作する計算機の性能異常を検出できない問題がある．

継続的に動作する計算機の性能異常を検出するには，データ収集，データ格納，および解析をサービス実行中に繰り返し行う必要がある．また，解析の処理は負荷が大きいため，サービス処理への影響を抑制することが重要である．そこで，既性能プロファイリングシステムの分散化を行い，継続的に動作する計算機の性能異常検出を可能にする．

4.3.2 分散化した性能プロファイリングシステム

既性能プロファイリングシステムを分散化したシステム構成を図 4.2 に示す．図 4.1 に示した四つのプログラムのうち，ユーザコマンドの解析処理機能を性能プロファイリング対象マシンから解析処理計算機に分離する．また，データの格納先を解析処理計算機のリモートディスクへ変更する．データ格納処理を文献 [71] と同じ処理にするため，リモートディスクへの格納は分散ファイルシステムを利用する．処理の流れを以下に説明する．

（1）ユーザから指定されたデータ収集周期とデータ収集時間でメモリバッファ上にデータ収集を行う．

（2）収集後にデータ（A）とデータ（B）をディスクに格納する．
以降,（1）と（2）の処理を繰り返す．

（3）格納データを基に定期的な解析処理を行う．

ここで，性能プロファイリングデータにノイズ（格納処理の影響）が載ることを防ぐために，処理（2）のデータ格納処理は，収集毎ではなくまとめて定期的に行う．また，データ格納処理中は，処理（1）のデータ収集を停止させる．OS や VMM の性能統計情報の収集では，性能統計情報データを収集毎に格納する．しかし，性能プロファイリングでは，従来手法 [33, 32, 34] においても，データの収集と格納は同時に行わない．これは，性能統計情報の収集に比べて，性能プロファイリングは，データ収集周期が千分の一以下と短く，かつ単位時間あたりの収集データ量が大きく，格納処理が収集データに及ぼす外乱が無視できな

いためである．なお，この停止時間をデータ収集停止時間と呼ぶ．さらに，データ収集時間とデータ収集停止時間の和の時間を格納周期とする．

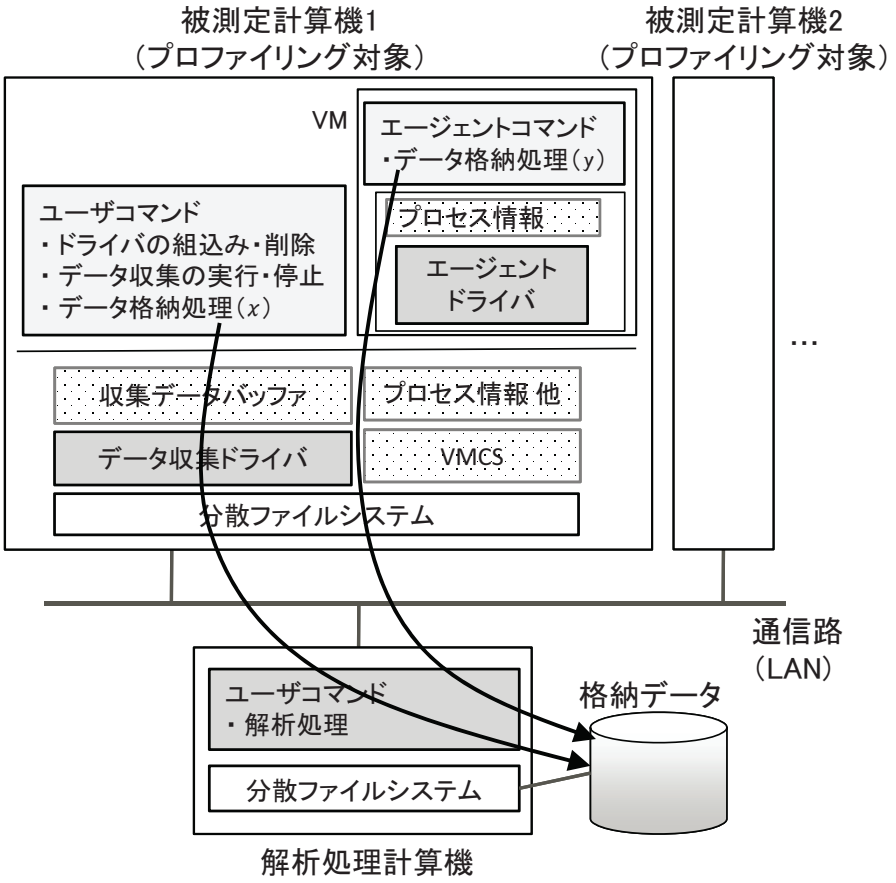

図 4.2 性能プロファイリングシステムの分散化の構成例

また，処理（2）で定期的に格納するデータは，データ（A）の収集データとデータ（B）のプロセス情報のみとする．その他のデータ（C）～データ（E）は静的な情報であり別途入手する．これにより，サービスへの影響を抑制する．

以上に述べたシステムの分散化では収集データをリモートディスクへ格納するため，収集データを定期的に格納する際のデータ収集の停止時間の増加が懸念される．この問題と対処については，4.4 章で述べる．

4.4 分散システムにおけるデータ格納時のデータ収集停止時間の短縮法

4.4.1 定期的なデータ格納処理での問題

データ収集の停止時間を短縮するための課題を明らかにするため，データ収集停止時間を実測し分析する．

実験環境と測定条件を表 4.2 に示す．本項の実験では，仮想 CPU（vCPU）数は 1 に固定とし，VM 数を 1VM〜10VM で測定を行う．各 VM 上でのアプリケーション負荷として，文献 [71] と同じ姫野ベンチマーク [73] に含まれる jacobi 関数を用いる．jacobi 関数は，ポアッソン方程式解法をヤコビ反復法で解くための関数で，配列への連続アクセスとその配列要素に対する浮動小数点演算をループ処理で繰り返し行うプログラムで実装されている．このため，CPU の演算能力やメモリバンド幅を必要とする処理となっている．この結果，jacobi 関数の処理実行では，CPU の演算能力やメモリバンド幅がボトルネックとなる．データ格納処理プログラム x，y と格納データ（A）〜（E）の配置を図 4.3 に示す．格納周期毎（t_1, t_2, ...）に格納するデータは，プログラム動作情報（A）とプロセス情報（B）である．

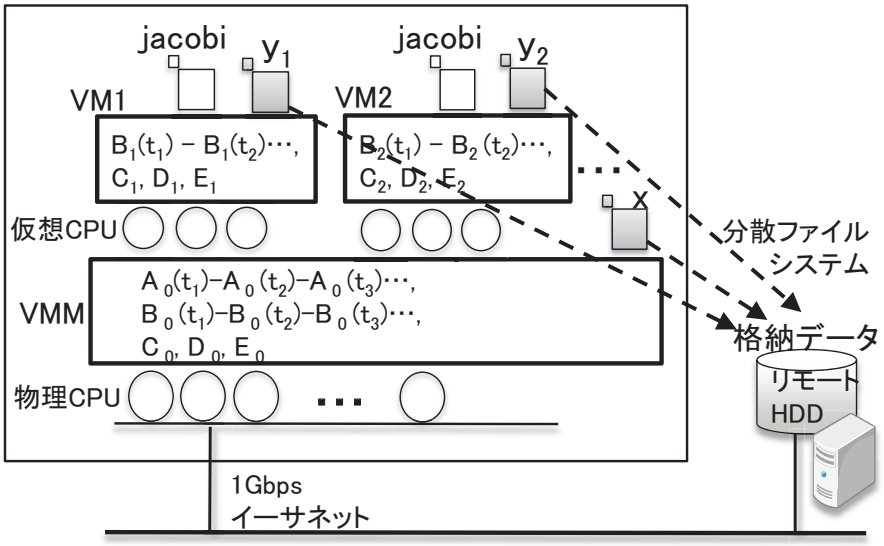

図 4.3 分散システムでのデータ格納処理プログラムと格納データの配置

具体的な測定方法を以下に述べる．

（1）vCPU 数 1 個として VM を起動する．

（2）起動した VM 上で jacobi 関数を繰り返し実行し続けるプロセスを 1 つ起動する．

表 4.2 実験環境と測定条件

ホスト環境	
CPU	Intel Xeon E5-2697v3 2.60GHz 14 コア× 1
Memory	24GB
LAN	1Gbps Ethernet
OS	CentOS Linux 7.2.1511 64bit （kernel 3.10.0-327.28.2.el7.x86_64）
VMM	qemu-kvm-0.12.1.2
ゲスト環境	
Memory（1VM あたり）	4 GB/VM
OS	CentOS Linux 7.2.1511 64bit （kernel 3.10.0-327.28.2.el7.x86_64）
測定条件	
VM 数	1, 2, 3, ..., 10
vCPU 数	1, 2, 3
データ収集周期	1 ミリ秒
データ収集時間と 1VM 時データサイズ	20 秒（35MB）, 30 秒（48MB）, 60 秒（86MB）, 120 秒（163MB）

（3）継続的な性能プロファイリング測定を開始する．

（4）データ収集時間 60 秒毎にデータ格納を行う．

なお，データ格納時のデータ収集停止時間は，TSC による測定結果から算出する．TSC による測定は，先ずデータ収集を停止するためにデータ収集ドライバが PMC のカウントアップを停止した直後に特定の物理 CPU（pCPU）の TSC 値を採取し，次に収集再開のために PMC のカウントアップを再開する直前に同じ pCPU の TSC 値を採取する．これらの TSC 値の差分を収集停止時間として算出する．

データ格納 1 回あたりのデータ収集停止時間を図 4.4 に示す．図 4.4 より，二つの問題がわかる．

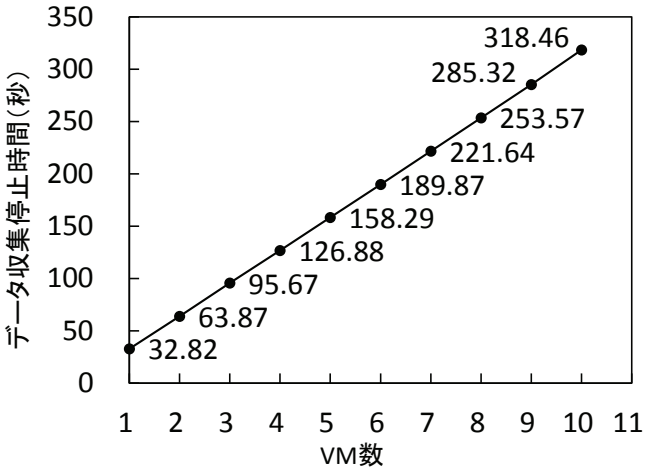

図 4.4 データ格納 1 回あたりのデータ収集停止時間

第一の問題は，一周期あたりのデータ収集停止時間が占める割合が大きいことである．例えば，1VM の場合で，60 秒のデータ収集時間に対してデータ収集停止時間は 32.82 秒となる．この時，格納周期あたりのデータ収集停止時間が占める割合をデータ収集のアンカバー率として式 4.1 で定義すると，データ収集のアンカバー率は約 35%である．

$$\text{アンカバー率} = \frac{\text{データ収集停止時間}}{\text{データ収集時間}+\text{データ収集停止時間}} \tag{4.1}$$

従って，このデータ収集停止時間を短縮する必要がある．この問題の要因は，VM 上で実行されているデータ格納処理プログラム y と jacobi 処理プロセスが，プロセススケジューリング上で同じ優先度で動作し，かつ同じ CPU を共有しているため，データ格納処理プログラム y が CPU を優先的に使用できずに動作できていない時間があるためである．そこで，同

じ CPU 上で他のアプリケーションプログラムが動作していても，データ格納処理プログラムが優先的に CPU を使用できるように対処する．詳細は，4.4.2 項で述べる．

