監視制御アプリケーションに対する
クラウドコンピューティングの適用に関する研究

提出先　　大阪大学大学院情報科学研究科

提出年月　2017 年 12 月

金 子　雄

関連発表論文

1. 学会論文誌発表論文

1. 金子 雄, 伊藤俊夫, 原 隆浩：ビル設備機器の状態監視における仮想化技術の適用可能性の検討, 情報処理学会論文誌 (ネットワークサービスと分散処理特集), Vol. 58, No. 2, pp. 461–470 (2017 年 2 月).

2. 金子 雄, 伊藤俊夫, 原 隆浩：仮想化を利用した設備機器の状態監視における計測時刻誤差の推定手法, 情報処理学会論文誌 (ネットワークサービスと分散処理特集), (採録決定，2018 年 2 月予定).

2. 著書（章筆）

1. 電気学会 スマートグリッドに関する電気事業者・需要家間サービス基盤技術調査専門委員会 編：国際標準に基づくエネルギーサービス構築の必須知識, オーム社 (2016 年 12 月).

3. 国際会議発表論文

1. Kaneko, Y., Ito, T., Maegawa, T.: A Virtual Machine Resource Management Method with Millisecond Precision, Proc. of IEEE Int'l Conference on Autonomic Computing (ICAC 2015), pp. 223–226 (July 2015).

2. Kaneko, Y., Ito, T.: A Reliable Cloud-based Feedback Control System, Proc. of IEEE Int'l Conference on Cloud Computing (CLOUD 2016), pp. 880–883 (June 2016)

3. Kaneko, Y., Ito, T., Hara, T.: A Measurement Study on Virtualization Overhead for Applications of Industrial Automation Systems, Proc. of IEEE

Int'l Conference on Emerging Technologies and Factory Automation (ETFA 2016), pp. 1–8 (Sep. 2016).

4. Ito, T., Kaneko, Y.: A Semantic Dataflow Logger Connecting Java Objects and Database Rows and Columns, Proc. of Int'l Symposium on Computing and Networking (CANDAR 2016), pp. 84–90 (Nov. 2016).

5. Kaneko, Y., Ito, T., Ito, M., Kawazoe, H.: Virtual Machine Scaling Method Considering Performance Fluctuation of Public Cloud, Proc. of IEEE Int'l Conference on Cloud Computing (CLOUD 2017), pp. 782–785 (June 2017).

6. Kaneko, Y., Ito, T., Hara, T.: Performance Estimation Method for Virtualized Building Facilities Monitoring Applications, Proc. of IEEE Int'l Conference on Emerging Technologies And Factory Automation (ETFA 2017), (presented, Sep. 2017).

4. 国内会議発表論文（査読有）

1. 金子 雄, 伊藤俊夫, 原 隆浩：仮想化が機器状態監視の定刻性に与える影響の一評価, 情報処理学会マルチメディア，分散，協調とモバイル (DICOMO2016) シンポジウム論文集, pp. 1566–1577 (2016 年 7 月).

2. 毛カイ毅, 伊藤俊夫, 金子 雄：来歴情報を用いた複数システム間でのデータ追跡に関する一提案, 情報処理学会マルチメディア，分散，協調とモバイル (DICOMO2016) シンポジウム論文集, pp. 1491–1496 (2016 年 7 月).

5. その他の研究会等発表論文

1. 金子 雄, 伊藤俊夫, 前川智則：仮想マシンのリソースをミリ秒精度で管理する手法の検討と試作, 情報処理学会 全国大会論文集, Vol. 2015, No. 1, pp. 15–16 (2015 年 3 月).

2. 金子 雄, 毛カイ毅：OpenADR 2.0 Profile B の課題と対策の一検討, 情報科学技術フォーラム (FIT2015) 講演論文集, pp. 463–464 (2015 年 9 月).

3. 金子 雄, 伊藤俊夫, 原 隆浩：仮想化された機器監視アプリケーションの計測時刻の誤差推定手法, 情報処理学会研究報告 (マルチメディア通信と分散処理), Vol. 2017, No. 11, pp. 1–8 (2017 年 2 月).

4. 金子 雄, 田中立二：OpenFMB の実用化に向けた課題の一検討, 電子情報通信学会技術研究報告, Vol. 117, No. 89, pp. 7–12 (2017 年 6 月).

6. 受賞歴

1. 金子 雄, 伊藤俊夫, 原 隆浩: 山下記念研究賞, 情報処理学会研究報告 (マルチメディア通信と分散処理), (2017 年 2 月).

7. 紀要

1. 金子 雄, 山田孝裕, 松澤茂雄：地域エネルギーマネジメントシステムの OpenADR2.0b 対応, 東芝レビュー, Vol. 70, No. 9, pp. 29–32 (2015 年 9 月).

2. 伊藤俊夫, 金子 雄, 前川智則：複雑なネットワークシステムでも高いサービス品質を維持するためのデータフロー管理技術, 東芝レビュー, Vol. 71, No. 6, pp. 40–43 (2016 年 12 月).

8. 特許出願

1. 金子 雄, 前川智則, 米良恵介, 伊藤将志：情報処理装置及び通信装置, 特開 2016-046724.

2. 金子 雄, 伊藤俊夫, 前川智則：リソース割当て装置、方法、及びプログラム, 特開 2016-091109.

3. 金子 雄, 伊藤俊夫, 前川智則：リソース制御装置、方法、及びプログラム, 特開 2016-118841.

4. 金子 雄 : 情報収集管理装置、方法、及び情報収集システム, 特開 2017-034307.

5. 金子 雄：コントローラおよび制御システム, 特開 2017-092873.

以上

内容概要

監視制御システムとは，機器群の状態を計測し，必要に応じて制御することで，機器群および機器群から構成されるシステムの稼働状態を安定に保つためのシステムである．監視制御システムの適用対象は様々であり，例えば，ビル，工場，プラント，飛行機，鉄道，自動車などがある．監視制御システムは，監視制御対象の安定性を保つためのシステムであるから，高い安定性が要求される．この場合の安定性とは，稼働率と性能により定義される．

現在，多くの監視制御システムは，高い安定性を達成するために，専用のハードウェアを用いて構築されている．さらに，専用のオペレーティングシステム (OS; Operating System) が用いられている場合もある．そのため，監視制御システムは高価になる傾向があり，低コスト化が求められている．

監視制御システムを，クラウドコンピューティングを利用して構築することで，監視制御システムの低コスト化を実現しようとする研究が注目されている．クラウドコンピューティングにおいて，ベースとなる技術は計算機の仮想化技術である．計算機の仮想化技術は，ソフトウェアによって仮想的な計算機 (VM; Virtual Machine) を作成する．ここで，VM 上で動作するアプリケーションの性能は低下する傾向にあるため，求められる性能を達成できなくなる可能性が生じる．また，既存のパブリッククラウドのプロバイダが保証する VM の稼働率は，監視制御システムに要求される稼働率に及ばない．そのため，単純にパブリッククラウドを利用すると，要求される稼働率を達成できない．クラウドコンピューティングを利用した監視制御システムにおいて安定性を実現するためには，これらの問題を解決する必要がある．

そこで本論文では，監視制御システムにクラウドコンピューティングを適用した場合に，監視制御システムの高い安定性を保証するための方法について議論する．本論文ではまず，プライベートクラウド上で監視制御システムを構築することを想定し，監視制御アプリケーションを VM 上で動作させた場合の性能への影響を把握する．プライベートクラウドとは，企業や大学などの組織が，自組織で使用するために構築するクラウドコンピューティング環境である．次に，監視制

御アプリケーションをVM上で動作させた場合の性能を推定する手法を提案する．次に，パブリッククラウド上で監視制御システムを構築することを想定し，その稼働率を向上する手法を提案する．パブリッククラウドとは，多数の企業や大学などに利用してもらうために構築されたクラウドコンピューティング環境であり，プライベートクラウドと比べると，稼働率を保証することが難しい．

本論文は5章から構成され，その内容は次の通りである．まず第1章で序論を述べる．第2章では，プライベートクラウド上で監視制御システムを構築することを想定する．プライベートクラウドであるから，監視制御システムに要求される高い稼働率を，ハードウェアにより保証できる．一方，VM上で動作するアプリケーションの性能は低下する傾向にある．監視制御アプリケーションに要求される性能を達成するためには，VMが監視制御アプリケーションの性能に与える影響を把握することが重要である．第2章では，VMに割り当てるリソース量や，複数のVMの実行が性能に与える影響を評価する．また，物理マシンのCPUコアの一部を仮想化ソフトウェアに占有させることによる性能への影響や，CPUスケジューリングの方法が性能に与える影響を評価する．評価結果から，VM間のCPU競合や，仮想化ソフトウェアのCPU不足が，監視制御アプリケーションの性能を低下させる主要因であることを確認した．

次に第3章において，第2章で得られた知見に基づき，VM上で動作する監視制御アプリケーションの性能を推定する手法を提案する．この手法は，物理マシンやVMの挙動をモデル化し，そのモデルに基づくシミュレーションにより，複数の監視制御アプリケーション／VMを単一の物理マシン上で実行した場合の性能を推定する．モデルにおける物理マシンの性能に依存するパラメータは，実機をモニタリングした結果から値を求めることで，シミュレーションの精度を向上する．複数のVMによるCPU競合を考慮して性能を推定する点と，仮想化ソフトウェアのCPU不足を考慮して性能を推定する点が，提案手法の特徴である．実機を用いた性能評価と，提案手法による推定結果を比較し，提案手法の推定精度を検証した．その結果，監視制御アプリケーションの性能が著しく低下する状況を推定できることを確認した．

第4章では，監視制御アプリケーションをパブリッククラウド上で実行する環

境を想定し，その稼働率を向上させる手法を提案する．この手法では，各クラウド上で実行される監視制御アプリケーションが，定期的にメッセージを交換することで，互いに生存確認を行う．また，単に生存確認を行うだけでなく，それぞれが実施している監視制御処理の情報や，監視制御アプリケーションと監視制御対象との間のネットワークの品質の情報を共有することで，障害発生時の制御品質の低下を抑える．パブリッククラウドを使用した評価を行い，提案手法が制御品質の低下を抑えられるかを検証した．その結果，提案手法は，監視制御処理の情報やネットワーク品質を共有しない場合と比べて，制御品質の低下を抑えられることを確認した．

第5章では，本論文の成果を要約した後，今後の研究課題について述べる．

目 次

第1章　序論

1.1　研究背景

監視制御システムとは，機器群の状態を計測し，必要に応じて制御することで，機器群および機器群から構成されるシステムの稼働状態を安定に保つためのシステムである．監視制御システムの適用対象は様々であり，例えば，ビル，工場，プラント，飛行機，鉄道，自動車などがある．例えばビルであれば，ビル管理システムと呼ばれる監視制御システムが導入されている場合がある．ビル管理システムは，ビル内の室温を計測し，室温を快適に保つために空調を制御したり，火災報知機の状態を定期的に確認し，火災発生時に防火扉やスプリンクラーを制御したりする．

監視制御システムは，監視制御対象の安定性を保つためのシステムであるから，監視制御システム自身も高い安定性が要求される．この場合の安定性とは，稼働率と性能により定義される．IEC 61508 では，監視制御システムの稼働率の基準 (SIL; Safety Integrity Level) を定めている．例えば，連続動作するシステムの場合，SIL1 の稼働率は 99.999%，SIL2 は 99.9999%，SIL3 は 99.99999%である．このように高い稼働率が要求される理由は，監視制御システムの障害は，人々を危険にさらす可能性があるためである．性能に対する要件は，監視制御システムを構成する監視制御アプリケーションごとに定義される．必ずしも，高い性能が求められるわけではない．ただし，要求される性能を達成し続けることが求められる．

現在，多くの監視制御システムは，高い安定性を達成するために，専用のハードウェアを用いて構築されている．さらに，専用のオペレーティングシステム (OS; Operating System) が用いられている場合もある．そのため，監視制御システムは高価になる傾向があり，低コスト化が求められている．

一方，様々なシステムを構築／運用するためのプラットフォームとして，クラウドコンピューティングが普及しており，企業のウェブサイトやシステムの運用，アプリケーションの開発やテスト，様々なウェブサービスを提供するために用いられている[1]．クラウドコンピューティングは，プライベートクラウドとパブリッククラウドに分類できる．プライベートクラウドでは，ハードウェアコストの削減や，運用の柔軟性の向上などが期待できる．パブリッククラウドでは，さらに，システム運用のコスト削減や，システムのスケーラビリティの向上が期待できる．このようなメリットから，クラウドコンピューティングは様々なシステムの構築／運用に活用されている．

監視制御システムを，クラウドコンピューティングを利用して構築することで，上述のメリットを享受できる．そのために，クラウドコンピューティングを利用した監視制御システムの実現に向けた研究が行われている [28, 29]．ただしクラウドコンピューティングは，監視制御システムの要件である安定性の実現を難しくする．プライベートクラウドでもパブリッククラウドでも，ベースとなる技術は計算機の仮想化技術である．計算機の仮想化技術は，ソフトウェアによって仮想的な計算機 (VM; Virtual Machine) を作成する．VM 上で動作するアプリケーションの性能は低下する傾向にあるため，求められる性能を達成できなくなる可能性が生じる [5, 56]．また，既存のパブリッククラウドのプロバイダが保証する VM の稼働率は 99.95%であり，SIL1 で定義される 99.999%に及ばない．そのため，単純にパブリッククラウドを利用すると，要求される稼働率を達成できない．クラウドコンピューティングを利用した監視制御システムを実現するためには，これらの問題を解決する必要がある．