第二の問題は，VM 数の増加に比例してデータ収集停止時間も増加することである．この問題の要因は，逐次的にデータ格納処理を行うためである．具体的には，既性能プロファイリングシステムでは，図 4.5 に示す通り，三種類の処理，即ち処理 1（VMM 上でのプログラム x によるプロセス情報 B_0 の格納処理），処理 2（VM 上でのプログラム y_i によるプロセス情報 B_i の格納処理（i=1,2,3, ...)），処理 3（VMM 上でのプログラム x による収集データ A_0 の格納処理）が逐次的に実行される．特に，複数の VM がある場合，処理 2 は，各 VM に対しても逐次的に実行される．そのために，VM 数の増加に比例して特に処理 2 が増加していくと考えられる．そこで，処理 1〜3 の各処理時間の実測結果を表 4.3 に示す．others は，図 4.4 のデータ収集停止時間から処理 1〜3 の処理時間を引いた時間である．この結果より，各 VM 数において処理 2 の処理時間が全処理時間の 95.2%〜99.6%と支配的であること，および VM 数の増加に比例して処理 2 の処理時間が増加していくことがわかる．そこで，特に処理 2 に対し，VM 数が増加しても処理時間ができるだけ変化せず一定になるように処理の並列化を図る．詳細については，4.4.3 項で述べる．

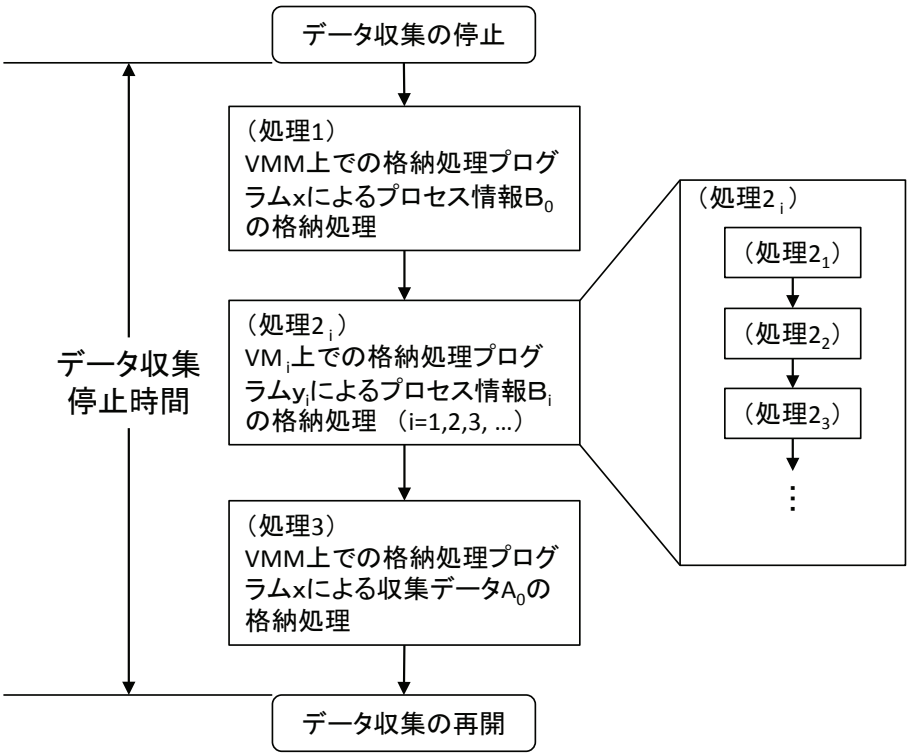

図 4.5 継続的な性能プロファイリングシステムでの逐次データ格納処理の流れ

表 4.3 データ格納処理の各処理時間

VM 数	処理時間 (秒)			
	処理 1	処理 2	処理 3	others
1	0.87	31.24	0.06	0.65
2	0.91	62.32	0.06	0.58
3	0.87	94.23	0.06	0.51
4	0.91	125.47	0.06	0.44
5	0.93	156.93	0.06	0.37
6	0.97	188.55	0.06	0.29
7	1.00	220.35	0.07	0.22
8	1.02	252.33	0.07	0.15
9	1.17	283.99	0.08	0.09
10	1.18	317.17	0.09	0.02

4.4.2 高優先度化による対処

データ格納処理プログラムの起動直後に優先度を高くなるように設定する．例えば，Linux では，プロセススケジューリング [74] における nice 値を使って，データ格納処理プログラムの処理の先頭で自身の優先度を最高位の-20 に設定する．これにより，同じ CPU 上で他のアプリケーションプログラムが動作していても，データ格納処理プログラムが優先的に CPU を使用できる．

4.4.3 並行動作化による対処

図 4.6 に示すように，各データ格納処理を並行動作させる．具体的には，先ず，データ収集の停止後，最初に動作するデータ格納処理プログラム x において，格納処理数に応じた子プロセスを生成する．処理 2 に対しては，VM 数に応じた数の子プロセスを生成する．従って，例えば VM 数が 2 の場合，処理 1 と処理 2_1 と処理 2_2 と処理 3 の合計 4 つの子プロセスを生成する．次に，各子プロセスに処理 1〜処理 3 までの各格納処理を実行させる．最後に，親プロセスのプログラム x が全ての格納処理の終了待ちをし，全ての格納処理が終了したらデータ収集を再開させる．

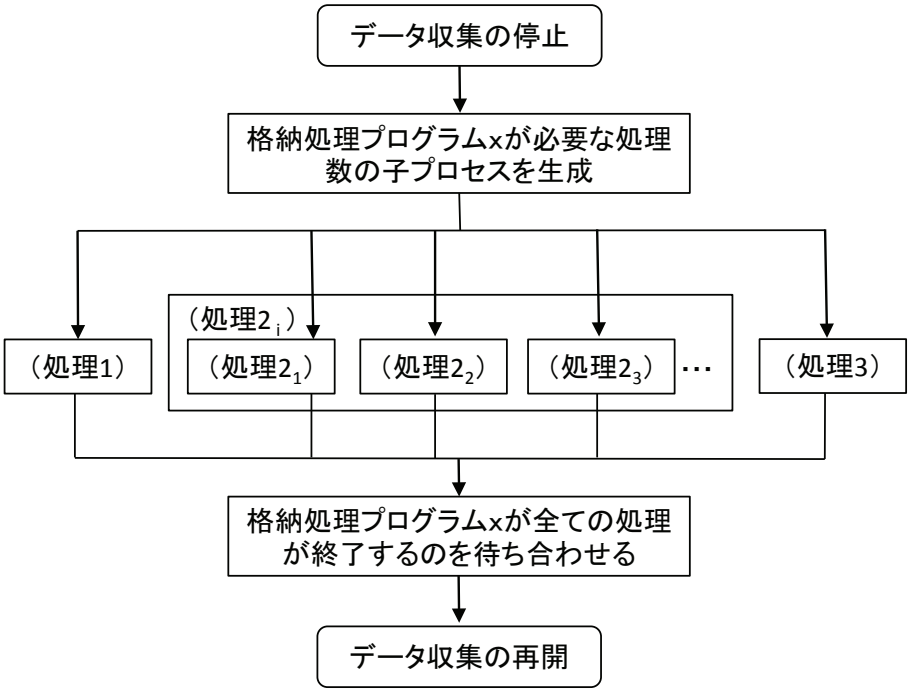

図 4.6 並行動作によるデータ格納処理の流れ

4.5 評価

4.5.1 データ格納を含めたオーバヘッド

定期的なデータ格納を含めた継続的な性能プロファイリングのオーバヘッドについて評価した．

測定は，4.4.1 項の表 4.2 に示す実験環境および測定条件で，vCPU 数は 2，VM 数は 1 に固定とし，データ収集時間を 20 秒，30 秒，60 秒，120 秒で jacobi 関数の処理時間の測定を行った．jacobi 関数の処理時間は以下の方法で測定した．

(1) vCPU 数 2 個として VM を 1 つ起動する．

(2) 定期的なデータ格納を伴う継続的な性能プロファイリング測定を開始する．

(3) VM 上で jacobi 関数を固定回数実行するプロセスを 1 つ起動する．

なお，固定回数は性能プロファイリング測定なしの時に約 10 分かかる回数を事前に決めておき，この固定回数の実行にかかる時間を処理時間として計測した．オーバヘッドは，性能プロファイリング測定なしの場合を基準にした処理時間の増加率から算出した．

先ず，処理時間の測定結果を図 4.7 に示す．データ収集時間が長いほど jacobi 関数の処理時間の増加が小さくなることがわかる．次に，処理時間の増加率（オーバヘッド）を図 4.8 に示す．図 4.8 を見ると，分散ファイルシステムによるオーバヘッドも含めて，データ収集時間が 30 秒以上の場合に提案手法のオーバヘッドが 1%以下となる．従って，サービス適用域（jacobi 処理のような CPU バウンドやメモリバウンドの処理）について，データ収集時間が 30 秒以上であれば，提案手法は，文献 [71] と同等なオーバヘッドを達成できている．

なお，物理計算機環境で実際に利用されている分散化された性能プロファイリングシステムの DCPI（DIGITAL Continuous Profiling Infrastructure）[49] ではオーバヘッドが 1%～3%，GWP（Google-Wide Profiling）[50] では数%であり，これらよりも低い負荷のプロファイラを VM 環境で作ることができた．

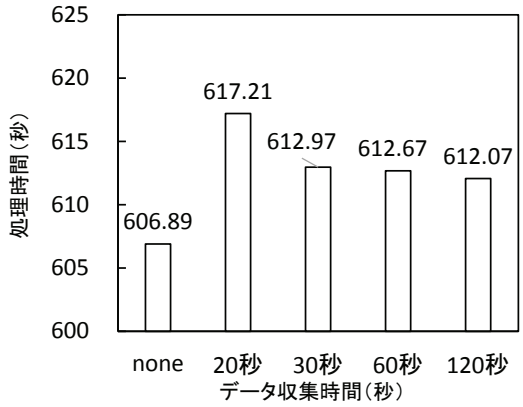

図 4.7 データ格納を含めた性能プロファイリング測定時の処理時間

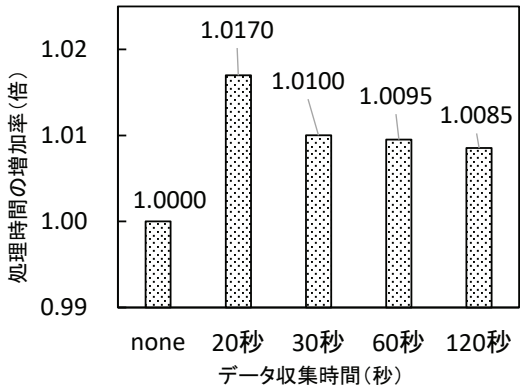

図 4.8 データ格納を含めた性能プロファイリングのオーバヘッド

4.5.2 データ格納時のデータ収集停止時間の短縮

4.4 節で提案した高優先度化と並行動作化の二つの対処について評価した．

先ず，各 VM 上で動作するデータ格納処理プログラム y に高優先度化を適用し，評価を行った．測定は，4.4.1 項と同じ環境と方法で，データ格納 1 回あたりのデータ収集停止時間を測定した．また，jacobi プロセスの nice 値を 0，同じ vCPU 上で動作するデータ格納処理プログラム y の nice 値を-20 に設定した．

次に，高優先度化に加え，各 VM 上で動作するデータ格納処理プログラム y に並行動作化も適用し，評価した．さらに，データ格納処理プログラム x も高優先度化した場合も評価した．

これらの測定結果を図 4.9 に示す．また，対処前との比較対象として，図 4.4 の結果を既性能プロファイリングとして示す．この結果より，以下の三つがいえる．

（1）先ず，高優先度化の効果として，1/4 近くにデータ収集停止時間が短縮できる．例えば，VM 数 1 の時に 28.1%に，VM 数 10 の時に 27.5%に短縮できた．これにより，4.4.1 項で説明した VM 数 1 の時のアンカバー率は 13%となり，35%から 1/3 近く短縮できた．しかし，高優先度化だけでは，まだ VM 数の増加に比例してデータ収集停止時間も大きくなる．

（2）高優先度化に加え，並行動作化の効果として，いずれの場合でも，データ収集停止時間の増加は非常に抑制されることがわかる．VM 数 1 の時に 23.6%に，VM 数 10 の時に 2.7%に短縮できた．

（3）データ格納処理プログラム x の高優先度化の効果はほとんど見られなかった．これは，今回の測定条件ではデータ格納処理プログラム x が別のアプリケーションと pCPU を共有せず，元々pCPU を専有使用できていたためである．

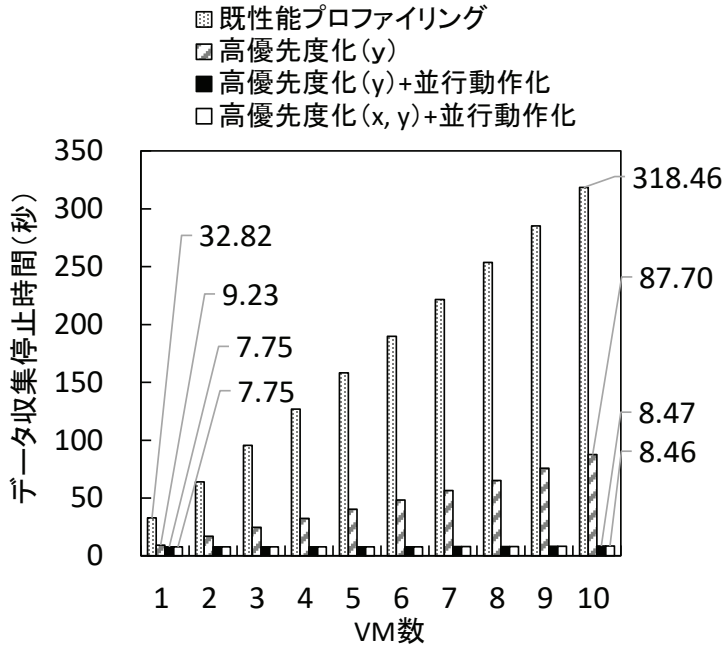

図 4.9 高優先度化と並行動作化によるデータ格納 1 回あたりのデータ収集停止時間

4.5.3 複数仮想 CPU でのデータ収集停止時間

ここでは，VM あたりの vCPU 数とデータ収集停止時間の関係を明らかにする．

測定環境と方法はこれまでと同じとし，データ格納処理プログラム y に高優先度化と並行動作化の対処を加える．また，vCPU 数が複数になった場合でも，本項の測定では VM 上で動作させる jacobi 関数の実行プロセスは 1 つのままとし，jacobi 関数の実行プロセスは特定の vCPU に固定するものとする．これにより，常に 1 つの vCPU は，データ格納処理プログラム y で占有使用可能となる．

図 4.10 に測定結果を示す．1vCPU/VM の結果は図 4.9 の高優先度化 (y)+並行動作化と同じものである．

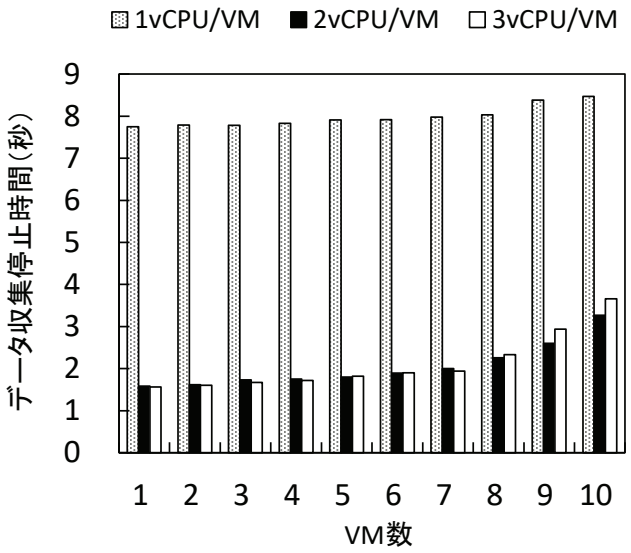

図 4.10 複数 vCPU 環境でのデータ格納 1 回あたりのデータ収集停止時間

この結果を見ると，先ず，2vCPU/VM と 3vCPU/VM のデータ収集停止時間の増加傾向はほぼ同じであることが分かる．例えば，VM 数の増加にともなうデータ収集停止時間の増分が，ともに，8VM から大きくなる．これは，動作 vCPU 数が pCPU 数を上回り始めることが理由として考えられる．具体的には，本評価では 2vCPU/VM でも 3vCPU/VM でも jacobi 関数実行プロセスとデータ格納処理プログラム y が使用する vCPU は最大二つまでである．対して，pCPU 数は 14 である．従って，7VM で動作 vCPU 数が 14 となり，pCPU の使用が飽和状態となる．8VM 以上では，pCPU が使えず待たされる vCPU が出始める．

このように，2vCPU/VM と 3vCPU/VM では同じ傾向がある一方で，8VM 以上では値が少し乖離し始める．これは，KVM では vCPU 数が増えると，VM 上のプロセス数が増えて定期

的に格納すべきプロセス情報のデータ量が増えることが理由として考えられる．具体的には，1vCPU 増えると，データ格納 1 回あたり格納すべきデータ量が 230KB/VM〜240KB/VM 増える．

次に，2vCPU/VM および 3vCPU/VM の場合と 1vCPU/VM のデータ収集停止時間を比較する．1vCPU/VM に対し，2vCPU/VM および 3vCPU/VM では，例えば，図 4.10 を見ると，VM 数 10 の時に，1vCPU/VM で 8.47 秒だったデータ収集停止時間が，2vCPU/VM であれば 3.66 秒にさらに短縮でき，アンカバー率も 5.75%となる．

この主たる要因の一つは利用可能な CPU 資源の違いである．この測定では，データ格納処理プログラムは jacobi 関数の実行プロセスより高優先度化されているが，1vCPU/VM の場合は CPU を占有的に使い続けられる訳ではなく，同じ vCPU 上に共存している jacobi 関数実行プロセスにも vCPU 使用がスケジューリングされ，その分データ格納処理が動けない時間が発生する．対して，2vCPU/VM や 3vCPU/VM の場合は，データ格納処理プログラムは一つの vCPU を占有して使い続けることができるため，その分データ格納処理が早く終了しデータ収集停止時間が短くなる．また，メモリ資源に関する競合もデータ収集停止時間に影響を与える要因として推察される．2vCPU/VM や 3vCPU/VM の場合と比べて，1vCPU/VM の場合，一つの同じ vCPU 上でデータ格納処理プログラムと jacobi 関数が切り替わって動作するのでキャッシュや TLB（Translation Lookaside Buffer）のヒット率が低下するためと推察する．従って，キャッシュミスや TLB ミスのためにデータ格納処理が遅延し，データ収集停止時間が長くなると推察できる．

以上のことから，本提案手法は，VM あたりの vCPU 数が 1 の場合に比べ 2 以上の場合は，データ収集停止時間が小さくなり有効である．なお，1vCPU/VM のデータ収集停止時間は，pCPU が不足していなくても VM 数 8 と 9 の間で増分が大きくなっている．この増分の要因は，VM 数が多い時に CPU 資源，システムバス帯域，ネットワーク帯域などの複数の要因が相互に影響することによるものと推察され，その詳細な分析は今後の課題としたい．

4.5.4 複数アプリ動作時のデータ収集停止時間

ここでは，VM あたりの動作アプリケーションプログラム数とデータ収集停止時間の関係を明らかにする．

本評価では，各 VM の vCPU 数は，3vCPU のみとし，複数ある jacobi 関数の実行プロセスの特定 vCPU への固定割り付けは，いずれも行わないとする．その他の測定条件と方法は，4.5.3 項と同じである．

測定結果を図 4.11 に示す．図 4.11 から，二つのグラフには一つ，もしくは二つの変化点があることがわかる．先ず，jacobi 関数を 2 プロセス動作させている場合，データ収集停止時間は，1VM から 6VM では変わりなく，6VM から 7VM ではデータ収集時間（60 秒）以上となり，7VM から 10VM は変わらない．これは，動作している vCPU 数が 12 から 14 になる所で，jacobi 関数実行プロセスとデータ格納処理プログラム y が動作する vCPU 数に対して pCPU 数が飽和し始める所である．

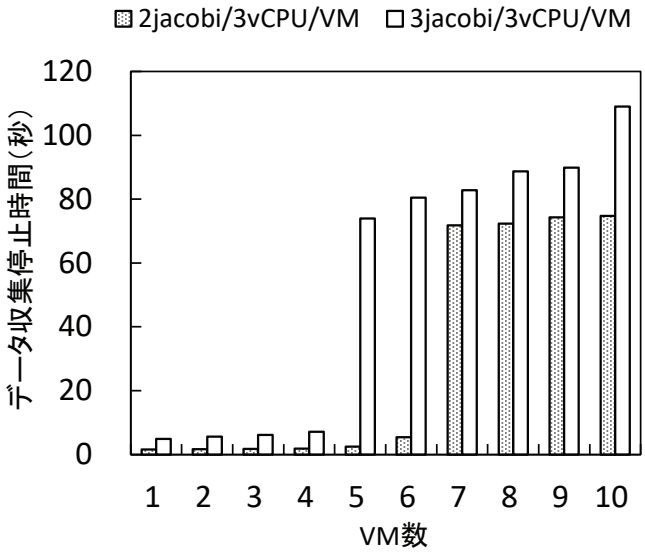

図 4.11 複数アプリ負荷によるデータ格納 1 回あたりのデータ収集停止時間

一方，jacobi 関数を 3 プロセス動作させている場合は，二か所の変化点がある．最初の変化点は，VM 数が 4 から 5 の所である．これは，動作している vCPU 数が 12 から 15 になるところで，vCPU 数が pCPU 数を超えるところと一致する．もう一つの変化点は VM 数が 9 から 10 の所である．これは，動作している vCPU 数が 27 から 30 になるところで，pCPU を 2vCPU で使用できていた場合から 3vCPU で使用する場合への遷移点である．

以上のことから，本提案手法は，VM あたりの動作アプリケーションプログラム数が vCPU 数以下であり，かつ各 VM の動作アプリケーションプログラム数の総和が pCPU 数以下であれば，データ収集停止時間を 7.10 秒以下にでき有効である．

4.6 関連研究

性能プロファイリングデータの継続収集の関連研究として，DCPI [49] と GWP [50] がある．先ず，共通の特徴として，いずれもデータ収集を継続して行い，データベースでデータを管理し，ユーザからの要求に応じて必要な解析結果を提供するというサービスシステムとなっている．しかし，いずれの文献も，本章で議論したような，データ格納方法の詳細や，分散システムでの定期的なデータ格納時の問題については論じられていない．

次に，個々の特徴として，DCPI は，1990 年代に Alpha プロセッサと DIGITAL Unix をベースとしたシステム向けに Digital Equipment Corporation（DEC）社が開発した性能プロファイリングシステムである．PMC のカウンタオーバフロー割込みを契機に用いたデータ収集を行う．基本仕様として，データ収集時間は 10 分で，収集毎にメモリ上の収集データをデータベースへ格納する．なお，DCPI のサブセット機能として Oprofile が派生している．一方，GWP は，2000 年代に Google 社が DCPI を参考にして IA サーバ向けに開発した性能プロファイリングシステムである．GWP は，Oprofile をベースに用いた自社データセンタ用の性能プロファイリングシステムで，文献 [51] では，2 万台以上のマシンを対象にした GWP の使用が報告されている．