1.2　クラウドコンピューティング

クラウドコンピューティングとは，その利用者 (ユーザ) に対して，CPU やメモリなどの計算資源を，必要なときに必要なだけ簡単に使えるようにするためのアプリケーション／システム実行環境である．従来のレンタルサーバと比べて，使

[1] https://www.rightscale.com/lp/2015-state-of-the-cloud-report

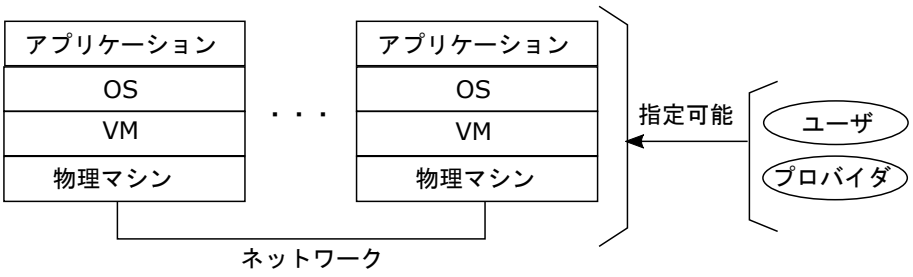


図 1.1: プライベートクラウドの環境と，ユーザおよびプロバイダの関係

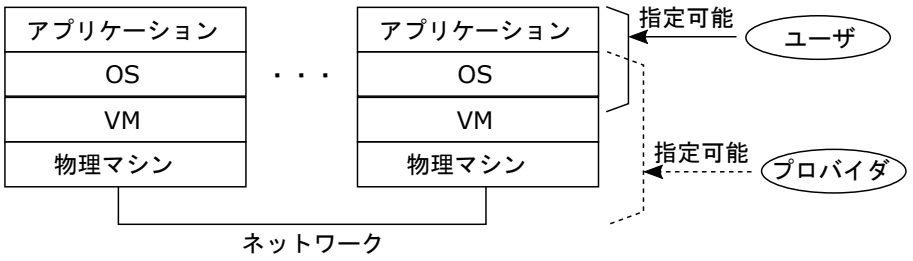


図 1.2: パブリッククラウドの環境と，ユーザおよびプロバイダの関係

用する計算資源の量を柔軟に指定できる点や，使用した計算資源の量に応じて課金する「従量課金制」を導入している点，計算資源を使用するための準備が短時間で済む点が特徴である．

クラウドコンピューティングは，プライベートクラウドとパブリッククラウドに分類される．プライベートクラウドとは，企業や大学などの組織が，自組織で使用するために構築するクラウドコンピューティング環境である．図 1.1 にプライベートクラウドと，ユーザおよびプロバイダの関係を示す．プライベートクラウドにおいては，ユーザとプロバイダは同一人物の場合がある．したがって，プライベートクラウドにおいては，ユーザが自らの目的に応じてアプリケーション，オペレーティングシステム，物理マシンなどの種類を指定できる．また，ユーザの数や種類を想定しやすく，かつ，制限しやすい．したがって，ユーザがプライベートクラウドの稼働率や性能を調節することは，後述するパブリッククラウドと比べると容易である．

一方，パブリッククラウドとは，多数の企業や大学などに利用してもらうために構築されたクラウドコンピューティング環境である．例えば Amazon や Google はパ

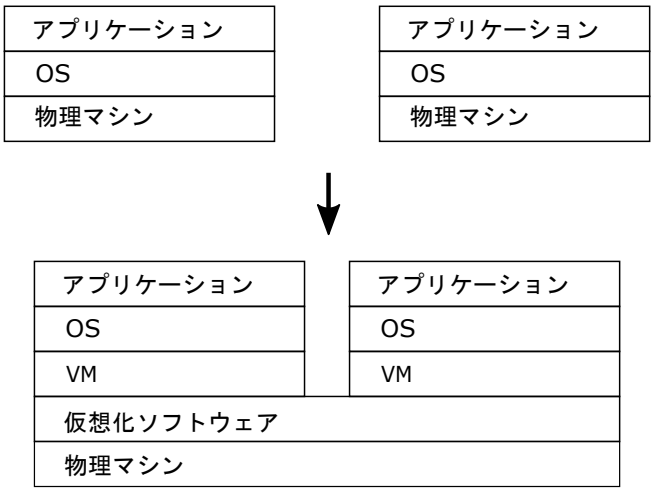


図 1.3: 計算機の仮想化の適用による OS やアプリケーションの集約の一例

ブリッククラウドのプロバイダであり，それぞれ Amazon EC2 や Google Compute Engine というパブリッククラウドを提供している．

図 1.2 にパブリッククラウドと，ユーザおよびプロバイダの関係を示す．パブリッククラウドでは，プロバイダが物理マシンやネットワークなどから構成されるシステムを設計し，構築する．ユーザはこれらを指定できない．オペレーティングシステムも基本的にはプロバイダが指定する．ただし，複数のオペレーティングシステムの中から，ユーザが選択できる場合もある．ユーザは，アプリケーションは指定できる．また，パブリッククラウドでは，他のユーザとハードウェアを共有することになる．したがって，ユーザがパブリッククラウドの稼働率や性能を調節することは，プライベートクラウドと比べると難しい．その代わり，プライベートクラウドよりも低コストで利用できる．

プライベートクラウドであれパブリッククラウドであれ，クラウドコンピューティングの基礎をなす技術は，計算機の仮想化技術である．計算機の仮想化技術とは，仮想的な CPU や NIC などを備えた VM をソフトウェアとして実現する技術である．図 1.3 に，計算機の仮想化技術を物理マシンに適用した場合の例を示す．このように，計算機の仮想化技術を使うことで，個別の物理マシン上で稼働していた OS やアプリケーションを，単一の物理マシンに集約できる．仮想化ソフトウェアとは，VM を管理するソフトウェアである．Xen [10] や KVM [55] は仮想

化ソフトウェアの一例である．仮想化ソフトウェアは，Virtual Machine Monitor (VMM) や Hypervisor と呼ばれる場合もある．

仮想化技術を適用した環境 (仮想化環境) において，各 VM は個別の OS を実行できるため，各 VM 上のアプリケーションは，他の VM 上のアプリケーションを意識せずに動作できる．ある OS やアプリケーションに障害が発生しても，別の VM 上のアプリケーションに影響が及ぶのを防ぐことができる．また，アプリケーションや OS の状態を保持したまま VM のバックアップを取ることができるため，障害発生後の復旧が容易になる．

単一の物理マシン上で複数の VM を実行し，それぞれの VM 上でアプリケーションを実行すれば，アプリケーション群を実行するために必要な物理マシン数を削減できる．物理マシンを減らすことは，クラウド環境の構築にかかる設備投資の削減につながる．設備投資とは，物理マシンの購入費，物理マシンが消費する電気，物理マシンを配置するための場所の確保である．設備投資を削減することは，クラウドコンピューティングを利用するためのコストの削減につながる．しかし，単一の物理マシン上で複数の VM を実行すると，VM 間で物理マシンの計算資源 (CPU やメモリなど) を奪い合うことになる．そのため，VM 上のアプリケーションの性能は，VM を使わない場合と比べて低下する傾向にある．したがって，クラウドコンピューティングには，コストと性能のトレードオフが存在する．アプリケーションの性能要件を満たしつつ，できるだけコストを下げる，すなわちできるだけ多くの VM を単一の物理マシン上で実行することは，クラウドコンピューティングにおける重要な課題である．

1.3 研究内容

本論文では，監視制御システムにクラウドコンピューティングを適用した場合に，監視制御システムの高い安定性を保証するための方法について議論する．具体的には，以下の三つの研究課題に取り組む．最初の二つの研究課題は，プライベートクラウド上で監視制御システムを構築するために解くべき課題である．三つ目の研究課題は，パブリッククラウド上で監視制御システムを構築するために

解くべき課題である．

本論文で想定する監視制御アプリケーションは，ビルに設置された空調や照明の状態を監視制御するアプリケーションである．監視制御のための通信処理を一定周期で繰り返すという単調なアプリケーションであるため，CPU やメモリの使用量は一定であり，ディスク IO はほぼ発生しない．所定のタイミングで監視制御の通信を実施する点と，IEC 61508 の SIL1 に準ずるために 99.999%の稼働率が要求される点が特徴である．

1.3.1　仮想化が監視制御アプリケーションの性能に与える影響の評価 (第 2 章)

最初の課題では，監視制御アプリケーションをプライベートクラウド上で実行する環境を想定する．プライベートクラウドであるから，監視制御システムに要求される高い稼働率を，ハードウェアにより保証できる．一方，1.2 節で述べたように，VM 上で動作するアプリケーションの性能は低下する傾向にある [5, 56]．監視制御アプリケーションに要求される性能を達成するために，VM が監視制御アプリケーションの性能に与える影響を把握することが重要である [31, 57]．

VM 上の監視制御アプリケーションの性能を評価した研究として，文献 [28, 62] があるが，VM に割り当てる CPU などのリソース量が性能に与える影響や，単一の物理マシン上で複数の VM を実行することが性能に与える影響は評価していない．そこで，VM に割り当てるリソース量や，複数の VM を実行することが監視制御アプリケーションの性能に与える影響を把握するための評価を行う．また，物理マシンの CPU コアの一部を仮想化ソフトウェアに占有させることによる性能への影響や，CPU スケジューリングの方法が性能に与える影響を把握するための評価を行う．

1.3.2 仮想化された監視制御アプリケーションの性能の推定 (第 3 章)

1.2 節で述べたように，アプリケーションの性能要件を満たしつつ，できるだけ多くのアプリケーション／VM を単一の物理マシン上で実行することは，クラウドコンピューティングにおける重要な課題である．監視制御アプリケーションの場合も，その性能要件を満たせる範囲で，できるだけ多くの監視制御アプリケーション／VM を単一の物理マシン上で実行できるとよい．これは，物理マシンの購入費を減らすだけでなく，物理マシンを設置するための場所の節約や，電力消費量の節約，運用にかかる手間の削減につながる．

アプリケーション／VM を物理マシンに追加する際に，アプリケーションの性能要件を満たせるかを判断する手法として，リソースに基づく手法がある．リソースとは CPU やメモリ，ストレージ，ネットワークの一部または全部である．リソースに基づく手法は，追加するアプリケーション／VM のリソース消費量の予測値よりも，物理マシンのリソース残量が大きい場合に，VM の追加を許可する [65, 87, 97]．しかし実際には，物理マシンのリソースが余っていても，複数の VM によるリソースの奪い合いが生じることで，アプリケーションの性能は低下する．この性能の低下により，監視制御アプリケーションの性能要件を満たせなくなる場合がある [46]．

この問題を解決するために，VM 上で動作する監視制御アプリケーションの性能を推定する手法を提案する．提案手法は，物理マシンや VM の挙動をモデル化し，そのモデルに基づくシミュレーションにより，複数の監視制御アプリケーション／VM を単一の物理マシン上で実行した場合の性能を推定する．モデルにおける物理マシンの性能に依存するパラメータは，実機をモニタリングした結果から値を求めることで，シミュレーションの精度を向上する．複数の VM によるリソースの奪い合いを考慮して性能を推定する点と，すでに稼働している監視制御アプリケーションに影響を与えない点が，提案手法の特徴である．

1.3.3　パブリッククラウドにおける監視制御アプリケーションの稼働率の向上 (第 4 章)

1.3.1 項と 1.3.2 項で述べた課題を解くことで，プライベートクラウドで監視制御システムを構築し，高い安定性を達成できるようになる．第 4 章では，監視制御アプリケーションをパブリッククラウド上で実行する環境を想定する．

1.2 節で述べたように，パブリッククラウドでは，利用者が稼働率や性能を調節することが難しい．パブリッククラウド上のある物理マシンを占有するサービスを利用すれば，性能を調節することはできると考えられる．ただし，本論文で想定する監視制御アプリケーションは，IEC 61508 の SIL1 に準ずるために 99.999%の稼働率を必要とする．一方，パブリッククラウドのプロバイダが保証する典型的な VM の稼働率は，99.95%である[2]．監視制御システムをパブリッククラウドで実行するためには，この稼働率のギャップを解消する必要がある．

パブリッククラウドにおける監視制御アプリケーションの稼働率向上を目指した研究がある [36]．この研究では，監視制御アプリケーションを複製し，それぞれを異なるクラウド上で実行することで，稼働率を向上している．あるクラウド上の監視制御アプリケーションに障害が発生したことは，監視制御対象のビルや工場側に設置するゲートウェイを利用することで検出する．これらの手法は，ビル側に設置するゲートウェイの機能に依存しており，ゲートウェイが単一障害点となりうる．

この問題を解決するために，ゲートウェイに依存せずに，稼働率を向上する手法を提案する．提案手法では，各クラウド上で実行される監視制御アプリケーションが，定期的に生存確認用のメッセージを交換することで，互いに生存確認を行う．このメッセージ交換の通信は，パブリッククラウドでは課金の対象となるが，そのコストは専用のゲートウェイを導入するよりも小さい．また，単に生存確認を行うだけでなく，それぞれが実施している監視制御処理の状況や，監視制御アプリケーションと監視制御対象との間のネットワークの品質を共有することで，障害発生時の監視制御処理の引継ぎを円滑に行う．

[2] https://aws.amazon.com/jp/ec2/sla/

1.4 本論文の構成

本論文は 5 章から構成され，本章以降の内容は次の通りである．

第 2 章では，監視制御アプリケーションを VM 上で実行した場合の性能を把握するための評価を行う．1.3.1 項で述べた通り，VM に割り当てるリソース量や，並列実行する VM 数が監視制御アプリケーションの性能に与える影響を評価する．また，物理マシンの CPU コアの一部を仮想化ソフトウェアに占有させることによる性能への影響や，CPU による性能への影響を評価する．評価結果に基づき，監視制御アプリケーションの性能に影響を与える要因を分析する．