以上に示したように，DCPI と GWP は，物理計算機での継続的な性能プロファイリングシステムである．これに対し，本提案手法は，VM 向けの継続的な性能プロファイリングシステムを実現する手法であり，文献 [71] と同様に jacobi 処理のような CPU バウンドやメモリバウンドの処理を観測対象とする．

4.7 まとめ

本章では，継続的な性能プロファイリングを行う構成として，VM を利用した性能プロファイリングシステム [71] の分散化を述べた．また，収集データを格納する際に発生するデータ収集の停止時間を短縮する方法として，データ格納処理の高優先度化と並行動作化を述べた．

評価では，主に CPU の演算性能やメモリのバンド幅に処理性能が依存する jacobi 関数を用いて，データ格納も含めた性能プロファイリングによるオーバヘッドや高優先度化と並行動作化によるデータ収集停止時間の短縮効果について評価した．先ず，データ格納を含めた性能プロファイリングシステムによるオーバヘッド評価では，データ収集時間 30 秒以上で 1%以下の低オーバヘッドとなることを示した．

次に，データ収集停止時間の短縮効果の評価では，高優先度化により，データ格納プログラムとアプリケーションが同じ vCPU を共有している場合の問題に対してアンカバー率が

1/3 ほどに短縮できることを示した．さらに，並行動作化により，性能プロファイリング対象の VM 数が増加してもデータ収集停止時間を一定に保つ効果があり，10VM の時の 60 秒毎のデータ収集停止時間が 318.46 秒から 8.47 秒で 2.7%に短縮できることを示した．

この 8.47 秒のデータ収集停止は，継続的に動作する計算機の性能異常を検出する課題に対し許容可能である．例えば，パブリッククラウドの個々の VM ユーザに対しては，Amazon Web Services [75] が提供する Amazon CloudWatch サービス [76] のように，VM ユーザが予め定めたアプリケーションレベルの 1〜5 種類の非常に少ない性能統計情報を秒単位で Amazon CloudWatch に発行し，最短で閾値による 10 秒単位のアラーム発報を受けることができる．しかし，パブリッククラウドおよびプライベートクラウドのサービス運用では，VMM 上で 1,000 種類以上の非常に多い性能統計情報を収集し解析処理に時間がかかる．例えば，VMware によるクラウド基盤 [77] では，性能異常を検出する契機は短くても 20 分間隔である．これに対して，VMM 上での 68.47 秒毎の検出契機は短く，継続的に動作する計算機の性能異常検出を可能にする．

第 5 章

分散化した性能プロファイリングシステムの解析処理時間の短縮

5.1　概要

計算機の性能低下異常を検出し，その要因処理を特定するために，継続的に性能プロファイリングを行う構成として，仮想計算機（VM：Virtual Machine）を利用した性能プロファイリングシステムの分散化が提案され，データ格納を含めたオーバヘッドやデータ収集停止時間の評価結果が報告されている．一方，継続的に性能プロファイリングを行うためには，データ収集やデータ格納だけではなく，解析処理も含めて連続実行できる必要がある．本章では，先ず，分散化した性能プロファイリングシステムで継続的に性能プロファイリングを行うために必要な条件として，データ収集，データ格納，解析の各処理時間の関係条件を示し，次に，この条件を満たすため，解析処理時間の短縮法として並行動作処理方式を述べ，評価結果を報告する．

5.2　既存の解析処理

5.2.1　分散化した性能プロファイリングシステム

計算機の性能低下異常を検出し，その要因処理を特定するために，継続的に性能プロファイリングを行う構成として，VM を利用した性能プロファイリングシステムの分散化 [78] が提案されている．図 5.1 に，分散化した性能プロファイリングシステムの処理プログラムの

構成例を示す．処理プログラムのうち，ユーザコマンドの解析処理機能だけ性能プロファイリング対象となる被測定計算機から解析処理計算機に分離する．また，データの格納先は解析処理計算機のディスクとする．被測定計算機から解析処理計算機のリモートディスクへの格納は分散ファイルシステムを利用する．処理の流れを以下に説明する．

（1）ユーザから指定されたデータ収集周期とデータ収集時間で，メモリバッファ上に，命令アドレスやプロセス ID などを含むプログラム動作情報のデータ収集を行う．このデータを収集データと呼ぶ．

（2）収集後に，VM モニタ（VMM）上から収集データと VMM 上および VM 上からプロセス情報をディスクに格納する．
以降,（1）と（2）の処理を繰り返す．

（3）格納データを基に定期的な解析処理を行う．

ここで，性能プロファイリングデータにノイズ（格納処理の影響）が載ることを防ぐために，処理（2）のデータ格納処理は収集毎ではなくまとめて定期的に行う．また，データ格納処理中は処理（1）のデータ収集を停止させる．この停止時間をデータ収集停止時間と呼ぶ．さらに，データ収集時間とデータ収集停止時間の和の時間を格納周期とする．

また，処理（2）で定期的に格納するデータの他に解析処理に必要なオブジェクトファイルやカーネルのシンボルマップ，および実行条件や環境情報など付帯情報は静的な情報であり別途入手する．これにより，サービスへの影響を抑制する．

以上に述べた VM を利用した性能プロファイリングシステムの分散化と処理（1）（2）については，文献 [78] でデータ格納を含めたプロファイリングシステムによるオーバヘッドとデータ収集停止時間の評価結果が報告されている．本章では，解析処理も含めて一連の処理を連続実行できるようにするために，図 5.1 の破線囲み内の環境と処理（3）に対する評価と解析処理時間の短縮法について述べる．具体的には，先ず，既存の解析処理の処理流れと連続実行のための各処理時間の関係条件について 5.2.2 項，既存の解析処理時間の評価について 5.2.3 項で述べる．次に，既存の解析処理の問題と対処について 5.3 節，対処手法の評価について 5.4 節で述べる．

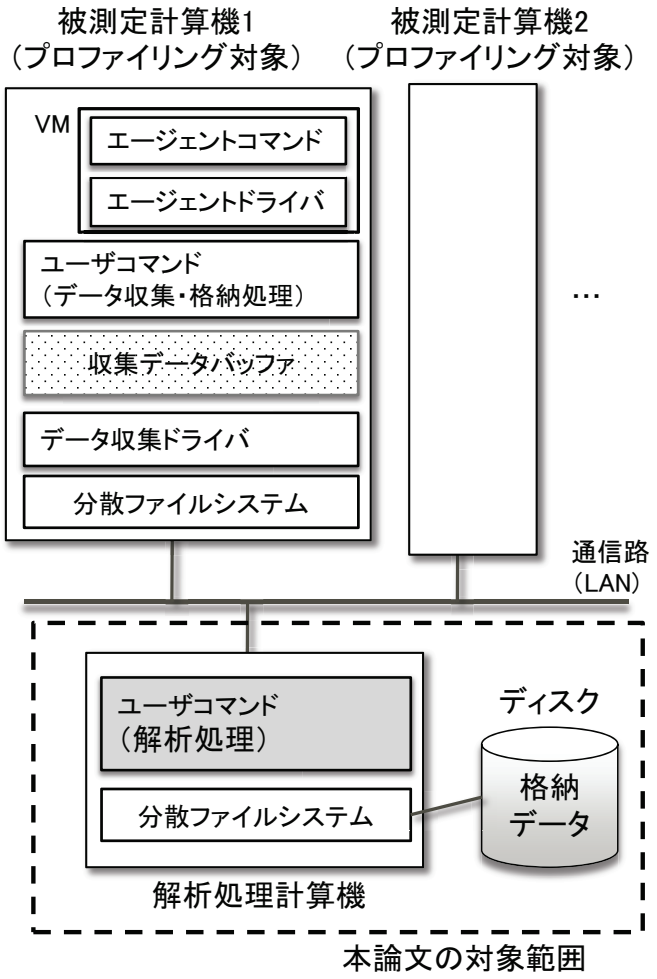

図　5.1　分散化した性能プロファイリングシステムの構成例

5.2.2　既存の解析処理流れと連続実行条件

図 5.2 に，VM を利用した性能プロファイリングシステムでの既存の解析処理流れを示す．三種類の処理が逐次的に実行される．処理 1 は，VMM 上の動作プログラムに対する解析処理で，既存のプロファイリングシステムの解析処理と同じである．具体的には，文献 [71] 2.1 節で説明されているシンボル解決処理後に，シンボル単位で頻度集計を行い，頻度順で結果を出力する．なお，ここで言うシンボルはプログラムの関数名のことである．処理 2 では，文献 [71] 3.4 節と 3.5 節および文献 [78] 2.3 節で述べられている手法を用いて，VMM 上の収集データから VM の収集データを VM 単位で疑似的に生成する．処理 3 は，処理 2 で生成された各 VM の疑似収集データを用いて，処理 1 と同じ解析処理を VM 毎に逐次的に行う．なお，格納データの読込み開始から処理 3 の全 VM の解析処理が終了するまでの時間を解析処理時間と呼ぶ．

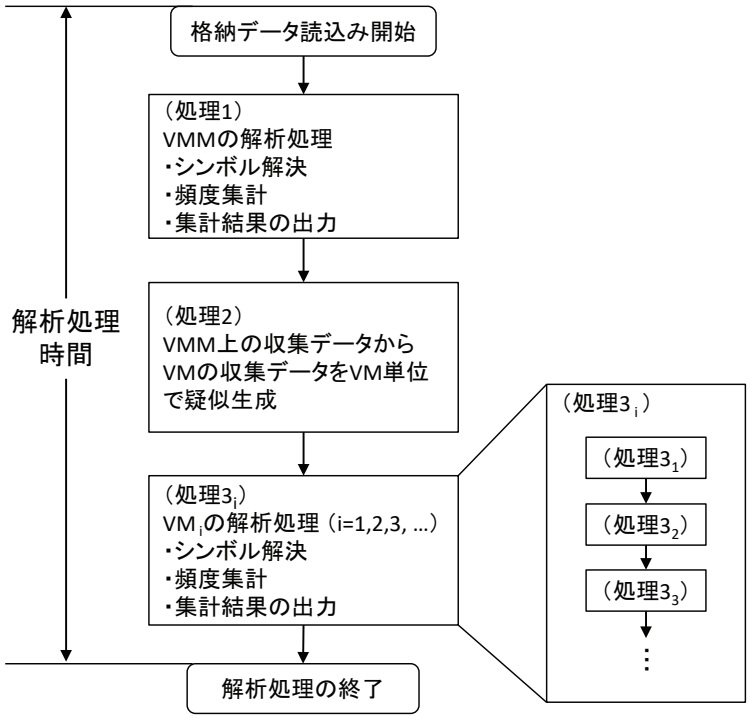

図 5.2 仮想計算機を利用した性能プロファイリングシステムにおける既存の解析処理の流れ

分散化した性能プロファイリングシステムにおいて，継続的な性能プロファイリングを行うために，データ格納時間 t_0，データ収集時間 t_1，および解析処理時間 t_2 が満たすべき関係は，図 5.3 のような関係である．