第 3 章では，監視制御アプリケーションを VM 上で実行する場合の性能を推定する手法を提案する．1.3.2 項で述べた通り，物理マシンや VM の挙動をモデル化し，そのモデルに基づくシミュレーションを実施することで，複数の監視制御アプリケーション／ VM を単一の物理マシン上で実行した場合の性能を推定する．実機評価により得られた性能の実測値と，提案手法により推定された性能を比較し，提案手法の推定精度を検証する．

第 4 章では，監視制御アプリケーションをパブリッククラウド上で実行する場合を想定し，稼働率の向上手法を提案する．この手法では，1.3.3 項で述べた通り，各クラウド上で実行される監視制御アプリケーションが，定期的に生存確認用のメッセージを交換し，互いに生存確認を行う．パブリッククラウドを用いた実験により，提案手法により，ある監視制御アプリケーションに障害が発生した場合に，別の監視制御アプリケーションに処理が引き継がれることを検証する．

最後に第 5 章では，本論文の成果を要約したのち，今後の研究課題について述べる．

第 2 章は文献 [46, 49, 50] で公表した結果に，第 3 章は文献 [47, 51, 52] で公表した結果に，第 4 章は文献 [45, 48] で公表した結果に基づき論述する．

第2章　仮想化が監視制御アプリケーションの性能に与える影響の評価

2.1　まえがき

ビルの空調や照明などの設備機器を管理するシステムとして，ビル管理システムがある．ビル管理システムは，設備機器の状態を監視し，異常を発見した場合にはビル管理者に通知したり，電力消費量を削減するための制御を実施したりする．ビル管理システムは，そのシステムを構成するサーバなどを設置するための専用の監視室を必要とする．そのため，大規模ビルへの導入は進んでいるが，中規模／小規模ビルへの導入は進んでいない．

中小ビルの管理を効率的に行うためのシステムとして，遠隔ビル管理システムがある．遠隔ビル管理システムは，インターネットなどのWide Area Network (WAN)を経由して，複数のビルに設置された設備機器を管理する [9, 15, 38, 77]．従来のビル管理システムのようにビル内に専用の監視室を設ける必要がないため，中規模／小規模のビルへの適用が進んでいる．遠隔ビル管理システムの管理対象は，空調や照明などの設備機器の稼働状態や，室内の温度や湿度などである．以降，これらの管理対象のことを「監視点」，また監視点から得られるデータのことを「監視点データ」と呼ぶ．

遠隔ビル管理システムの監視制御機能は通常のビル管理システムと類似しており，例えば以下である．

- **可視化機能**: 監視点データを，ビル管理者にわかりやすく表示する機能．最新値を表示したり，値の推移をグラフとして表示したりする．

- **警報機能**: 設備機器が異常であると判断した場合に，警報を発行する機能．例えば，ビル管理者向けの監視画面に警告を表示したり，ビル管理者にメールで通知したりする．

- **課金機能**: 監視点データから設備機器の稼働時間を計算し，ビルのテナントに対する課金額を計算する機能．

- **フィードバック制御機能**: ある設備機器に対してフィードバック制御を行う機能．例えば，水の流量を調節するバルブを，ある開閉度に維持する場合などがある．

- **機器異常診断機能**: 監視点データの推移から，機器に異常が発生していないかを診断する機能．

遠隔ビル管理システムには，ビル群の監視点データを計測し，上記の監視制御機能に提供するための機能が存在する [17, 74, 85, 43, 53]．以降，本機能のことを「クローラ」と呼ぶ．上述の監視制御機能は，クローラが計測するデータに基づいて動作するため，クローラは重要な機能である．

遠隔ビル管理システムが管理する各ビルにはオーナー（ビルオーナー）が存在する．ビルオーナーごとにクローラを実行することで，あるクローラに障害が発生しても，他のビルオーナーへの監視制御機能の提供を継続しやすくなる．障害とは，例えば，あるビルのネットワークや機器の異常により，クローラの動作が遅れたり停止したりすることである．また，管理対象のビルを追加する際に，誤ったビル情報などを設定することにより，クローラが正しく動作しなくなることである．

ビルオーナーごとにクローラを実行するための構成として，ビルオーナーごとに物理マシンを用意し，各物理マシン上で 1 つのクローラを実行する構成が考えられる．クローラごとに専用の物理マシンを用意するため，あるクローラの障害が，他のクローラの動作に影響を与えることを避けられる．しかしこの構成は，ビルオーナーが保有するビルが 1 棟だとしても，専用の物理マシンを用意することになる．そのため，遠隔ビル管理システムの運用者にとって設備投資の増加が問

題となる．ここで設備投資とは，物理マシンの購入費，物理マシンの運用にかかる電気料金，物理マシンを設置するための場所の確保／整備などである．

単一の物理マシンかつ単一のオペレーティングシステム (OS; Operating System) 上で，複数のクローラを実行する構成も考えられる．この構成は，ビルオーナーごとに物理マシンを用意する構成と比べて物理マシンを減らせるため，設備投資が増加する問題を解決できる．しかし，各クローラは，OS が提供するファイルシステムやネットワークなどの機能を共有することになるため，クローラ間の干渉が生じやすい．例えば，複数のクローラが同じポート番号を使用して通信を試み，通信に失敗する場合が考えられる．

設備投資が増加する問題を解決しつつ，各クローラの実行環境の独立性を保証するための方法として，計算機の仮想化技術がある [10, 55]． 以降，単に「仮想化技術」と記載する場合は，計算機の仮想化技術のことを指す．仮想化技術は，仮想的な CPU やネットワークインターフェースカードなどを備えた仮想マシン (VM; Virtual Machine) を，ソフトウェアで模擬する．各 VM では個別の OS を実行することが可能であり，その上でクローラを実行できる．あるクローラや OS における障害は，それを実行する VM 内に閉じるため，他の VM 上で動作するクローラへの影響を制限できる．また，各クローラは，個別のファイルシステムやネットワークを使用できる．そのため，例えば，共通のポート番号を使用して通信できるため，クローラ間の干渉を避けられる．さらに，VM のメモリやデバイスの状態などを保存するスナップショット機能を使えば，クローラのバックアップとリストアも容易になる [12].

ただし，VM 上で動作するアプリケーションの性能は，仮想化によるオーバーヘッドや，複数の VM を単一の物理マシン上で実行することにより低下する [5, 56]．クローラを含め，監視制御アプリケーションは高い信頼性が要求されるため [31, 57]，VM の利用がクローラの性能に与える影響を把握することは重要である． VM 上の監視制御アプリケーションの性能を評価した研究として文献 [28, 62] があるが，VM に割り当てる CPU などのリソース量が性能に与える影響や，並列実行する VM 数が性能に与える影響は評価していない．

本章では，VM に割り当てるリソース量や，並列実行する VM 数がクローラの

性能に与える影響を評価する．本章が明らかにする知見を以下にまとめる．

1. 物理マシンの CPU リソースに余裕がある状況でも，VM 数が増えると，CPU 競合により，クローラの性能が低下する．

2. VM 群が多くの CPU リソースを消費すると，VM を管理する仮想化ソフトウェアが CPU を使えなくなり，クローラの性能が低下する．

これらの知見に基づき，3 章で，監視制御アプリケーションを VM 上で実行する場合の性能を推定する手法を提案する．

以降，2.2 節で本章で想定する遠隔ビル管理システムを，2.3 節で仮想化技術の概要を説明する．2.4 節で関連研究について述べる．2.5 節で評価の内容を説明し，2.6 節では，監視制御アプリケーションを実行する VM の管理における課題を述べる．最後に 2.7 節でまとめる．

2.2　遠隔ビル管理システム

本節では，本章で想定する遠隔ビル管理システムの概要とクローラの要件について説明する．

2.2.1　クローラによる機器状態の計測

図 2.1 は，遠隔ビル管理システムとビル群のシステム構成である．遠隔ビル管理システムの事業者は，プライベートクラウド上に遠隔ビル管理システムを構築する．遠隔ビル管理システムはクローラや，監視点データを保存するデータベース (DB)，監視制御機能を実行するサーバ，ビル管理者用の操作端末である Human Machine Interface (HMI) を備える．図の見やすさを考慮して，それぞれ 1 つしか書いていないが，実際には複数個が存在しうる．遠隔ビル管理システムとビル群は WAN で接続される．本章ではこの WAN として，インターネットのようなベストエフォート型のネットワークではなく，通信品質が保証されたネットワークを想定する．

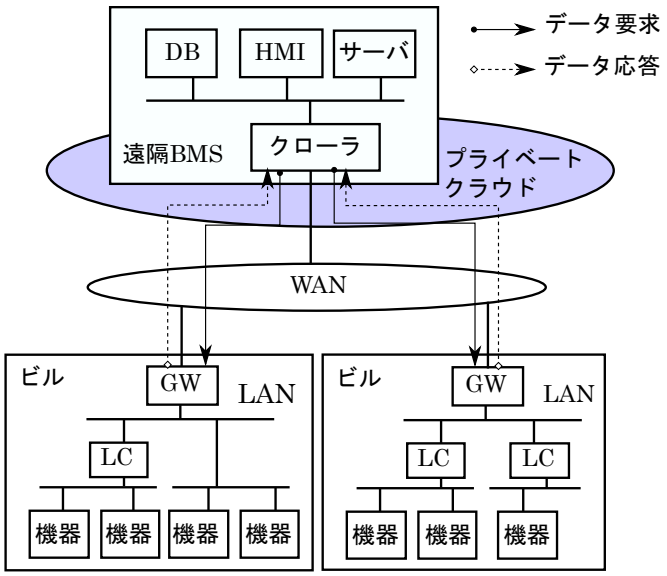


図 2.1: 遠隔ビル管理システムとビル群のシステム構成

クローラのデータ取得処理のシーケンスを図 2.2 に示す．クローラとゲートウェイは，BACnet/WS [7] や IEEE1888 [40]，oBIX [71] などの遠隔監視制御向けの通信プロトコルを使う [44, 79]．クローラは，監視点データを取得するため，ビルに設置されたゲートウェイにデータ要求を送信する．ゲートウェイはデータ要求を受けると，ローカルコントローラ (LC; Local Controller) にデータ要求を送信する．ゲートウェイとローカルコントローラは，ビル内の監視制御用プロトコルである BACnet [6] などを使い通信する [69]．ローカルコントローラは，データ要求とは関係なく，自身が収容する監視点の値を計測しており，データ要求を受けた時点で自身が把握している最新の監視点データを，データ応答としてゲートウェイに返す．ゲートウェイはデータ応答をクローラに返し，クローラは監視点データをデータベースに蓄積する．このように，クローラの処理は主にゲートウェイとの通信である．

監視点データを計測した時刻 (計測時刻) は，ゲートウェイが打刻することを想定する．ローカルコントローラや機器は，計測時刻を打刻する能力を持たない場合があるためである．例えば，BACnet の ReadProperty というサービスにより得られる監視点データには，計測時刻が付加されない．また，機器が計測時刻を打

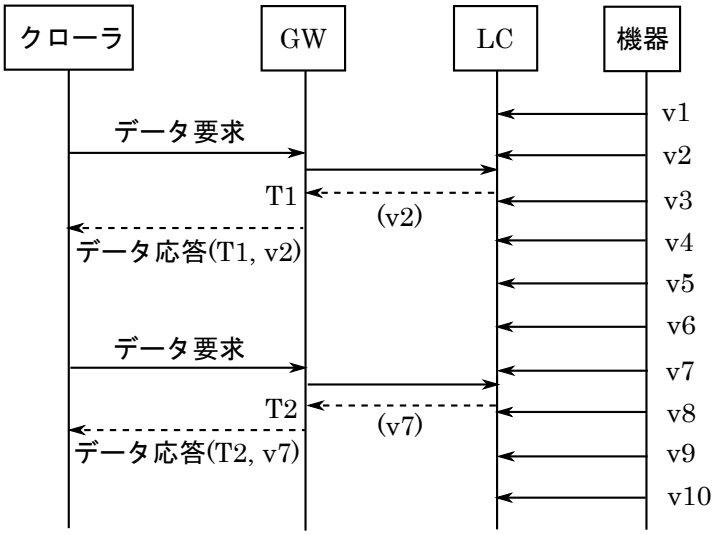


図 2.2: データ取得処理のシーケンス．T1 や T2 はタイムスタンプを，v1 や v2 は監視点データを表す．

刻する方法も考えられるが，多数の機器間で時刻同期を取る必要が発生する．他種多用かつ多数の機器間で時刻同期を行うことは，コストの観点から現実的ではない．

図 2.2 ではゲートウェイが 2 つの計測時刻 T1 と T2 を，機器の状態 v2 と v7 に，それぞれ打刻している．ゲートウェイが監視点データに打刻する計測時刻と，機器がその監視点データの状態であった時刻との間には誤差が生じる．しかし，ゲートウェイとローカルコントローラは専用の Local Area Network (LAN) により接続されるため，誤差のばらつきはマイクロ秒オーダーであり，監視制御機能の品質に影響を与えない．

また本論文では，クローラから通信を開始するリクエスト／レスポンス型の通信パターンを想定する．ビル側から遠隔ビル管理システムに対して監視点データを自律的に送信する通信パターンも考えられるが，両者は一長一短である．リクエスト／レスポンス型の通信パターンの長所は，いつ，どの監視点データを計測するかという設定情報を集中管理できる点である．そのため，設定情報の変更が容易であり，ゲートウェイやローカルコントローラなどのビル側の装置を簡素化できる．これらの長所は遠隔ビル管理システムの適用を容易にすることにつなが