この関係は，1 台の解析処理計算機が解析対象とする VMM 数を n とすると，式 5.1 である．

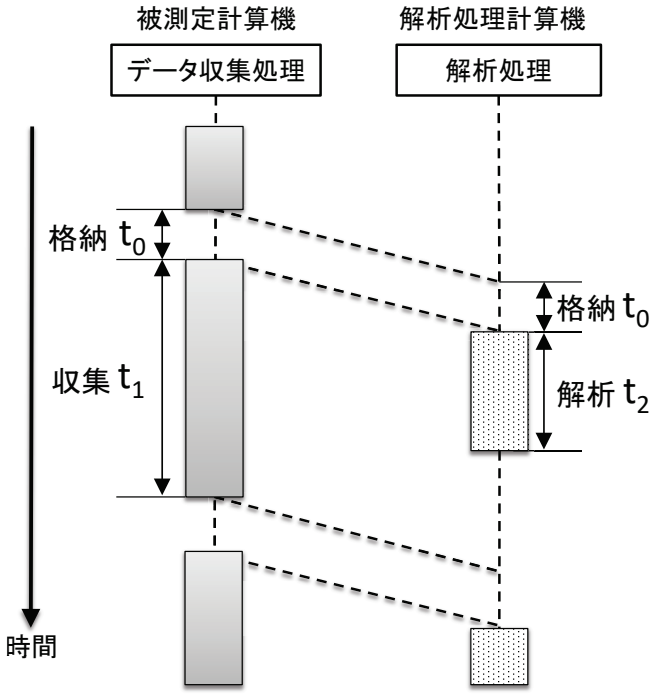

図 5.3 継続的な性能プロファイリング時のデータ収集，データ格納，および解析の各処理時間の関係

$$t_0 + t_1 > (t_0 + t_2) \times n \tag{5.1}$$

5.2.3 既存の解析処理時間評価

複数の VM がある場合，図 5.2 の処理 3 が各 VM に対して逐次的に実行されるため，VM 数の増加に比例して解析処理時間の増加が懸念される．そこで，VM 数の変化に対する既存の解析処理時間を実測し評価する．

実験環境と解析処理に用いる収集データ条件を表 5.1 に示す．収集データの収集環境は，文献 [78] の実験環境と同条件に合わせた．データ収集時に各 VM 上では，文献 [78] と同じ姫野ベンチマーク [73] に含まれる jacobi 関数を繰り返し実行していた．文献 [78] より，データ格納も含めたオーバヘッドとデータ収集のカバレッジの観点から，この環境における最適なデータ収集時間は 60 秒となるので，本評価で用いる収集データも 60 秒で収集したデータを用いる．解析に用いる収集データは，解析処理計算機のローカルディスクに予め格納されているものとし，図 5.2 に示すようにローカルディスクからデータを読込み開始するところから解析処理時間の測定を開始する．

また，継続的に性能プロファイリングを行うための条件として，式 5.1 において，n=1，データ収集時間 t_1=60 とすると，解析処理時間 t_2 は 60 秒以下となる必要があり，これを本論文の解析処理時間の目標とする．

表 5.1 実験環境と解析処理に用いる収集データ条件

ホスト環境（被測定計算機，解析処理計算機）	
CPU	Intel Xeon E5-2697v3 2.60 GHz 14 コア× 1
Memory	24 GB
LAN	1 Gbps Ethernet
Disk	HDD SATA 7200 rpm 250 GB
OS	CentOS Linux 7.2.1511 64 bit （kernel 3.10.0-327.28.2.el7.x86_64）
VMM	qemu-kvm-2.3.0-31.el7.16.1
ゲスト環境	
VM 数	1, 2, 3, ..., 10
vCPU 数	1
Memory	4 GB
OS	CentOS Linux 7.2.1511 64 bit （kernel 3.10.0-327.28.2.el7.x86_64）
収集データ条件	
データ収集周期	1 ミリ秒
データ収集時間	60 秒
1VM 時データサイズ	86 MB

図 5.4 に，VM 数 1 の時の収集データから VM 数 10 の時の収集データまで，各 VM 数の収集データの解析処理時間の測定結果を示す．この結果を見ると，VM 数が 1〜7 までならそのままで目標の 60 秒以下を満たせることがわかる．しかし，VM 数 8 以上で継続的に性能プロファイリングが行えないこととなる．

次に，同じ収集データを SSD(NVMe PCI3.0x4) 上において，図 5.4 の場合と同様に，解析処理した場合の処理時間を図 5.5 に示す．この結果より，ディスクを SSD に変えただけ

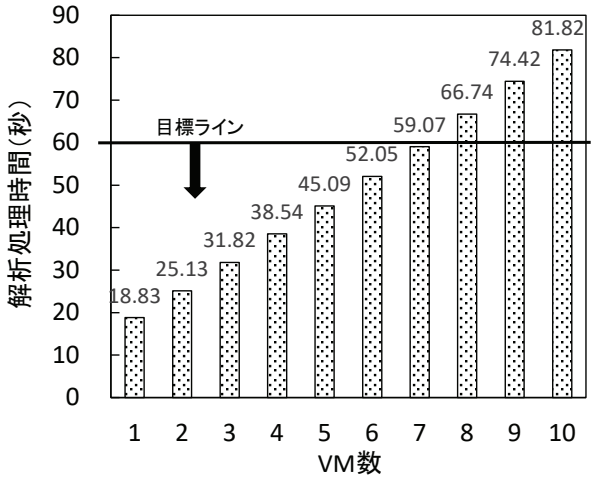

図　5.4　仮想計算機を利用した性能プロファイリングシステムにおける解析処理時間

では，解析処理時間は短縮されないことがわかる．即ち，ディスクがボトルネックにはなっていないことがわかる．理由は，収集データなど解析に利用するデータが全てサイズ的に計算機の物理メモリに載るためと考えられる．例えば，本測定環境と条件で 10VM の時に格納した 1 回分の収集データとそのほかの解析に利用するオブジェクトファイルなども含めた全データサイズは 833MB となり，解析処理計算機の物理メモリ 24GB 上ですべて処理されうることになる．

ディスクによる性能差はないので，以降の解析処理時間の測定でも，前段研究の文献 [78] と同じく，ディスクは HDD を使用することとする．

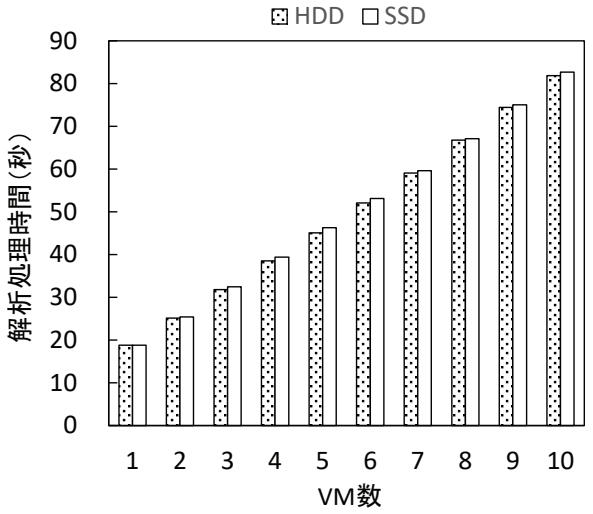

図　5.5　解析に使用するデータが HDD にある場合と SSD にある場合の解析処理時間の比較

5.3　解析処理時間の短縮法

5.3.1　既存の解析処理の問題

VM 数の増加に対する既存の解析処理の課題を明らかにするため，図 5.2 の処理 1～3 の処理毎の実行時間を実測し分析した．測定は，5.2.3 項と同じ環境と方法で行った．図 5.6 に，処理 1～3 の各処理時間の測定結果を積み上げ棒グラフで示す．この結果から，処理 2 と処理 3 の処理時間が VM 数に比例して増加することがわかる．特に，VM 数 10 の時，処理 2 は解析処理時間全体（81.82 秒）の 30.18%，処理 3 は 42.06%となり，処理 3 の占める割合がもっとも大きくなることがわかる．

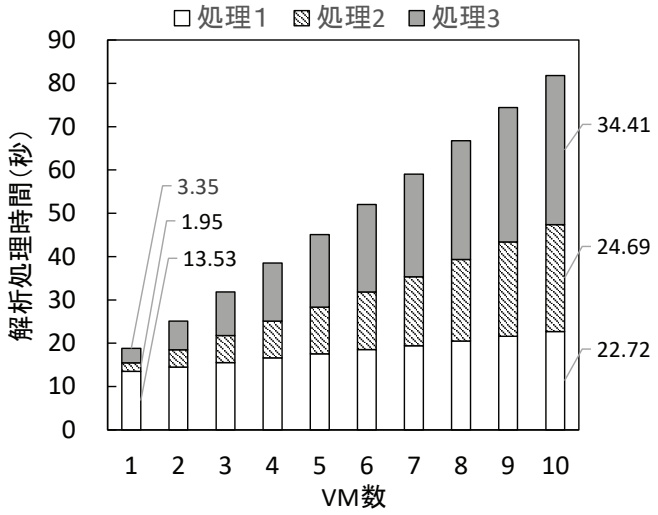

図　5.6　仮想計算機を利用した性能プロファイリングシステムにおける解析処理の内訳時間

5.3.2　並行動作化による対処

処理 3 に対し，VM 数が増加しても処理時間が増加せず一定になるように処理の並行動作化を提案する．具体的な対処としては，図 5.7 に示すように，処理 3 を VM 単位で並行動作させる．先ず，処理 3 の冒頭で，VM 数に応じた数の子プロセスを生成する．次に，各子プロセスに 1VM 分の解析処理を実行させる．最後に，親プロセスが全ての処理 3 の終了待ちをし，全ての処理 3 が終了したら解析処理を終了する．

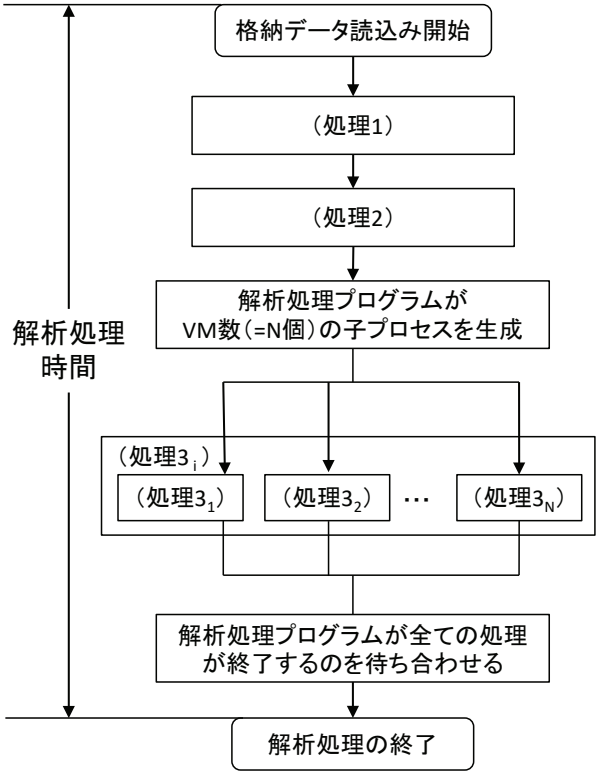

図　5.7　並行動作による解析処理の流れ

5.4　評価

5.4.1　並行動作化による解析処理時間の短縮

本項では，図 5.7 に示す処理 3 の並行動作化の効果を評価した．解析処理時間の測定は，5.2.3 項と同じ環境と方法で行った．

先ず，並行動作化を適用する前の既性能プロファイリングによる解析処理時間と並行動作化を適用した解析処理時間の測定結果を図 5.8 に示す．これらの結果を比較することにより，既性能プロファイリングでの解析処理では VM 数が 8～10 の場合は目標が達成できなかったが，処理 3 の並行動作化で VM 数が 8～10 の場合でも目標を達成できるようになり，並行動作化の効果を確認した．