ると考え，リクエスト／レスポンス型を想定する．

2.2.2 クローラの要件

クローラは，事前に定められた時刻に監視点データを計測し，監視制御機能に提供する必要がある．以降，事前に定められた時刻と，実際の計測時刻との差を，計測時刻誤差と呼ぶ．

データ要求が送信されてから監視点データが計測されるまでには，クローラとビルの間の WAN における通信遅延のばらつきや，ビル側の装置における処理時間のばらつきが発生しうる．しかし，通信品質を保証可能な WAN 回線を利用すれば，通信遅延のばらつきは抑えられる．また，ビル側の装置は，監視制御システムで使用される機器であるから，そもそも処理時間のばらつきは小さい．以上より本章では，計測時刻誤差に影響を与える箇所として，クローラを実行する物理マシン上の仮想化環境のみに着目する．

計測時刻誤差は小さいほうがよい．以下にその理由を述べる．

- 可視化機能は，監視点データが一定間隔で計測されていれば，ビル管理者にとって見やすいグラフ，すなわち，値が等間隔でプロットされたグラフを作成できる．
- 課金機能は監視点データから機器の稼働時間を計算し，テナントへの課金額を計算する．そのため，公正に課金額を計算するためには，一定間隔で稼働または非稼働を判定することが重要である．
- 監視点データが所定の時刻に計測されないと，フィードバック制御機能の品質が低下する．例えば，制御対象の機器の状態が不安定になる．

許容できる計測時刻の誤差は，監視制御機能の種類や，目指す品質に依存する．典型的には，空調や照明の状態を 1 秒間隔で計測する場合であれば，計測時刻誤差を 50 ミリ秒以下に抑えたい．また，火災報知機は 100 ミリ秒間隔で計測する場合があり，この場合，計測時刻誤差を 5 ミリ秒以下に抑えたい．このように，計測時刻誤差を許容範囲内で抑えることが，クローラの性能要件である．

またクローラは，性能要件を高い確率で達成する必要がある．本章では，IEC 61508 で定義される Safety Integrity Level (SIL) の連続動作モードのレベル 1 に基づき，99.999%以上の確率で性能要件を達成することを目標とする．SIL1 は「故障しない確率」を 99.999%以上と定義しているが，本論文では「性能要件を満たせない状態」も「故障」とみなす．監視制御システムでは，性能要件を満たせないことは，異常事態と同等だからである．以上より，99.999%以上の確率で，計測時刻誤差を許容範囲内で抑えることを，クローラの要件とする．

2.3　仮想化ソフトウェア

本章では，仮想化ソフトウェアの概要と，本章で使用する Xen の概要を説明する．仮想化ソフトウェアとは，VM の作成／削除／実行／停止などを行う機能を備えたソフトウェアのことである．近年，Docker[1]をはじめとするコンテナ技術が注目されているが，本論文では高可用性に関する機能が充実している VM を利用する．VM とコンテナの比較は 2.4 節で論じる．

2.3.1　仮想化ソフトウェアの種類

仮想化ソフトウェアには，OS 上のアプリケーションとして動作するスタンドアロン型の仮想化ソフトウェアと，OS の一機能または OS と物理マシンの間で動作するハイパーバイザ型の仮想化ソフトウェアが存在する．ハイパーバイザ型の仮想化ソフトウェアは，複数の VM を同時に実行できるため，本章ではハイパーバイザ型の仮想化ソフトウェアを想定する．

ハイパーバイザ型の仮想化ソフトウェアは，2 種類に分類できる．1 つは，VM を管理する機能のみに特化した仮想化ソフトウェア (仮想化特化型) である．例えば Xen [10] や VMware ESXi[2]がこれに相当する．もう 1 つは，Linux や Windows などの OS に，VM を管理する機能を追加した仮想化ソフトウェア (OS 一体型) である．例えば Linux の KVM [55] がこれに相当する．

[1] https://www.docker.com/
[2] https://www.vmware.com/products/esxi-and-esx.html

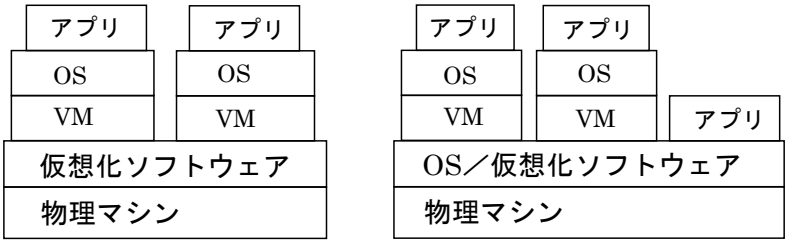


図 2.3: 仮想化特化型の仮想化ソフトウェア (左) と，OS 一体型の仮想化ソフトウェア (右)

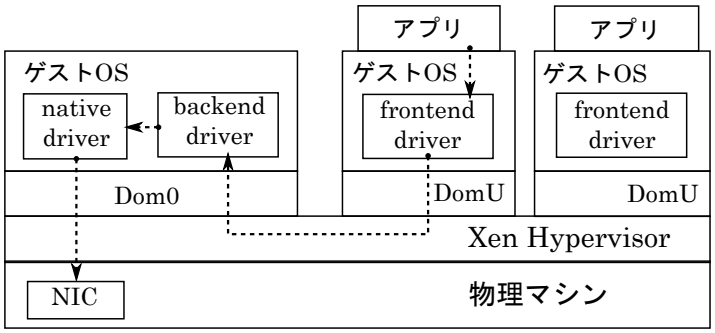


図 2.4: Xen のアーキテクチャ

仮想化特化型と OS 一体型の構成の比較を図 2.3 に示す．仮想化特化型の上で動作するプロセスは，全て VM である．一方，OS 一体型の場合は，VM だけでなく，一般的なアプリケーションも実行できる．仮想化特化型は，その機能を VM の管理に限定しているため，ソフトウェアを小さく維持することができ，ソフトウェアの信頼性を高めやすい．この特徴は，安定性を重視する監視制御アプリケーションに適していると言える．そこで本章では，仮想化特化型のハイパーバイザである Xen を使用することとする．

2.3.2 仮想化ソフトウェア Xen

Xen は Xen Project により開発されている仮想化ソフトウェアである．オープンソースであり，Google や Amazon Web Service，Intel，AMD などが開発に協力

している[3]．Xen をベースとした商用の仮想化ソフトウェアも存在する[4]．図 2.4 に Xen のアーキテクチャを示す．

完全仮想化と準仮想化

Xen は，完全仮想化 (Full Virtualization) と準仮想化 (Para Virtualization) の両方に対応している．完全仮想化とは，物理マシンの CPU などのハードウェアをソフトウェアにより模擬する仮想化の手法である．完全仮想化の場合，VM 上の OS (ゲスト OS) は，自身が VM 上で動いていることを知らずに動く．そのため，ゲスト OS がハードウェア (例えば CPU) を使おうとしたときに，その動作を仮想化ソフトウェアで検知し，本当の物理マシンのハードウェアを使うための命令に変換する必要がある．この変換処理によりアプリケーションの性能が低下するため，完全仮想化は，CPU の仮想化支援機能 (例: Intel VT) に依存する場合が多い．また，仮想化ソフトウェア向けに OS を修正する必要がないため，Windows のようにソースコードが公開されていない OS を，ゲスト OS として VM 上で実行できる．

準仮想化は，ゲスト OS が仮想化ソフトウェアの存在を意識して動作することで，ハードウェアを使うための命令を変換することによるオーバヘッドを削減する手法である．すなわち，ゲスト OS は，仮想化ソフトウェアがゲスト OS 向けに提供している準仮想化用の API を使用して動く．完全仮想化の場合と異なり，仮想化ソフトウェアはゲスト OS が発行するハードウェア使用のための命令を検知／変換する必要はない．そのため，準仮想化は完全仮想化よりも性能が高くなる傾向にある．ただし，仮想化ソフトウェアが提供する API を使用するように，ゲスト OS を修正する必要がある．例えば本章の評価で使用する Linux 3.13.0 は，Xen の準仮想化用の API に対応しているため，準仮想化の環境で動作できる．

準仮想化は完全仮想化と比べて，計算処理や通信処理の性能が高く，VM 数に対するスケーラビリティも高いことが知られている [14, 16, 18, 84]．

---

[3] https://www.xenproject.org/users/why-the-xen-project.html
[4] http://xenserver.org

Dom0 と DomU

Xen は 2 種類の VM を実行する．1 つは Dom0 と呼ばれる VM である．Dom0 は Xen 上に 1 つだけ存在する VM であり，DomU と呼ばれる他の VM の作成／起動／停止や，ハードウェアにアクセスするデバイスドライバを実行する権限を持つ．DomU は，Dom0 により作成される VM のことである．Xen のユーザは，一般的に，アプリケーションを DomU 上の OS 上で実行する．

図 2.4 の点線の矢印は，DomU 上のアプリケーションがデータの送信を行う場合のデータの流れを表している．アプリケーションが送信したデータは，DomU の frontend driver から Dom0 の backend driver へと渡される．そして Dom0 が，物理マシンに搭載されているネットワークインターフェースカードのデバイスドライバを実行し，データをネットワークへ送信する．

デバイスドライバは Dom 上の OS に実装されており，Xen 自身はデバイスドライバを含まない．そのため，Xen はその実装を小さく，単純に保ちやすい．また，デバイスドライバの障害が発生したとしても，その影響が Hypervisor や他の DomU に波及することを防ぎやすい．例えば，デバイスドライバの障害を検出して，初期化することが可能である [26]．安定性を重視しているという点で，監視制御アプリケーションに適した設計と言える．

CPU スケジューラ

CPU スケジューラとは，どの仮想 CPU を，どれだけの時間，どの物理 CPU で実行するかを計算する機能である．Xen は複数の CPU スケジューラを備えている．本章の評価で使用する Xen バージョン 4.6 のデフォルトの CPU スケジューラは credit スケジューラである．

credit スケジューラは定期的に，各 VM の仮想 CPU に一定量のクレジットを付与する．付与するクレジットは，weight というパラメータに応じて決定する．そしてクレジット残量が 0 より大きい仮想 CPU に優先的に，物理 CPU コアをラウンドロビンで割当てる．各仮想 CPU は物理 CPU を使用した時間に応じて，クレジットを消費する．そのため，あまり物理 CPU を使用しない仮想 CPU のクレジッ

ト残量は増加していく．クレジット残量が所定の閾値まで増加した場合，credit スケジューラはその仮想 CPU のクレジットの一部を破棄し，他の仮想 CPU に優先して物理 CPU を割り当てようとする．また，credit スケジューラは timeslice というパラメータを持つ．timeslice の単位は「ミリ秒」である．仮想 CPU は，最大で，timeslice で設定されている時間まで物理 CPU を連続して使い続けられる．timeslice のデフォルト値は 30 ミリ秒である．

credit2 スケジューラは，credit スケジューラとは異なる方針で各仮想 CPU にクレジットを付与するスケジューラである [22]．credit2 スケジューラは，仮想 CPU のクレジット残量が閾値に達しても，破棄せず，閾値に留める．そして，クレジット残量が多い仮想 CPU に優先し物理 CPU を割当てる．そのため，credit2 スケジューラは credit スケジューラと比べて，処理量が少ないアプリケーションに対して優先して物理 CPU を割り当てる傾向にある．

他には，実験段階の CPU スケジューラとして，Real-Time Deferrable Server (RTDS) スケジューラがある [92, 93]．RTDS スケジューラは，リアルタイム性を重視したスケジューラであり，各 VM の動作時間を厳密に制御することを目指している．

CPU affinity

どの物理 CPU でどの仮想 CPU を実行するかは，基本的に，CPU スケジューラの方針に依存して決まる．しかし，CPU affinity という機能を使用することで，仮想 CPU を実行する物理 CPU を明示的に指定できる．

Xen では hard affinity と soft affinity を使用できる．hard affinity を使用すると，ある仮想 CPU を実行する物理 CPU を指定できる．これにより，ある VM に特定の物理 CPU を占有させることが可能となる．soft affinity を使用すると，ある仮想 CPU が，特定の物理 CPU で実行されやすくなる．これにより，物理 CPU 間の負荷を緩く調整できる．

表 2.1: 可用性の観点における VM とコンテナの比較 (文献 [60] から抜粋)

機能	VMware	Citrix XenServer	Docker/LXC	OpenVZ
冗長化	あり	あり (実験段階)	なし	なし
ライブマイグレーション	あり	あり	なし	あり
チェックポイント	あり	あり	なし	あり
障害検知	あり	あり	なし	なし
フェイルオーバー	あり	あり	なし	なし

2.4 関連研究

計算機の仮想化，すなわち VM は，クラウドコンピューティングの基礎をなす技術であり，Web サーバや DB サーバなどを中心に幅広く利用されている[5]．VM を利用することで物理マシン数を減らし，障害の波及を制限しつつ，物理マシンにより消費される電力やスペースを削減することが目的である．VM の性能や安定性の向上に伴い，近年は，監視制御システムに適用するための研究が行なわれている [29]．監視制御システムのレガシーなハードウェアを仮想化することでシステムの延命を図る取り組みや [12]，テスト時に活用しようとする取り組みがある [13]．Programmable Logic Controller (PLC) や CAN デバイスなどの具体的な監視制御アプリケーションを仮想化する取り組みもある [28, 54, 80]．

仮想化を実現するソフトウェアとしては Xen [10] や KVM [55]，VMware ESXi などがある．2.3.2 項で述べたように，安定性を重視した設計であることから，本章では Xen を使用する．

計算機の仮想化技術と類似する技術として，近年，Docker をはじめとするコンテナ技術が注目されている [21]．コンテナは VM と比べて性能に対するオーバーヘッドが小さいことが知られている [24, 86]．しかし，ライブマイグレーションやチェックポイントなどの高可用性に関する機能が充実している VM のほうが，監視制御システムの仮想化には適していると考える [60]．

[5] https://www.f5.com/pdf/reports/enterprise-virtualization.pdf

監視制御アプリケーションを VM 上で実行する場合の懸念の一つは性能である．VM は CPU をはじめとするハードウェアをソフトウェアで模擬するため，処理性能が低下する．仮想化がアプリケーションの性能に与える影響は広く研究されており，最大スループットに注目しているものが多い．例えば，文献 [5] は iPerf[6]を用いて，Xen の通信スループットを評価している．また，文献 [8, 56] は Cachebench[7] や UnixBench[8]などのベンチマークツールを用いて，計算処理のスループットを評価している．いずれもの文献も，複数の CPU コアを利用すると，L1 キャッシュにおけるキャッシュミスが増えるため，1CPU コアの場合と比べて CPU コアあたりの性能が低下することを示している．Web サーバや DB サーバ，IP 電話などのアプリケーションレベルの性能の評価も行われている [4, 64, 75, 90]．

スループットは重要な性能指標であるが，監視制御アプリケーションの場合，処理のスループットよりもタイミングが重要である．文献 [28] は監視制御アプリケーションの 1 つである PLC を仮想化した場合の処理のタイミングを評価している．文献 [62] は，監視制御アプリケーションを想定して VM の通信遅延を評価している．しかし，どちらの文献も，VM に割り当てるリソース量や，単一の物理マシンで同時に実行する VM 数が処理のタイミングに与える影響を評価していない．また，同時に実行する VM 数が，物理マシンの CPU コア数を大きく超える状況における評価も実施していない．同時に実行する VM 数がアプリケーションの性能に与える影響は文献 [39, 56, 83] などで示されているが，ベンチマークツールを使用した評価であり，監視制御アプリケーションの性能への影響は評価していない．

本章では，監視制御アプリケーションの具体例としてクローラ [43] を想定し，監視点データの計測時刻誤差を評価する．そして，VM に割り当てる CPU リソース量や，単一の物理マシン上で同時に実行する VM 数が計測時刻誤差に与える影響を評価する．また，credit2 スケジューラや CPU affinity を使用した評価も実施する．credit2 スケジューラを使用した性能評価は文献 [37, 88] で行われているが，credit2 スケジューラを使用して監視制御アプリケーションの性能を評価した研究は，本論文が初めてである．

---

[6]https://iperf.fr/

[7]http://icl.cs.utk.edu/llcbench/cachebench.html

[8]https://www.ostechnix.com/unixbench-benchmark-suite-unix-like-systems/

2.5 仮想化されたクローラの性能評価

本節では，VM 上で動作するクローラの性能評価について述べる．

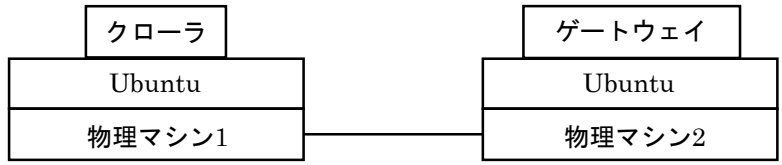
2.5.1 評価の目的

遠隔ビル管理システムの運用者は，単一の物理マシン上で，できるだけ多くのクローラ／VM を実行することで，ビル群を管理するために必要な物理マシン数を減らし，設備投資を抑えたい．ただし，各クローラは 2.2.2 項で述べた要件を満たす必要がある．各物理マシンの性能や各クローラの処理負荷は同一とは限らないため，運用者が各クローラをどの物理マシンに配置するかを考えることは手間である．そのため，各クローラの性能要件を満たしつつ，物理マシンのリソース利用率を最大化してくれる VM 管理手法があることが望ましい．このような VM 管理手法は，Web サーバや DB サーバを対象として研究されている [19, 33, 67, 68]．しかし，クローラのような監視制御アプリケーションを対象としたものは存在しない．本評価では，クローラを対象とした VM 管理手法の実現に向けて，クローラの性能に影響を与える要因を明確にすることを目的とする．

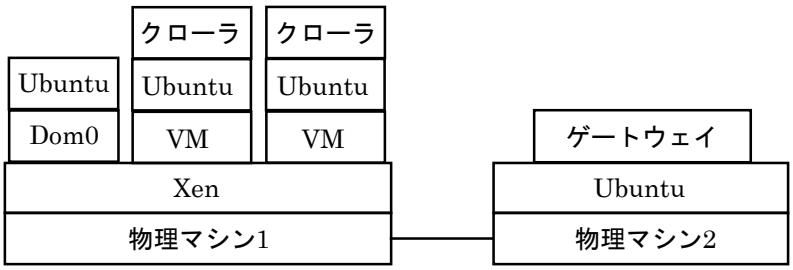
2.5.2 評価の環境

図 2.5 に，仮想化せずにクローラを実行する評価環境（ネイティブ環境）と，VM 上でクローラを実行する評価環境（仮想化環境）を示す．評価には 2 つの物理マシンを用いる．一方でクローラを動かし，もう一方でビル群のゲートウェイを模擬する．2 つの物理マシンは 1000BASE-T で接続する．以降，単に「VM」と述べる場合は DomU のことを指す．

クローラは，事前に与えられた計測スケジュールに基づき，データ要求を送る．計測スケジュールとは，いつ，どの監視点データを計測するかを定義した情報である [43]．データ要求には BACnet/WS [7] の getValues リクエストを使う．BACnet/WS は HTTP をベースとするビル管理システム向けの通信プロトコルである．



(a) 仮想化を使わない環境 (ネイティブ環境)



(b) 仮想化を使う環境 (仮想化環境)

図 2.5: 評価環境

BACnet/WS における全てのリクエストとレスポンスは Extensible Markup Language (XML) により表現される．

ビル群のゲートウェイは，Apache HTTP サーバで模擬する．クローラとゲートウェイの間の通信には，ビルごとに片道 10 ミリ秒の遅延をエミュレーションする．10 ミリ秒という値は，神奈川県にある研究所と，Amazon EC2 の東京リージョンとの間の通信遅延の平均値と同等の値である．この通信遅延は，`ping` コマンドを使用して計測した．2 つの物理マシン間の時刻同期には Network Time Protocol (NTP) を使用する．

仮想化用の物理マシンは Intel(R) Xeon(R) 1.80GHz CPU (8 コア)，32GB メモリを備える．仮想メモリアドレスと物理メモリアドレスの変換処理を効率化する Extended Page Tables (EPT) 機能は有効化する．また，ビルゲートウェイ用の物理マシンは Intel(R) Xeon(R) 2.60GHz CPU (32 コア)，80GB メモリを備える．物理マシンの OS として Ubuntu 16.04 (Linux 4.4.0) を，VM の OS として Ubuntu 14.04 (Linux 3.13.0) を使用する．Xen のバージョンは 4.6.0 とする．Apache HTTP サーバはバージョン 2.4.10 を使用する．

クローラの実装言語は C 言語である [43]．実際の環境では，クローラは計測した監視点データを，データベースや監視制御機能に渡す．しかし，データベース

表 2.2: 予備実験の結果のまとめ

予備実験の内容	結果
完全仮想化と準仮想化の性能比較	準仮想化が優位
credit スケジューラと RTDS スケジューラの性能比較	RTDS スケジューラが劣位
クローラのリソース消費の傾向	CPU バウンド

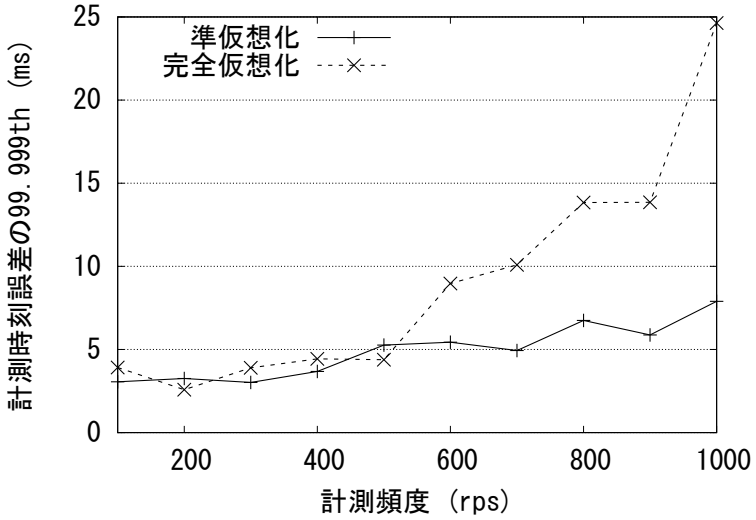


図 2.6: 完全仮想化と準仮想化の計測時刻誤差の比較

サーバなどの性能に起因するクローラの計測時刻への影響を除外するため，本評価で用いるクローラは，計測した監視点データを破棄する．

2.5.3 予備実験の結果

本章で考察すべき対象を絞り込むために実施した予備実験の内容と結果をまとめる．予備実験の結果を表 2.2 に示す．

完全仮想化と準仮想化の性能比較 Xen は仮想化の方法として完全仮想化と準仮想化をサポートしているため，両者におけるクローラの計測時刻誤差を比較した．本評価ではクローラ／VM は 1 つとして，監視点データの計測頻度 (rps; Request

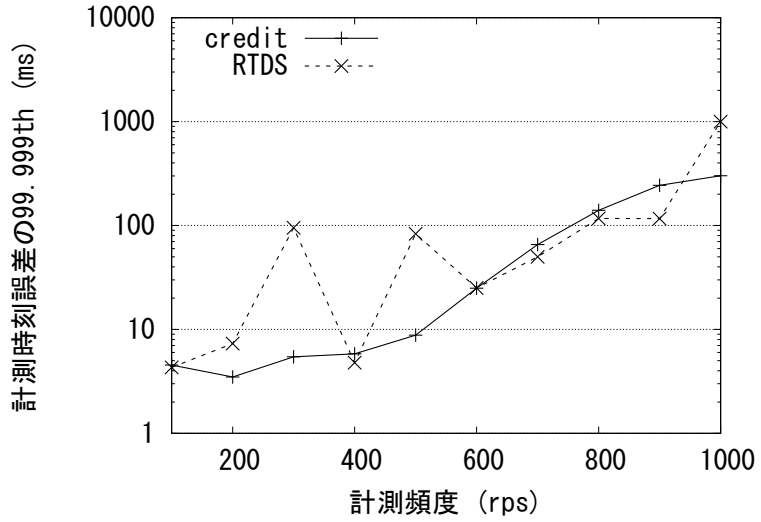


図 2.7: credit スケジューラと RTDS スケジューラの比較

per Second) を変化させた．計測時刻誤差の 99.999 パーセンタイル値を計算するために，評価期間は 20 分とし，10 万回以上の通信処理をクローラに行わせた．

図 2.6 に，評価結果を示す．計測頻度が 600rps 以上の場合に差が生じた．準仮想化のほうが，計測時刻誤差を数十ミリ秒ほど小さく抑えられたため，以降は準仮想化を使用する．

credit スケジューラと RTDS スケジューラの性能比較　Xen バージョン 4.6 におけるデフォルトの CPU スケジューラである credit スケジューラと，実験段階のリアルタイム CPU スケジューラである RTDS スケジューラの性能を比較した．本評価ではクローラ／ VM は 1 つとして，監視点データの計測頻度を変化させた．計測時刻誤差の 99.999 パーセンタイル値を計算するために，評価期間は 20 分とし，10 万回以上の通信処理をクローラに行わせた．

図 2.7 に，評価結果を示す．図から，RTDS スケジューラは credit スケジューラよりも，計測時刻誤差が不安定であることがわかる．例えば，計測頻度が 300rps や 500rps の場合に，計測時刻誤差が約 100 ミリ秒に増大している．また，1000rps の場合に，計測時刻誤差が約 1 秒になっている．RTDS は実験段階の CPU スケジューラであり，特に Dom0 や DomU が複数の仮想 CPU を持つ場合の実装が未

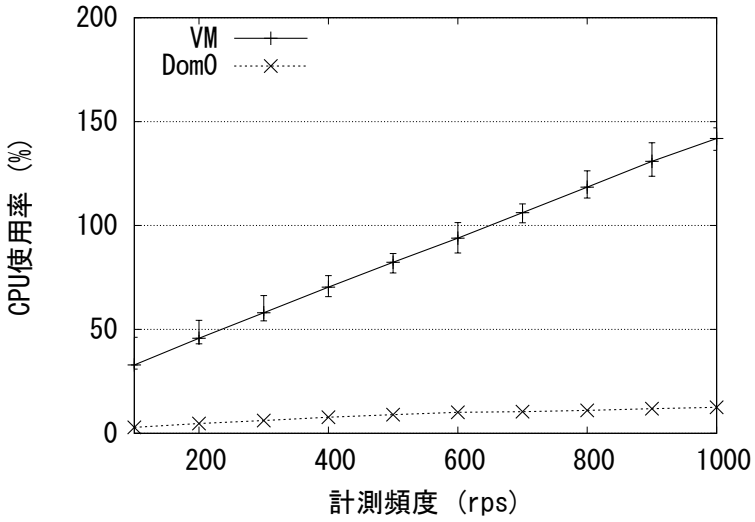


図 2.8: 平均 CPU 使用率 (1 秒計測)