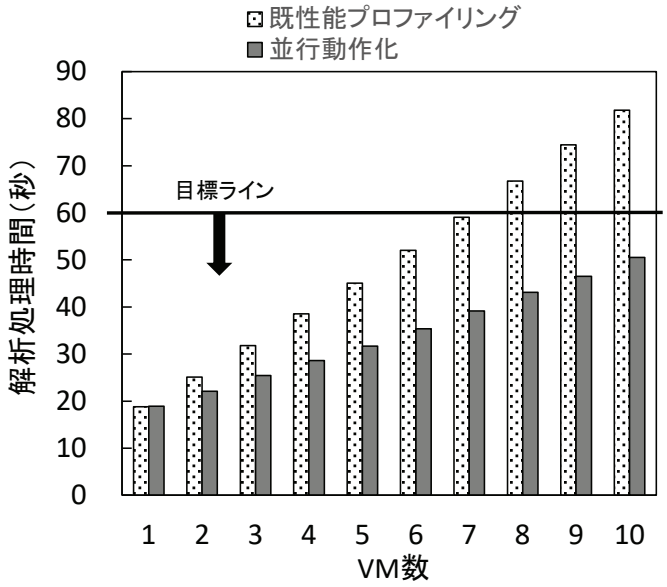

図　5.8　既性能プロファイリングと並行動作化の解析処理時間の比較

さらに，処理 3 の並行動作化の効果の裏付けとして，処理 3 に並行動作化を適用した解析処理の処理 1～3 の処理毎の測定結果を図 5.9 に示す．これを見ると，VM 数 1 の時と VM 数 10 の時の処理 3 の処理時間はそれぞれ 3.38 秒と 3.63 秒となり，VM 数が 10 倍に増加しても処理時間は 1.07 倍の増加に抑えられほぼ一定であることがわかる．即ち，5.3.2 項で述べた課題は達成できたといえる．

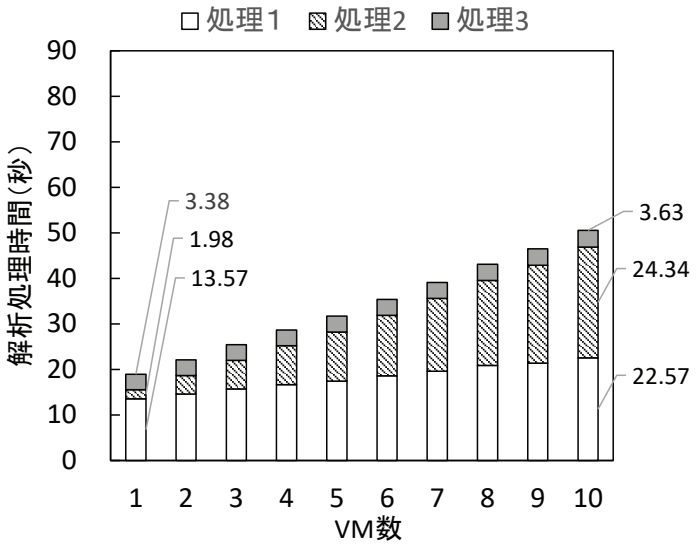

図　5.9　並行動作化した解析処理の処理時間の内訳

5.4.2　複数 VMM 環境を想定した解析処理時間の評価

5.4.1 項の評価では，解析対象となる被測定計算機が 1 台だった．一方，継続的に性能プロファイリングを行う場合は，1 台の解析処理計算機で複数の被測定計算機，即ち複数の VMM 環境を対象とする必要がある．そこで，本項では，継続的に性能プロファイリングを行うために，本提案手法で 1 台の解析処理計算機あたり何台の VMM まで継続的に処理することができるか評価した．

解析処理時間は，5.2.3 項と同じ環境と条件において，以下の方法で測定した．

(1) 先ず，1VMM 分のデータセットを用意する．具体的には，1 台の VMM 上の m 台の VM の各 VM 上で jacobi 関数を実行中に，VMM 上で 60 秒間のデータ収集を 1 回だけ行い，データ収集終了後に収集データをディスクに格納する．格納データには，VMM での収集データ，および VMM と各 VM のプロセス情報，オブジェクトファイル，カーネルのシンボルマップ，付帯情報が含まれる．

(2) 用意した 1 VMM 分のデータセットを図 5.10 に示すように異なる n 個のディレクトリにコピーし配置することにより，nVMM 分のデータセットとする．

(3) VMM 数 n を変化させながら，解析処理時間を測定する．先ず，時間測定を開始する．次に，VMM 数を n とすると， n 個の子プロセスを生成する．そして，生成した各子プロセスで並行して 1VMM 分のデータセットを解析処理する．最後に，全ての子プロ

セスが終了したら時間測定を終了する．なお，各子プロセスはそれぞれ図 5.7 に示す提案手法で解析処理を行う．

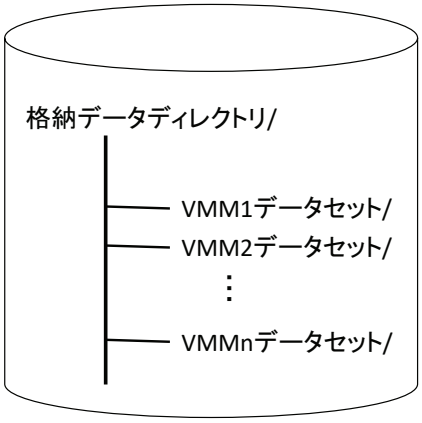

図 5.10 複数 VMM 分のデータセット配置

図 5.11 に，1VMM あたりの VM 数 m=5 の場合の測定結果を示す．これを見ると，表 5.1 に示す計算機環境では，1VMM あたり 5VM の場合，本提案手法で，1 台の解析処理計算機あたり 16VMM，即ち被測定計算機 16 台を継続的に性能プロファイリングできるといえる．図 5.12 に，1VMM あたりの VM 数 m=10 の場合の測定結果を示す．これを見ると，1VMM あたり 10VM の場合は，1 台の解析処理計算機で 4VMM，即ち被測定計算機 4 台を 1 台の解析処理計算機で継続的に性能プロファイリングできるといえる．これらの結果は，継続的な性能プロファイリングシステムを設計する際の台数設計の基準にできる．

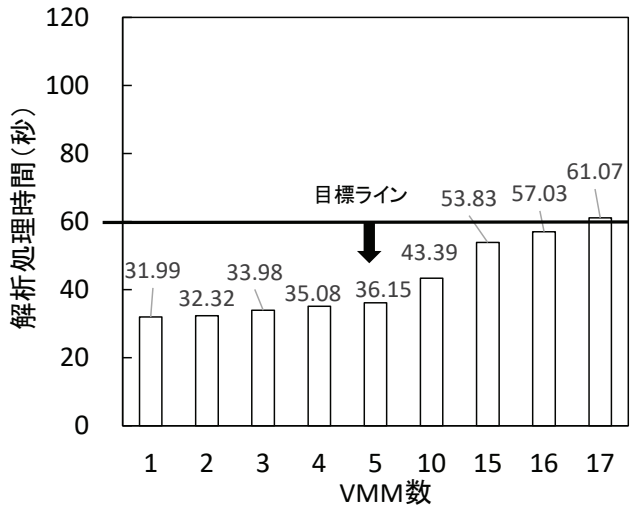

図　5.11　1VMM あたり 5VM の場合の VMM 数に対する解析処理時間

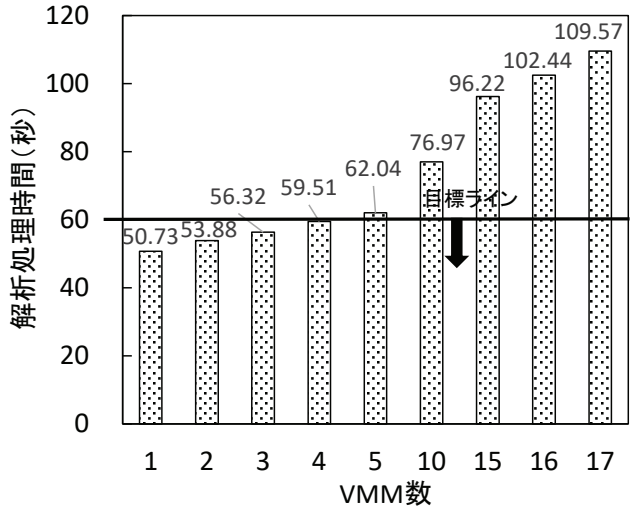

図　5.12　1VMM あたり 10VM の場合の VMM 数に対する解析処理時間

5.5　関連研究

継続的な性能プロファイリングの関連研究として，DCPI（DIGITAL Continuous Profiling Infrastructure）[49] と GWP（Google-Wide Profiling）[50, 51] がある．これらは，CPU が備える性能監視カウンタ（PMC）のカウンタオーバフロー割込みベースのデータ収集を行い，ネットワーク経由でリモートのディスクへデータ格納するという点においては，本研究のデータ収集とデータ格納と同じである．一方，いずれもデータ収集とデータ格納を継続して行い，データベースで収集データを管理し，ユーザからの要求に応じて必要なプログラムの収集データの解析処理を行い，解析結果をユーザに提供するというサービスシステムとなっている．従って，解析まで含めて連続実行しているわけではない点は，本研究手法とは異なる．また，定量的な評価としては，DCPI でのデータ収集時のオーバヘッドが 1%-3%と述べられているのみである．GWP では，オーバヘッドが数%以下におさまるように，予備測定を行いデータ収集条件を調整する．ともに解析の処理時間の定量評価はない．しかし，GWP では，解析処理の中でシンボル解決処理が時間がかかり課題の一つとして述べられている．例えば，1 日分の収集データのシンボル解決には数週間を要する．この対処として，数百台の解析処理計算機を用意した MapReduce による分散バッチ処理を行っている．

クラウド環境での，継続的な性能監視ツールとして，Amazon CloudWatch [76] と OpenStack Ceilometer [79] がある．ともに，性能指標値の収集を行い，指標値の提示や閾値によるアラーム発砲などを行うツールである．しかし，これらのツールにより，動作プログラムの性能低下異常の要因処理を特定することはできない．

PMC と同じく CPU が備える性能監視機能を用いたプロファイリング手法の関連研究として，Bitzes らの研究 [42] と Sharma らの研究 [80] がある．Bitzes ら [42] は，Linux perf [34] を使って，Intel CPU が備える PMC 機能の Counting mode，Sampling mode，および PEBS（Precise Event Based Sampling）機能，LBR（Last Branch Record）機能の各機能によるデータ収集時のオーバヘッドを調査し報告している．ただし，本研究でも用いている性能プロファイリングのデータ収集でもっともよく利用されている PMC を使ったタイムベース Sampling と比較できる結果はない．また，解析処理に関する評価もない．しかし，多種な性能イベントによるデータ収集やイベント切り替えによるデータ収集なども調査されている．Sharma ら [80] は，Intel の Broadwell マイクロアーキテクチャ世代以降の CPU が備える最新の性能監視機能である Intel PT（Processor Trace）機能を使った，仮想計算機上の動作プログラムのトレース手法に関する研究である．これまでトレースデータ収集には，例えば，3.6%-28.7%のオーバヘッドがかかっていたが，Intel PT により 1%以下に抑えることができる．解析処理時間の評価は行われていない．

5.6　まとめ

本章では，先ず，継続的に性能プロファイリングを行うために必要な条件として，分散化した性能プロファイリングシステムにおける，データ収集，データ格納，解析の各処理時間の関係条件を示した．さらに，各処理時間の関係条件を満たすため，既存の解析処理の問題点を述べ，対処として解析処理の並行動作化を提案した．最後に，並行動作化による効果や，1 台の解析処理計算機で複数の被測定計算機を対象とする場合を想定した解析処理時間について評価した．その結果，本提案手法で，表 5.1 に示す計算機環境では，1 台の解析処理計算機あたり，5VM が動作している被測定計算機 16 台，10VM が動作している被測定計算機なら 4 台を継続的に性能プロファイリングできることを示した．

第 6 章

結論

本論文では，VM 環境における継続的な性能プロファイリングシステムの実現手法の確立について述べた．本論文で確立した性能プロファイリングシステムは，クラウドの性能低下異常を検出し，その要因処理を特定することで，クラウドの保守性向上を支えるシステムである．

第 1 章では，VM における性能プロファイリングシステムの研究に関し，その背景となるクラウドと VM，および性能プロファイリングシステムについて述べた．さらに，性能プロファイリングシステムに関する研究の状況ならびに本論文の目的と課題について述べた．