成熟である．具体的には，あるドメイン (Dom0 または DomU) の仮想 CPU 群が，互いに互いが使用している物理 CPU を奪い合い，それにより処理が遅延する状況が生じる．この状況は，各仮想 CPU が，現在使用している物理 CPU を使用し続ければ避けられるが，そのような実装になっていない．以上より，以降の評価では RTDS スケジューラは使わない．

クローラのリソース消費の傾向 クローラを実行する VM のリソース消費の傾向を分析した．本評価では，クローラ／ VM は 1 個とし，VM の仮想 CPU は 4 個，Dom0 の仮想 CPU は 8 個とした．CPU スケジューラは credit スケジューラとし，CPU affinity は設定せずに評価した．`xentop` コマンドを使用し，VM の CPU 使用率を 1 秒ごとに計測した．計測時刻誤差の 99.999 パーセンタイル値を計算するために，評価期間は 20 分とし，10 万回以上の通信処理をクローラに行わせた．

図 2.8 に計測した CPU 使用率を示す．図中のエラーバーは，計測期間 (1 分) における CPU 使用率の最大値と最小値である．図に示すように，クローラを実行する VM の CPU 使用率はデータ要求の計測頻度に比例することや，CPU 使用率のばらつきは 20%程度であることを確認した．また，クローラ停止時の VM の CPU 使用率は約 0.1%であることを確認した．つまり，クローラが停止している時，VM

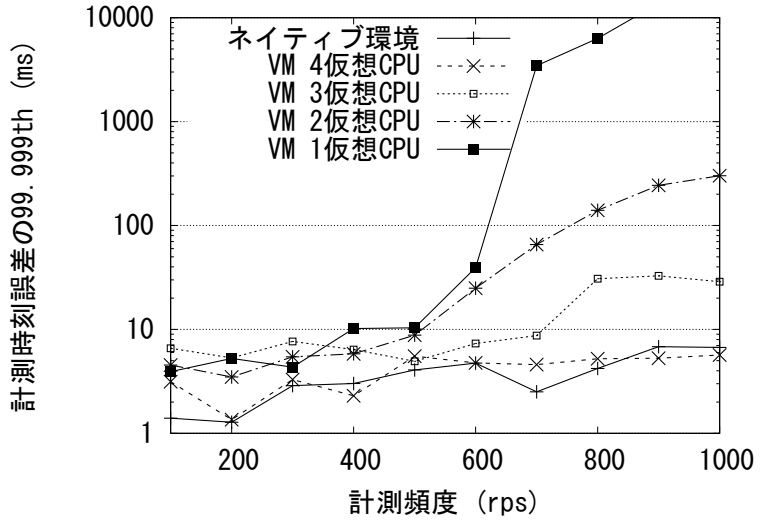


図 2.9: ネイティブ環境と仮想化環境の性能比較

はほとんど CPU を使用していない．CPU 以外のリソースであるネットワークやメモリの使用量も，計測頻度に比例する傾向を示した．ただし，CPU 以外のリソースの使用量は，絶対値としては小さかった．例えば，計測頻度が 1000rps の時点の通信量は約 600Kbps であった．また，クローラのディスク IO は，起動直後の計測スケジュールの読み込み処理と，動作ログをファイルに出力する際の書き込み処理のみであり，小さかった．したがって，クローラの処理は CPU バウンドであると言える．以降，CPU リソースに着目して評価および考察する．

2.5.4　性能評価の結果

本項では，まず仮想化によるオーバーヘッドを評価し，次に複数 VM を同時に実行することが性能に与える影響を評価する．そして，credit スケジューラと credit2 スケジューラを比較し，最後に CPU affinity が性能に与える影響を評価する．

仮想化によるオーバーヘッド

仮想化がクローラの性能に与える影響を評価する．クローラ／VM は 1 個とし，VM の仮想 CPU は 1 個から 4 個とする．仮想 CPU が少ないほどクローラの性能

は低下すると予想できるが，その低下の度合いを把握するために，仮想 CPU の数を変化させる．また，Dom0 が通信処理のボトルネックとならないよう，Dom0 の仮想 CPU は 8 個とする．CPU スケジューラはデフォルトの credit スケジューラとする．CPU affinity は使用しない．

監視点の数は 100 から 1000 とし，計測周期は 1 秒とする．すなわち，クローラの計測頻度は 100rps から 1000rps となる．中小規模ビルが備える監視点数は数十点から数百点であり，かつ，BACnet/WS の getValues リクエストは，1 リクエストで N 個の監視点データを取得できる[9]．したがって 1000rps は数棟から数百棟の中小規模ビルの監視に相当し，評価の設定として十分に現実的だと考える．クローラは 99.999%以上の確率で正しくデータを計測する必要があるため，計測時刻誤差の 99.999 パーセンタイル値を評価軸とする．99.999 パーセンタイル値を計算するために，評価期間は 20 分とし，10 万回以上の通信処理をクローラに行わせる．

図 2.9 に評価結果を示す．縦軸は計測時刻誤差の対数軸である．いずれの場合も，計測頻度の増加にともない計測時刻誤差が増加する傾向にある．単位時間あたりの通信回数が増えることで，計画通りのタイミングで通信を行なえる確率が減るためである．また，仮想 CPU が 4 個の場合の仮想化環境における性能と，ネイティブ環境における性能に大差はないことから，本評価で用いたクローラにとって 4 個の仮想 CPU は十分な CPU リソースであると言える．一方，仮想 CPU が 1 個の場合は，700rps の時点で誤差が増大した．図 2.8 から，700rps の時点で VM の CPU 使用率は 100%を超えることがわかる．つまり，1 個の仮想 CPU では 700rps の通信処理を行いきれず，誤差が増大した．

仮想 CPU が 2 個の場合は，1000rps の時点で誤差が約 300 ミリ秒となった．図 2.8 によれば，1000rps の場合の平均 CPU 使用率は約 140%であり，2 個の仮想 CPU，すなわち 200%の CPU リソースは十分なように思える．しかし図 2.8 に示した値は，`xentop` コマンドを用いて 1 秒間隔で計測した平均値であり，1 秒よりも短い期間において 200%以上の CPU を消費している可能性がある．これを明らかにするため，VM の CPU 使用率をミリ秒間隔で計測するツールを実装した[10]．本

[9] IEEE1888 [40] も同様の機能を備える．
[10] `xentop` コマンドは，ミリ秒間隔で CPU 使用率を計測できない．

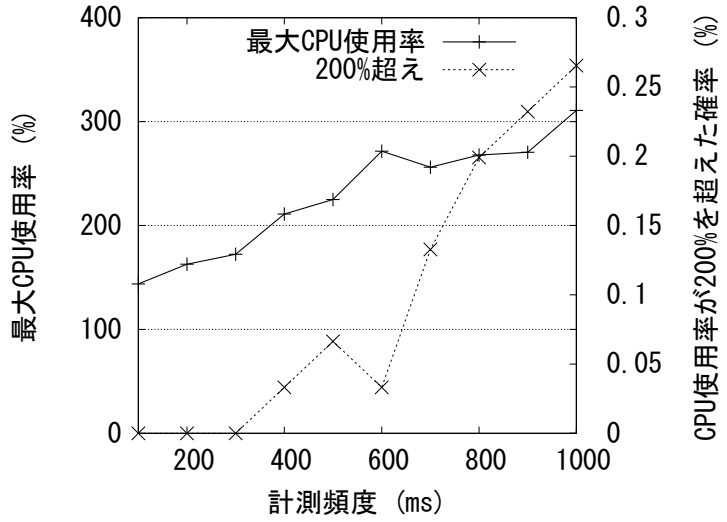


図 2.10: (100 ミリ秒ごとに CPU 使用率を計測した場合の最大 CPU 使用率

ツールはミリ秒単位の計測間隔を引数に取る．そして，指定された間隔で Xen の xc_domain_get_cpu_usage 関数を呼び出すことで各 VM の CPU 消費時間を取得し，CPU 消費時間と計測間隔から CPU 使用率を計算する．

実装したツールで，VM の CPU 使用率を 100 ミリ秒間隔で計測した．VM が真に必要とする CPU リソースを把握するため，十分な CPU リソース，すなわち 4 個の仮想 CPU を VM に割り当てた．結果を図 2.10 に示す．実線は計測期間中の最大 CPU 使用率を，点線は CPU 使用率が 200%を超えた時間の割合を示している．計測頻度が 400rps の時点で最大 CPU 使用率は 200%を超えている．つまり，100 ミリ秒単位で見ると，200%以上の CPU リソースが必要な時間帯が生じている．その時間帯において，仮想 CPU が 2 個では，CPU リソースが足りないため，クローラの処理が遅れ，計測時刻誤差が増加する．CPU リソースが足りない時間帯の割合は，1000rps の場合でも 1%以下だが，計測時刻誤差の 99.999 パーセンタイル値は増加する．

本項の評価により，十分な CPU リソースを VM に割り当てれば，ネイティブ環境と同等の性能を達成できることを確認できた．一方，VM に割り当てる CPU リソースを，1 秒間隔で計測した CPU 使用率に基づいて決定すると，計測時刻誤差が数百ミリ秒まで増加する場合があることがわかった．監視点を 1 秒間隔で計測

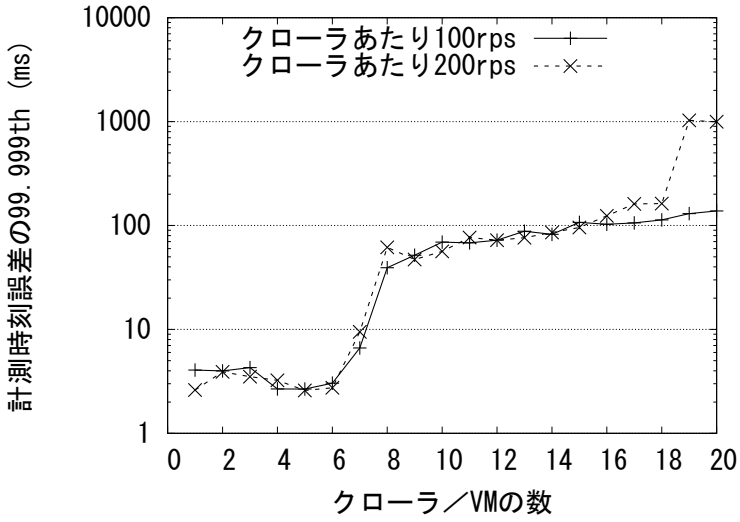


図 2.11: 複数 VM を同時実行した場合の VM 数と計測時刻誤差の関係

する場合，数百ミリ秒の計測時刻誤差は許容できない．許容される計測時刻の誤差と同等の間隔で CPU 使用率を計測し，その結果に基づいて割り当てる CPU リソース量を決定すべきである．

VM 数が計測時刻誤差に与える影響

複数の VM を実行する場合の性能を評価する．多くの VM を実行して評価するために，各クローラ／VM の計測頻度は小さく設定する．ここでは，各クローラが数棟のビルを監視する状況を想定し，計測頻度として 100rps と 200rps の二通りを設定して評価する．各 VM には 1 個の仮想 CPU を割り当てる．Dom0 の仮想 CPU は 8 個とする．CPU スケジューラは credit スケジューラとする．CPU affinity は使用しない．

図 2.11 に VM 数と計測時刻誤差の関係を示す．どちらの場合も，VM 数が 8 個の時点で計測時刻誤差が増大している．各クローラの計測頻度は 100rps または 200rps であるから，図 2.8 によれば，各 VM が必要とする CPU リソースは 50%以下である．本評価に用いた物理マシンは 8 コア，すなわち 800%の CPU リソースを備えるから，8 個の VM に対しては十分だと思える．

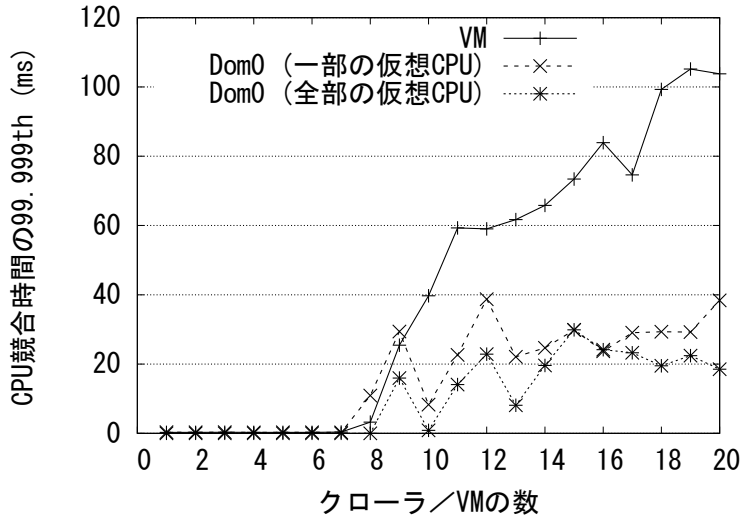


図 2.12: 複数 VM を同時実行した場合の VM 数と CPU 競合時間の関係

この結果の原因を明確にするために，xentrace と xenalyze を使用して，各 VM の CPU 競合状態を分析した．CPU 競合状態とは，VM の仮想 CPU が物理 CPU を使用したいのに，使用できずに待機している状態である．分析時の計測頻度は 100rps とした．

分析の結果を図 2.12 に示す．縦軸は CPU 競合状態が継続した時間 (CPU 競合時間) の 99.999 パーセンタイル値である．Dom0 は 8 個の仮想 CPU を持つため，その一部が CPU 競合状態であった時間と，全ての仮想 CPU が競合状態であった時間をそれぞれ示している．図 2.12 から，VM 数が 8 個の時点から CPU 競合時間が増加していることがわかる．特に，Dom0 の一部の仮想 CPU が CPU 競合状態であった時間が約 10 ミリ秒まで増加している。Xen には VM (DomU) 以外に Dom0 が存在する．評価で使用した物理マシン 1 は 8 コアであるため，Dom0 と 8 個の VM の全てを同時に実行できない．つまり，CPU 競合時間が生じる．CPU 競合状態の間，クローラ／ VM は動作できないため，計測時刻誤差は増加する．

Dom0 の全ての仮想 CPU が CPU 競合状態であった時間は，約 30 ミリ秒以下に留まっている．本項の評価では，credit スケジューラの timeslice パラメータの値はデフォルトの 30 ミリ秒としたため，Dom0 や VM は物理 CPU を最大で 30 ミリ秒

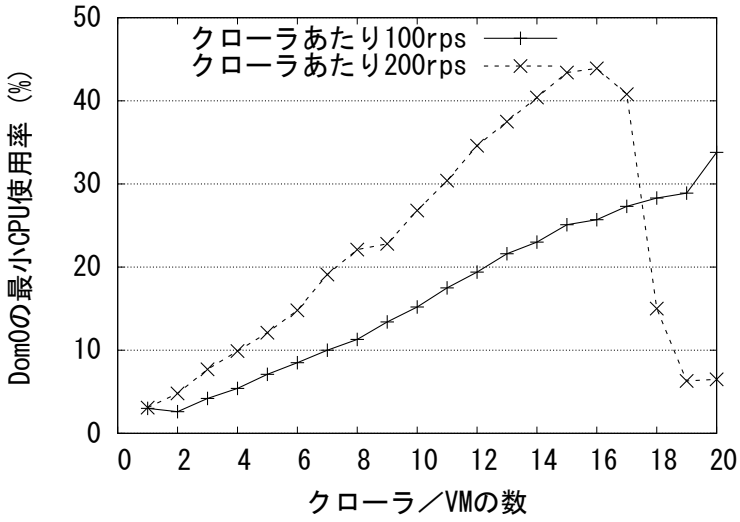


図 2.13: 複数 VM を同時実行した場合の Dom0 の CPU 使用率の最小値

間占有する[11]．また，本項の評価では Dom0 に 8 個の仮想 CPU を割り当てたため，Dom0 は VM と比べて物理 CPU を取得しやすい．したがって，連続する timeslice において Dom0 が物理 CPU を取得できないという状況は発生しづらく，Dom0 の CPU 競合時間は timeslice の設定値である 30 ミリ秒よりも長くならなかったものと考える．

各クローラの計測頻度が 200rps の場合は，VM 数が 19 個の時点で再び計測時刻誤差が増大し，1 秒を超えた (図 2.11)．VM 数が 19 個のとき，全 VM と Dom0 の CPU 使用率の平均値の合計は 722%であった．平均で約 78%の余裕があるにもかかわらず，計測時刻誤差が増大した理由は，Dom0 が CPU を使用できない時間帯が増えたためである．図 2.13 は，評価期間中の，Dom0 の CPU 使用率の最小値である．この CPU 使用率は `xentop` を用いて 1 秒間隔で計測した．計測頻度が 100rps の場合，Dom0 の CPU 使用率は常に増加傾向を示している．図 2.4 に示すように，全ての通信処理は Dom0 を経由して行われるため，Dom0 の CPU 使用率がクローラ／VM 数に比例する結果は妥当である．一方，200rps の場合は，VM 数が 17 個の時点で減少に転じ，VM 数が 19 個と 20 個の時点で約 6%となった．この結果は，

[11]実行するべき処理がない場合は，30 ミリ秒に達する前に，他の VM に物理 CPU を引き渡す可能性はある．

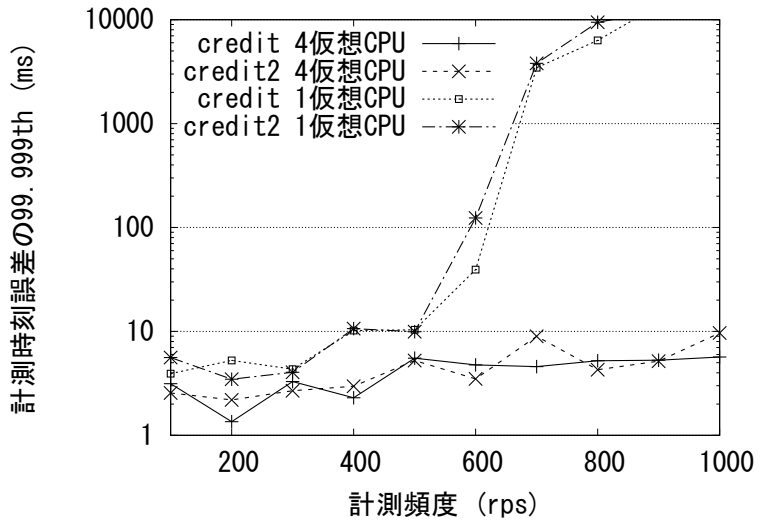


図 2.14: 単一 VM を credit スケジューラと credit2 スケジューラで実行した場合の性能比較

VM 群の処理の総量が増加すると，Dom0 が CPU をほぼ利用できない時間帯が発生し，その影響で通信処理のタイミングが乱れ，計測時刻誤差が増大することを示している．

本項の評価結果から，複数の VM を実行する環境では，計測時刻誤差が増大する箇所が 2 点存在することがわかった．1 点は，VM 群の処理に必要な CPU リソースの総量が増え，Dom0 に十分な CPU リソースを割り当てられなくなったときである．Dom0 は全ての通信処理に関与するため，計測時刻誤差が増大する．もう 1 点は，VM 群により使用される仮想 CPU の数が，物理 CPU の数を超えたときである．この時点から CPU 競合が発生し，計測時刻誤差が増大する．CPU 競合時間は数十ミリ秒に及ぶため，監視制御機能の品質に影響を与えうる．複数の VM を実行する場合は，各 VM の処理量が小さい場合でも，CPU 競合時間を推定し，監視制御機能の品質に影響が生じるか判断すべきである．

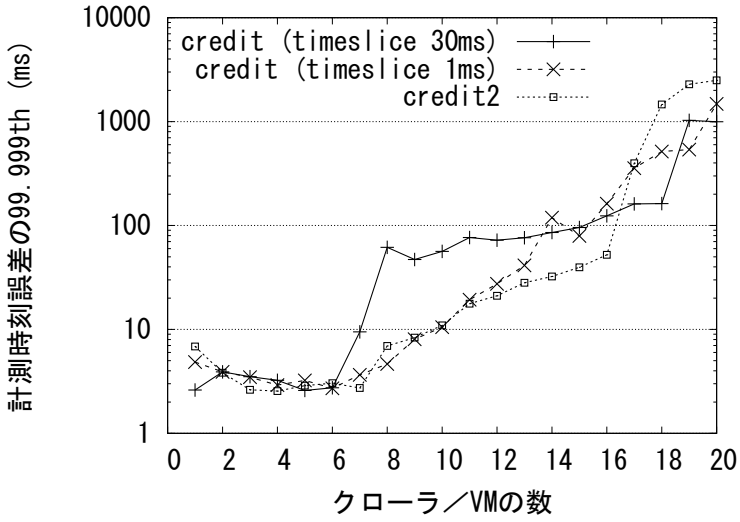


図 2.15: 複数 VM を credit スケジューラと credit2 スケジューラで実行した場合の性能比較

CPU スケジューラが計測時刻誤差に与える影響

仮想化によるオーバーヘッドや，VM 数が計測時刻誤差に与える影響について，credit スケジューラと credit2 スケジューラを比較する．

図 2.14 は，単一 VM を実行した場合の計測時刻誤差の比較である．計測頻度や仮想 CPU 数を変化させて比較したが，両者に大きな差は見られなかった．仮想 CPU 数が 1 個かつ 600rps の場合は，約 60 ミリ秒の差が生じているが，この差はスケジューラの違いによるものではないと考える．計測頻度が 600 の場合，CPU 使用率はときに 100%を超えるため (図 2.8)，仮想 CPU が 1 個では計測時刻誤差がばらつきやすく，かつ，計測時刻誤差の 99.999 パーセンタイル値を比較しているため生じた差であると考える．したがって，CPU リソースに余裕がある状況では，両スケジューラの差はないと言える．

図 2.15 は，複数 VM を実行した場合の計測時刻誤差の比較である．前項で述べたように，credit スケジューラの timeslice パラメータのデフォルト値は 30 ミリ秒であり，これが CPU 競合時間の増加につながっている．そこで，timeslice を 1 ミリ秒とした credit スケジューラの性能も合わせて評価した．各クローラの計測頻度

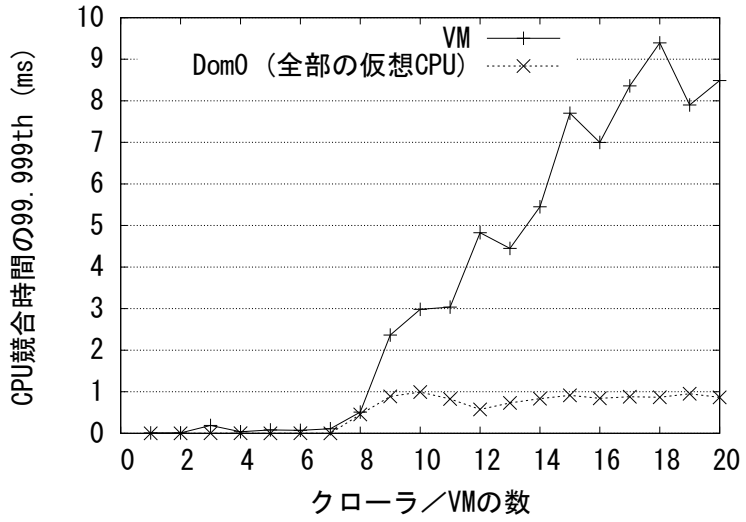


図 2.16: 複数 VM を同時実行した場合の VM 数と CPU 競合時間の関係 (timeslice は 1 ミリ秒)

は 200rps とした．各 VM には 1 個の仮想 CPU を割り当てた．Dom0 の仮想 CPU は 8 個とした．CPU affinity は使用しなかった．

図 2.15 から，timeslice が 1 ミリ秒の場合は，VM 数が 8 個の時点でも，計測時刻誤差を 10 ミリ秒以下に抑えられることがわかる．これは，想定通り，timeslice を短くすることで CPU スケジューリングを頻繁に実施するようになり，CPU 競合時間を短縮できたためである．`xenalyze` で CPU 競合時間を分析した結果を図 2.16 に示す．全体の傾向は timeslice が 30 ミリ秒の場合 (図 2.12) と同じだが，絶対値が減少していることを確認できた．

VM 数が 14 個以上になると，timeslice パラメータの値による計測時刻誤差の差はなくなった．timeslice を小さくすると，CPU スケジューリングの実行頻度が増すため，CPU スケジューリングによるオーバーヘッドが増える．VM 数が 8 個のときは，物理マシンの CPU リソースに余裕があるため，オーバーヘッドが増えても，CPU 競合時間の短縮による効果が大きく，計測時刻誤差を抑えられた．しかし，VM 数が増えると，CPU スケジューリングに必要な処理量は増加する．各 VM の情報を for ループで走査しながら確認する処理が含まれるためである．情報とは，例えば，各 VM の仮想 CPU のクレジット残量である．つまり timeslice が小さい

ほど，VM 数が増えたときの，CPU スケジューリングによるオーバーヘッドの増加率が高くなる．その分，CPU を通信処理に使用できる時間が減るため，通信処理が遅れる．このような理由から，VM 数が増えた場合に，timeslice による計測時刻誤差の差がなくなったと考える．

credit2 スケジューラの CPU スケジューリングの間隔は，500 マイクロ秒から 2 ミリ秒の間である．そのため，credit2 スケジューラの性能は，timeslice を 1 ミリ秒に設定した credit スケジューラの性能と似ている．ただ，credit2 スケジューラの場合は，VM 数が 16 個の時点まで，計測時刻誤差の増大を抑えられた．`xenalyze` で分析したところ，各 VM の CPU 使用時間は，credit2 スケジューラのほうが 10%ほど大きかった．2.3.2 項で述べたように，credit2 スケジューラは，使い切れなかったクレジットを破棄しない．これにより，クレジットを破棄された影響で CPU を使用できない状況がなくなるため，各 VM の CPU 使用時間が長くなる．つまりクローラはより多くの処理を行えるようになるため，credit スケジューラよりも計測時刻誤差を抑えられる．一方，クローラ／VM が 18 個以上においては，credit スケジューラよりも計測時刻誤差が大きくなった．credit2 スケジューラは，credit スケジューラよりもクローラ群に多くの CPU リソースを与えるため，Dom0 が CPU を使用できなくなりやすいのだと考える．

本項の評価結果から，CPU スケジューリングの頻度が高いほど，計測時刻誤差を抑えられることがわかった．credit スケジューラの場合は timeslice の値を変更する必要がある．credit2 スケジューラは，CPU スケジューリングの頻度が高く，さらに仮想 CPU のクレジットを破棄しないため，credit スケジューラよりも誤差を抑えられる．ただし，頻繁な CPU スケジューリングは CPU 切り替え処理のオーバーヘッドを増加させるため，監視制御機能が許容する計測時刻誤差に応じて，適切に CPU スケジューリングの頻度を設定すべきである．