第 2 章では，VM における性能プロファイリングシステムのデータ収集オーバヘッドの削除に向けた問題点と課題を提示し，VMM で行う性能プロファイリングシステムを提案した．また，VMM で行う性能プロファイリングシステムを実装し，評価を行った．評価では，先ず，VM においても，期待通りの性能プロファイリング結果が得られることを示した．次に，既存手法と比較してデータ収集オーバヘッドが約 70%削減できることを示し，さらに，許容可能なオーバヘッドでのデータ収集周期を提示した．最後に，本提案手法を用いた性能改善事例を述べ，VMM で行う性能プロファイリングシステムの有効性を示した．

第 3 章では，物理 CPU 数と仮想 CPU 数の違いにより発生する VM のスチール時間の定義や問題点を述べ，物理 CPU 数と仮想 CPU 数の違いを考慮した測定精度の向上手法を提案した．また，VM では，アプリケーション単位でスチール時間を把握する手段がないため，アプリケーションの正確な実行時間が求められない問題があり，このスチール時間を含む見かけ上の実行時間の補正法の提案を行った．評価では，これらの提案手法を実装し，先ず，既存手法と比較して測定精度の向上を示した．さらに，VM のアプリケーションの実行時間を期待通り補正できることを示した．

第 4 章では，継続的な性能プロファイリングを行う構成として，VM を利用した性能プロファイリングシステムの分散化を述べた．また，収集データを格納する際に発生するデータ収集の停止時間を短縮する方法として，データ格納処理の高優先度化と並行動作化を提案した．評価では，性能プロファイリングシステムのデータ収集およびデータ格納も含めたオーバヘッドや高優先度化と並行動作化によるデータ収集停止時間の短縮効果を示した．先ず，データ収集とデータ格納によるオーバヘッドの評価では，データ収集時間 30 秒以上で 1%以下の低オーバヘッドとなることを示した．次に，データ収集停止時間の短縮効果の評価では，高優先度化により，データ格納プログラムとアプリケーションが同じ仮想 CPU を共有している場合の問題に対してアンカバー率が 1/3 ほどに短縮できることを示した．さらに，並行動作化により，性能プロファイリング対象の VM 数が増加してもデータ収集停止時間を一定に保つ効果があり，10VM の時の 60 秒毎のデータ収集停止時間が 318.46 秒から 8.47 秒で 2.7%に短縮できることを示した．

第 5 章では，先ず，継続的に性能プロファイリングを行うために必要な条件として，分散化した性能プロファイリングシステムにおける，データ収集時間，データ収集停止時間（データ格納時間），解析処理時間の関係条件を示した．次に，この関係条件を満たすため，既存の解析処理の問題点を述べ，対処として解析処理の並行動作化を提案した．評価では，並行動作化による解析処理時間の短縮効果を示した．また，1 台の解析処理計算機で複数の被測定計算機を対象とする場合を想定した解析処理時間についても評価した．その結果，評価に用いた計算機環境（表 5.1）では，本提案手法で，1 台の解析処理計算機あたり，5VM が動作している被測定計算機 16 台，10VM が動作している被測定計算機なら 4 台を継続的に性能プロファイリングできることを示した．

以上より，本論文では，VM における性能プロファイリングシステムの既存手法の問題点解消のために，データ収集オーバヘッドの削減手法を第 2 章で確立し，物理/仮想 CPU 数の違いを考慮した測定精度の向上手法を第 3 章で確立した．さらに，新たに VM における継続的な性能プロファイリングシステムの実現手法を確立するために，継続的な性能プロファイリングを可能にするシステムの分散化とデータ収集停止時間を短縮する手法を第 4 章で確立し，分散化した性能プロファイリングシステムの解析処理時間の短縮手法を第 5 章で確立した．これらにより，本研究では，クラウドで利用されている VM における継続的な性能プロファイリングシステムの実現手法を確立した．このため，クラウドの更なる発展と普及が期待される将来においても，本研究は，クラウドの保守性向上における有効性を期待できる．また，クラウドの円滑な運用を支えることを通し，本研究で確立した性能プロファイリングシステムは，現在の情報社会（Society 4.0）およびこれからの人間中心の社会（Society 5.0）

における円滑な社会活動を支えるシステムとなることを期待できる.

謝辞

本論文をまとめるにあたり，ご査読いただき，かつご助言を賜りました岡山大学 大学院自然科学研究科 産業創成工学専攻 谷口秀夫 教授，名古屋彰 教授，山内利宏 准教授に心より感謝いたします．

特に，谷口秀夫 教授には，本研究を進めるにあたり，終始，非常に丁寧なご指導やご鞭撻を賜りました．深く感謝いたします．

また，名古屋彰 教授，山内利宏 准教授には，研究に関する貴重なご教示を賜りました．厚く感謝いたします．

さらに，岡山大学 大学院 自然科学研究科 産業創成工学専攻 佐藤将也 助教には，本研究を進めるためのご支援を賜りました．ここに感謝いたします．

本論文は，株式会社富士通研究所における研究から新たな方向に発展させた研究成果をまとめたものであります．

本研究の機会を与えて頂き，さらに様々なご指導やご鞭撻を頂いた元 株式会社富士通研究所 久門耕一 取締役，株式会社富士通研究所コンピュータシステム研究所 堀江健志 所長（現富士通株式会社），株式会社富士通研究所データシステムプロジェクト 赤星直輝 プロジェクトディレクター（現 ICT 研究所），株式会社富士通研究所データシステムプロジェクト 中島耕太 主管研究員（現先端コンピュータプロジェクト）の各位に心より御礼申し上げます．

また，本論文を執筆する機会を与えて頂くと共に，温かいご支援を頂いた株式会社富士通研究所サービスビジネス開発運用・ユニット 野村祐士 ユニット長，株式会社富士通研究所運用革新プロジェクト 奥田將人 プロジェクトディレクター，渡辺幸洋 マネージャーの各位に厚く感謝いたします．

参考文献

[1] アメリカ合衆国国勢調査局：UNIVAC UNVEILED AT PHOLADELPHIA DEDICATION, *Census Bulletion*, Vol. I, No. 16 (online), available from ⟨https://www.census.gov/history/pdf/censusbulletin-univac.pdf⟩ (1951).

[2] Weik, Martin H.: *A third survey of domestic electronic digital computing systems. Ballistic Research Laboratories Report No. 1115, Department of the Army Project No.5B03-06-002, Ordnance Management Structure Code No.5010.11.812.*, Aberdeen Proving Ground, Maryland (1961).

[3] Patterson, D. A. and Hennessy, J. L.: コンピュータの構成と設計第 3 版＜別冊＞歴史展望，日経 BP 社 (2007).

[4] 谷口秀夫：オペレーティングシステム，第 10 巻，昭晃堂 (1995).

[5] Tanenbaum, A. S. and Bos, H.: *Modern Operating Systems*, Prentice Hall Press, Upper Saddle River, NJ, USA, 4th edition (2014).

[6] 谷口秀夫：分散処理，オーム社 (2005).

[7] Gray, J. and Reuter, A.: *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition (1992).

[8] Metcalfe, R. M. and Boggs, D. R.: Ethernet: Distributed Packet Switching for Local Computer Networks, *Commun. ACM*, Vol. 19, No. 7, pp. 395–404 (online), DOI: 10.1145/360248.360253 (1976).

[9] Comer, D. E.: *Internetworking with TCP/IP*, Addison-Wesley Professional, 6th edition (2013).

[10] O'Reilly, T.: What Is Web 2.0 - Design Patterns and Business Models for the Next Generation of Software, O'Reilly (online), available from ⟨https://www.oreilly.com/pub/a/web2/archive/what-is-web-20.html⟩ (accessed 2018-12-19).

[11] Smith, J. and Nair, R.: *Virtual Machines: Versatile Platforms for Systems and Processes (The Morgan Kaufmann Series in Computer Architecture and Design)*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2005).

[12] Mell, P. and Grance, T.: NISTによるクラウドコンピューティングの定義，独立行政法人情報処理推進機構（オンライン），入手先 ⟨https://www.ipa.go.jp/files/000025366.pdf⟩（参照 2018-09-24）.

[13] NIST: NIST Cloud Computing Program - NCCP, National Institute of Standards and Technology (NIST) (online), available from ⟨https://www.nist.gov/programs-projects/nist-cloud-computing-program-nccp⟩ (accessed 2018-09-24).

[14] 内閣府公式ページ:科学技術基本計画，内閣府（オンライン），入手先 ⟨http://www8.cao.go.jp/cstp/kihonkeikaku/index5.html⟩（参照 2018-09-24）.

[15] 内閣府公式ページ:Society 5.0, 内閣府 (online)，available from ⟨http://www8.cao.go.jp/cstp/society5_0/index.html⟩ (accessed 2018-09-24).

[16] JEITA: なぜ今、Society 5.0 を目指すのか　社会とテクノロジーの双方が求める“未来”，一般社団法人電子情報技術産業協会（オンライン），入手先 ⟨http://www8.cao.go.jp/cstp/society5_0/index.html⟩（参照 2018-09-24）.

[17] AIST: ABCI (AI Bridging Cloud Infrastructure), 国立研究開発法人産業技術総合研究所 (online)，available from ⟨https://abci.ai/ja/⟩ (accessed 2018-09-24).

[18] Rosenblum, M. and Garfinkel, T.: Virtual Machine Monitors: Current Technology and Future Trends, *Computer*, Vol. 38, No. 5, pp. 39–47 (online), DOI: 10.1109/MC.2005.176 (2005).

[19] Kivity, A., Kamay, Y., Laor, D., Lublin, U. and Liguori, A.: kvm: the Linux virtual machine monitor, *Linux Symposium* (2007).

[20] Red Hat, Inc.: KVM, , available from ⟨http://www.linux-kvm.org/⟩ (accessed 2018-10-29).

[21] レッドハット株式会社：KVM とは，レッドハット株式会社（オンライン），入手先 ⟨https://www.redhat.com/ja/topics/virtualization/what-is-KVM⟩（参照 2018-10-29）.

[22] RED HAT DEVELOPER: Setting up KVM on Red Hat Enterprise Linux - RHD Blog, Red Hat, Inc. (online), available from ⟨https://developers.redhat.com/blog/2016/08/18/setting-up-kvm-on-rhel/⟩ (accessed 2018-10-29).

[23] University of Cambridge Computer Laboratory: Computer Laboratory - Xen virtual machine monitor, University of Cambridge Computer Laboratory (online), available from ⟨https://www.cl.cam.ac.uk/research/srg/netos/projects/archive/xen/⟩ (accessed 2018-10-29).

[24] The Xen Project: The Xen Project, the powerful open source industry standard for virtualization., The Xen Project (online), available from ⟨https://www.xenproject.org/⟩ (accessed 2018-10-29).

[25] Citrix Systems, Inc.: Citrix Hypervisor (formerly XenServer) - Server Virtualization and Consolidation, Citrix (online), available from ⟨https://www.citrix.com/products/citrix-hypervisor/⟩ (accessed 2018-10-29).

[26] ヴイエムウェア株式会社：vSphere ESXi ベアメタルハイパーバイザー，ヴイエムウェア株式会社（オンライン），入手先 ⟨https://www.vmware.com/jp/products/esxi-and-esx.html⟩（参照 2018-10-29）.

[27] ヴイエムウェア株式会社：vSphere Hypervisor — 無償のベアメタルハイパーバイザー，ヴイエムウェア株式会社（オンライン），入手先 ⟨https://www.vmware.com/jp/products/vsphere-hypervisor.html⟩（参照 2018-10-29）.

[28] VMware, Inc.: VMware vSphere Hypervisor ESXi Support Center, VMware, Inc. (online), available from ⟨https://www.vmware.com/support/vsphere-hypervisor.html⟩ (accessed 2018-10-29).

[29] Microsoft Corporation: サーバーの仮想化—Windows Server 2016 — Microsoft，Microsoft（オンライン），入手先 ⟨https://www.microsoft.com/ja-jp/cloud-platform/server-virtualization⟩（参照 2018-10-29）.

[30] Microsoft Corporation: Download Microsoft Hyper-V Server 2008 R2 from Official Microsoft Download Center, Microsoft (online), available from ⟨https://www.microsoft.com/ja-jp/download/details.aspx?id=3512⟩ (accessed 2018-10-29).

[31] Microsoft Corporation: Hyper-V on Windows Server 2016 — Microsoft Docs, Microsoft (online), available from ⟨https://docs.microsoft.com/ja-jp/windows-server/virtualization/hyper-v/Hyper-V-on-Windows-Server⟩ (accessed 2018-10-29).

[32] Levon, J. and Elie, P.: Oprofile: A system profiler for linux, Hewlett-Packard Company (online), available from ⟨http://oprofile.sourceforge.net⟩ (accessed 2019-4-23).

[33] Intel Corp.: Intel VTune Amplifier, Intel Corp. (online), available from ⟨http://software.intel.com/en-us/articles/intel-vtune-amplifier-xe/⟩ (accessed 2019-4-23).

[34] Red Hat, Inc.: perf: Linux profiling with performance counters, Red Hat, Inc. (online), available from ⟨https://perf.wiki.kernel.org/index.php/Main_Page⟩ (accessed 2019-4-23).

[35] Kessler, S. G. P. and Mckusick, M.: Gprof: A call graph execution profiler, *ACM SIGPLAN Notices* (1982).

[36] Srivastava, A. and Eustace, A.: ATOM: A System for Building Customized Program Analysis Tools, *Proceedings of the ACM SIGPLAN 1994 Conference on Programming Language Design and Implementation*, PLDI '94, New York, NY, USA, ACM, pp. 196–205 (online), DOI: 10.1145/178243.178260 (1994).

[37] Zhang, X., Wang, Z., Gloy, N., Chen, J. B. and Smith, M. D.: System Support for Automatic Profiling and Optimization, *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, SOSP '97, New York, NY, USA, ACM, pp. 15–26 (online), DOI: 10.1145/268998.266640 (1997).

[38] De Bus, B., Chanet, D., De Sutter, B., Van Put, L. and De Bosschere, K.: The Design and Implementation of FIT: A Flexible Instrumentation Toolkit, *Proceedings of the 5th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, PASTE '04, New York, NY, USA, ACM, pp. 29–34 (online), DOI: 10.1145/996821.996833 (2004).

[39] Cohn, R., Goodwin, D., Lowney, P. G. and Rubin, N.: Spike: An Optimizer for Alpha/NT Executables, *Proceedings of the USENIX Windows NT Workshop on The USENIX Windows NT Workshop 1997*, NT'97, Berkeley, CA, USA, USENIX Association, pp. 3–9 (online), available from ⟨http://dl.acm.org/citation.cfm?id=1267658.1267661⟩ (1997).

[40] Romer, T., Voelker, G., Lee, D., Wolman, A., Wong, W., Levy, H., Bershad, B. and Chen, B.: Instrumentation and Optimization of Win32/Intel Executables Using Etch, *Proceedings of the USENIX Windows NT Workshop on The USENIX Windows NT Workshop 1997*, NT'97, Berkeley, CA, USA, USENIX Association, pp. 1–7 (online), available from ⟨http://dl.acm.org/citation.cfm?id=1267658.1267659⟩ (1997).

[41] Arnold, M. and Ryder, B. G.: A Framework for Reducing the Cost of Instrumented Code, *SIGPLAN Not.*, Vol. 36, No. 5, pp. 168–179 (online), DOI: 10.1145/381694.378832 (2001).

[42] Bitzes, G. and Nowak, A.: The overhead of profiling using PMU hardware counters, *CERN openlab Report* (2014).

[43] Menon, A., Santos, J. R., Turner, Y., Janakiraman, G. J. and Zwaenepoel, W.: Diagnosing Performance Overheads in the Xen Virtual Machine Environment, *Proceedings of the 1st ACM/USENIX International Conference on Virtual Execution Environments*, VEE '05, New York, NY, USA, ACM, pp. 13–23 (online), DOI: 10.1145/1064979.1064984 (2005).

[44] Du, J., Sehrawat, N. and Zwaenepoel, W.: Performance profiling of virtual machines, *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual Execution Environments, VEE'11* (2011).

[45] Serebrin, B. and Hecht, D.: Virtualizing Performance Counters, *5th Workshop on System-level Virtualization for High Performance Computing, Euro-Par 2011:HPCVirt* (August 2011).

[46] VMware, Inc.: Knowledge Base: Using Virtual CPU Performance Monitoring Counters (2030221), VMware, Inc. (online), available from ⟨http://kb.vmware.com/kb/2030221⟩ (accessed 2016-12-12).

[47] Red Hat, Inc.: Red Hat Enterprise Linux 7: Virtualization Tuning and Optimization Guide: §2.2. Virtual Performance Monitoring Unit (vPMU), Red Hat, Inc. (online), available from ⟨https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/Virtualization_Tuning_and_Optimization_Guide/sect-Virtualization_Tuning_Optimization_Guide-Monitoring_Tools-vPMU.html⟩ (accessed 2018-12-12).

[48] Ostrovsky, B.: Xen PMU: Perf Support in Xen, *Xen Project Developer Summit* (2013).

[49] Anderson, J. M., Berc, L. M., Dean, J., Ghemawat, S., Henzinger, M. R., Leung, S. A., Sites, R. L., Vandevoorde, M. T., Waldspurger, C. A. and Weihl, W. E.: Continuous Profiling: Where Have All the Cycles Gone?, *ACM Trans. Comput. Syst.*, Vol. 15, No. 4, pp. 357–390 (1997).

[50] Ren, G., Tune, E., Moseley, T., Shi, Y., Rus, S. and Hundt, R.: Google-Wide Profiling: A Continuous Profiling Infrastructure for Data Centers, *IEEE Micro*, pp. 65–79 (2010).

[51] Kanev, S., Darago, J. P., Hazelwood, K., Ranganathan, P., Moseley, T., Wei, G.-Y. and Brooks, D.: Profiling a Warehouse-scale Computer, *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, ISCA '15, ACM, pp. 158–169 (2015).

[52] Standard Performance Evaluation Corporation: SPEC CPU95, , available from ⟨https://www.spec.org/cpu95/⟩ (accessed 2019-3-8).

[53] Weaver, V. M., Terpstra, D., McCraw, H., Johnson, M., Kasichayanula, K., Ralph, J., Nelson, J., Mucci, P., Mohan, T. and Moore, S.: PAPI 5: Measuring Power, Energy, and the Cloud, *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS 2013)*, pp. 124–125 (April 2013).

[54] VMware, Inc.: Knowledge Base: Using Virtual CPU Performance Monitoring Counters (2030221), *http://kb.vmware.com/kb/2030221*.

[55] Kivity, A.: Performance Monitoring for KVM Guests, *KVM Forum* (2011).

[56] Yanmin, Z.: Enhance perf to collect KVM guest os statis-tics from host side, (online), available from ⟨http://lwn.net/Articles/378778⟩.

[57] Du, J., Sehrawat, N. and Zwaenepoel, W.: Performance profiling in a virtualized environment, *Proceedings of the 2nd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud'10* (2010).

[58] Intel Corp.: Hardware-Assisted Intel Virtualization Technology (Intel VT), *http://www.intel.com/content/www/us/en/virtualization/virtualization-technology/hardware-assist-virtualization-technology.html*.

[59] Intel Corp.: Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3: System Programming Guide, *http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html*.

[60] Huck, J. and Hays, J.: Architectural support for translation table management in large address space machines, *ISCA '93 Proceedings of the 20th annual international symposium on computer architecture*, pp. 39–50 (1993).

[61] Jones, S., Arpaci-Dusseau, A. and Arpaci-Dusseau, R.: AntFarm: Tracking Processes in a Virtual Machine Environment, *ATEC '06 Proceedings of the annual conference on USENIX '06 Annual Technical Conference*, pp. 1–14 (2006).

[62] Bellard, F.: QEMU, a Fast and Portable Dynamic Translator, *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '05 (2005).

[63] Butscher, B. and Weimer, H.: libquantum - Simulation of quantum mechanics, , available from ⟨http://www.libquantum.de/⟩ (accessed 2019-3-8).

[64] Oracle Corporation and/or its affiliates: MySQL, , available from ⟨http://www.mysql.com/⟩ (accessed 2018-12-12).

[65] Oracle Corporation: jdk-7u79-linux-x64.rpm, , available from ⟨http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html⟩ (accessed 2018-12-12).

[66] Standard Performance Evaluation Corporation: SPEC CPU 2006, , available from ⟨https://www.spec.org/cpu2006/⟩ (accessed 2019-3-8).

[67] Kopytov, A.: SysBench, sysbench-dev (online), available from ⟨https://github.com/akopytov/sysbench/⟩ (accessed 2018-12-12).

[68] Standard Performance Evaluation Corporation: SPECjbb 2013, , available from ⟨https://www.spec.org/jbb2013/⟩ (accessed 2018-12-12).

[69] The Linux man-pages project: Linux Programmer's Manual PROC(5), *http://man7.org/linux/man-pages/man5/proc.5.html*.

[70] PostgreSQL Global Development Group: PostgreSQL: The World's Most Advanced Open Source Relational Database, , available from ⟨http://www.postgresql.org/⟩ (accessed 2018-12-12).

[71] Yamamoto, M., Ono, M., Nakashima, K. and Hirai, A.: Unified Performance Profiling of an Entire Virtualized Environment, *International Journal of Networking and Computing*, Vol. 6, No. 1, pp. 124–147 (2016).

[72] Uhlig, R., Neiger, G., Rodgers, D., Santoni, A. L., Martins, F. C. M., Anderson, A. V., Bennett, S. M., Kagi, A., Leung, F. H. and Smith, L.: Intel virtualization technology, *Computer*, Vol. 38, No. 5, pp. 48–56 (online), DOI: 10.1109/MC.2005.163 (2005).

[73] Himeno, R.: Himeno benchmark, RIKEN (online), available from ⟨http://accc.riken.jp/supercom/himenobmt/⟩ (accessed 2019-4-23).

[74] Lozi, J., Lepers, B., Funston, J., Gaud, F., Quéma, V. and Fedorova, A.: The Linux Scheduler: A Decade of Wasted Cores, *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys '16, ACM, pp. 1:1–1:16 (2016).

[75] Amazon Web Services, Inc.: Amazon Web Services, Amazon Web Services, Inc. (online), available from ⟨https://aws.amazon.com/⟩ (accessed 2018-12-12).

[76] Amazon.com, Inc.: Amazon CloudWatch, Amazon.com,Inc. (online), available from ⟨https://aws.amazon.com/cloudwatch/⟩ (accessed 2018-10-18).

[77] VMware, Inc.: vCenter Server 5.5 Update 1 Release Notes, VMware, Inc. (online), available from ⟨https://www.vmware.com/support/vsphere5/doc/vsphere-vcenter-server-55u1-release-notes.html⟩ (accessed 2018-12-12).

[78] 山本昌生，中島耕太，山内利宏，名古屋彰，谷口秀夫：仮想計算機を利用した性能プロファイリングシステムの分散化，*IPSJ SIG Technical Reports*, Vol. 2017-OS-139, No. 8, pp. 1–8.

[79] Ceilometer Drivers: OpenStack Telemetry (Ceilometer), OpenStack Foundation (online), available from ⟨https://github.com/openstack/ceilometer⟩ (accessed 2018-10-18).

[80] Suchakrapani Datt Sharma, Hani Nemati, G. B. and Dagenais, M.: Low Overhead Hardware-Assisted Virtual Machine Analysis and Profiling, *IEEE Globecom Workshops (GC Workshops)* (2016).