CPU affinity が計測時刻誤差に与える影響

CPU affinity が性能に与える影響を評価する．まず soft affinity による影響を評価し，次に hard affinity による影響を評価する．

soft affinity を使用した評価では，各 VM の仮想 CPU が，特定の CPU ソケット

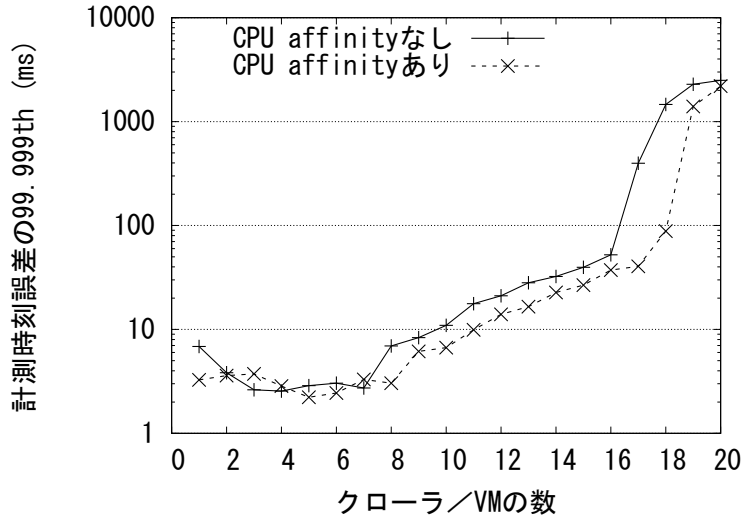


図 2.17: CPU hard affinity の有無の比較

に含まれる CPU コアにより優先して実行されるよう設定して評価する．評価に用いた物理マシン A は，2 個の CPU ソケット (4 コアずつ) を備えており，それぞれが L3 キャッシュを備えている．そこで，各 VM の仮想 CPU が，どちらか一方の CPU ソケットを優先するように soft affinity を設定した．このとき，各ソケットに割り当てる仮想 CPU の数がなるべく均等になるように設定した．評価の結果，soft affinity の適用有無による性能の変化は見られなかった．クローラの処理はデータ取得のための通信のみであり，その性能はキャッシュに大きく依存しないため，妥当な結果だと考える．

hard affinity を使用した評価では，1 個の物理 CPU を Dom0 に占有させるように設定する．また，VM 群は残り 7 個の物理 CPU を共有するよう設定する．CPU スケジューラは credit2 スケジューラとし，各 VM の計測頻度は 200rps とする．

図 2.17 に結果を示す．hard affinity を使用したほうが計測時刻誤差が小さいことがわかる．Dom0 に専用の物理 CPU を割り当てることで，VM と Dom0 との間で CPU 競合が発生しなくなり，その結果，VM の CPU 競合時間が減少した．逆に言えば，Dom0 に 8 個の仮想 CPU を割り当てた場合は，VM と Dom0 が CPU を奪い合う状況が多く発生していたと言える．そのため，VM 数が 8 個の時点から，hard affinity 適用の効果により，計測時刻誤差が小さくなっている．ただし hard affinity

を適用した場合でも，VM 数が 19 個の時点で計測時刻誤差が増大した．xenalyze で分析したところ，VM が 19 個の時点で，VM の CPU 競合の割合が増大していた．VM 群の処理の総量が増加し，VM 間の CPU 競合が頻繁に生じるようになったため，計測時刻誤差が増大したと考える．

この評価結果から，クローラにとっては soft affinity の効果がないこと，hard affinity を適切に設定することで誤差を削減できることを確認できた．また，VM 群による処理の総量が一定値を超えると，hard affinity を使用しても誤差の増大は避けられないことを確認できた．hard affinity により Dom0 に占有させるべき物理 CPU の数は，各 VM の処理や，各 VM の処理により生じる Dom0 の処理を考慮して決定すべきであるが，その組み合わせごとに性能評価を行うことは手間である．監視制御機能の性能が最大となるように，自動的に hard affinity を設定する仕組みがあるとよい．

2.6 VM 管理手法の実現に向けた課題

評価実験で得られた知見に基づき，クローラを実行する VM の管理手法の実現に向けた課題を述べる．VM 管理手法は，1) VM に割り当てるリソース量の決定と 2) VM を実行する物理マシンの決定，を実施する必要がある．

クローラの CPU 使用率は監視点データの計測頻度，すなわち rps に比例する (図 2.8)．計測頻度は，クローラが監視する監視点の数と，各監視点の計測周期から求められる．今回の評価で使用した物理マシンとは異なる物理マシンを使用したとしても，計測頻度と CPU 使用率の関係を導くことは容易であろう．ただし，CPU 使用率を計測する場合，その間隔を適切に設定する必要がある．1 秒間の CPU 使用率の平均値に基づいて仮想 CPU を割り当てると，計測時刻誤差が増加する場合があるためである．一方で，細かい間隔 (例えば 1 ミリ秒) で CPU 使用率を計測することは，オーバーヘッドを増加させるし，CPU 計測の精度を低下させる．許容される計測時刻誤差と同等の間隔で CPU 使用率を計測すべきである．

VM に割り当てるリソース量を決定したら，VM を実行する物理マシンを決定する．各 VM が必要とする CPU リソースの合計が，物理マシンの CPU リソースを

超えると，計測時刻誤差が急増する．そのような状況を避けるためには，複数の物理マシンの CPU の利用状況を把握し，その情報に基づいて VM の配置先を選ぶ必要がある．また，単一物理マシン上に 2 つ以上の VM を配置する場合は，CPU 競合に注意すべきである．VM 群が使用する仮想 CPU の数が，物理 CPU の数を超えた時点で，CPU 競合状態が発生する．CPU 競合が発生すると，物理マシンの CPU リソースに余裕が存在したとしても，計測時刻誤差は増加する (図 2.11)．同時に実行する VM 数とその処理負荷から，生じうる CPU 競合時間を予測した上で，VM を実行する物理マシンを決定すべきである．また，CPU スケジューリングの頻度や物理 CPU の占有数は，適切に設定しないと性能が低下する可能性があるため，監視制御機能の性能を最大化できるよう設定すべきである．

2.7　むすび

本章では，監視制御アプリケーションの 1 つであるクローラを仮想化した場合の性能を評価した．物理マシンが備える CPU コア数以上の VM 数を実行した場合や，CPU コアの一部を仮想化ソフトウェアに占有させた場合の性能を評価した．評価の結果，VM に割り当てる CPU リソース量や，VM 数の増加による CPU 競合，CPU スケジューリングの頻度が，クローラの性能に影響を与える要因であることを明らかにした．第 3 章では，本章で得られた知見に基づき，クローラを実行する VM の性能を推定する方法を検討する．

第3章　仮想化された監視制御アプリケーションの性能の推定

3.1　まえがき

第2章では，監視制御アプリケーションの一例としてクローラを取り上げ，仮想化環境におけるクローラの性能を評価した．本章では，第2章の評価により得られた知見に基づき，仮想化環境における監視制御アプリケーションの性能を推定する手法を提案する．本推定手法を用いることで，遠隔ビル管理システムの運用者は，クローラの性能要件を満たせる範囲で，できるだけ多くのクローラ／VMを単一の物理マシン上で実行できる．これは遠隔ビル管理システムに対する設備投資を削減することに寄与する．また，物理マシン数を減らすことは，物理マシンの管理にかかる人的コストの削減にもつながる．

本章で提案する手法は，シミュレーションにより仮想化環境における監視制御アプリケーションの性能を推定する．シミュレーションであるから，すでに稼働を開始しているクローラの性能に影響を与えない．また，第2章の評価で判明した，クローラの計測時刻誤差を増加させる2要因である 1) VM間のCPU競合と，2) Dom0のCPUリソース不足を再現する点に留意している点が特徴である．

以降，3.2節で想定環境について述べ，3.3節で関連研究を説明する．次に，3.4節で提案手法を説明する．3.5節で提案手法による推定の結果を実機評価の結果と比較し，提案手法の推定精度を評価する．最後に3.6節でまとめる。

3.2　想定環境

本節では，本章で想定する遠隔ビル管理システムとクローラについて説明する．

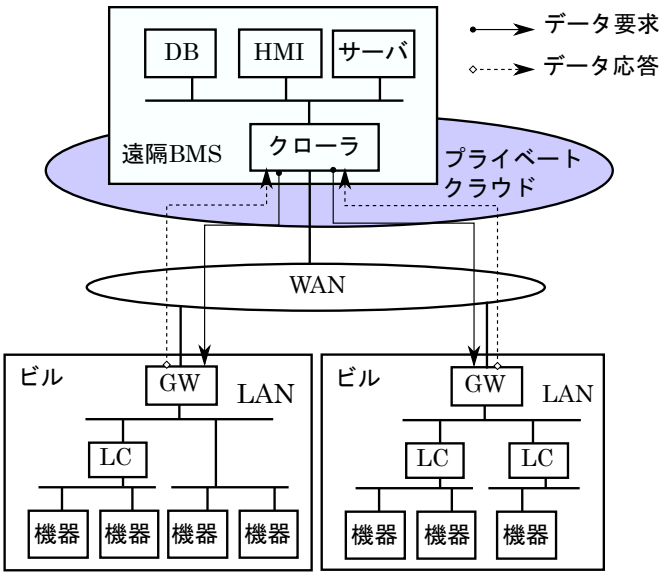


図 3.1: 遠隔ビル管理システムとビル群のシステム構成 (再掲)

3.2.1 遠隔ビル管理システム

本章では，第 2 章と同様に，図 3.1 に示す遠隔ビル管理システムを想定する．クローラの挙動や要件についても，第 2 章と同様の挙動を想定する．

3.2.2 クローラを追加する際の運用

クローラは，ビルオーナーと遠隔ビル管理システム事業者との間でビル管理契約が締結された場合に追加される．クローラを追加する手順を以下に記す．

1. ビルのゲートウェイの IP アドレスや，ゲートウェイが対応している通信プロトコル，各監視点の識別子や計測時刻などをデータベースに登録する．これらの情報は，後にクローラが参照する．

2. クローラを追加する物理マシンを選択する．このときに，遠隔ビル管理システムの運用者は，クローラを追加した後も，すでに稼働しているクローラを含めた全てのクローラの性能要件が満たされることを目指す必要がある．ク

ローラの性能要件を満たせそうな物理マシンが見つからない場合は，新たに物理マシンを購入し，その物理マシンを選択する．

3. 選択した物理マシン上にクローラ／VM を追加する．

4. クローラを実行する．

上記の手順 2 において，物理マシンごとに，クローラを追加した場合の計測時刻誤差を推定できれば，性能要件を満たせる物理マシンを適切に選択できる．

ビル管理契約の締結には人が介在するため，遠隔ビル管理システムの運用者は，クローラ／VM を追加するタイミングを調節できる．仮に同時に複数のビル管理契約が成立したとしても，クローラの追加は逐次的に行える．別の言い方をするなら，クローラを同時に追加することはない．また，一度追加したクローラを，別の物理マシンに移動することは考えない．仮想化ソフトウェアのライブマイグレーション機能を使えば，クローラを停止することなく別の物理マシンへ移動できるが，移動中はクローラの性能が低下するためである．

3.3 関連研究

本節では，仮想化環境におけるアプリケーションの性能を推定する既存の手法について説明する．いずれの手法も，何らかの方法で，アプリケーション／VM を追加した後の性能を推定し，その結果，それでもアプリケーションの性能要件を満たせると判断できた場合に限り，物理マシンに VM を追加する．

3.3.1 リソースに基づく性能推定手法

物理マシンのリソース残量に基づいて，アプリケーションの性能要件を満たせるかを判断する手法がある．このリソースに基づく手法は，追加する VM のリソース消費量の予測値よりも，物理マシンのリソース残量が大きい場合に，VM の追加を許可する [65, 87, 97]．つまり，物理マシンのリソースが余っていれば，アプリケーションの性能は低下しないという想定に基づいている．

これらの手法は，主にウェブアプリケーションを想定しており，物理マシンのリソースの利用率向上を重視している．2章の評価により，物理マシンのリソースが余っていても，VM間のCPU競合により性能が低下する場合があることがわかっている．この性能低下は，処理のタイミングが数ミリ秒から数十秒ずれる程度であるから，ウェブアプリケーションでは無視できるかもしれないが，監視制御アプリケーションでは無視できない場合がある．そのため，リソースに基づく手法を監視制御アプリケーション向けに使うことは難しいと考える．

3.3.2 実際の負荷に基づく性能推定手法

実際に負荷を発生させることで，アプリケーションの性能を推定する手法がある．DeepDive [70] は,「追加する予定のVMによる負荷」と類似する負荷を物理マシンに与えることで，その物理マシン上で稼働中のアプリケーションの性能が低下するかを確認する方法である．稼働中の物理マシンに実際に負荷を与えるため，アプリケーションの性能の変化を正確に推定できる．しかし，稼働中のアプリケーションの性能を低下させるリスクが生じるため，監視制御アプリケーション向けに使うことは難しい．また，追加するVM上のアプリケーションの性能は推定できない．

文献 [96] で提案されている手法は，製品環境で稼働中のVMをコピーすることで製品環境と同じテスト環境を構築する．ウェブアプリケーションを想定しており，製品環境に対して送られてきたHTTPリクエストを，テスト環境に対してコピーすることで，テスト環境でウェブアプリケーションの性能を評価する．この手法は，VMをコピーしている時間は，稼働中のアプリケーションの性能に影響を与える．また，テスト用のサーバやネットワーク機器が必要となるため，製品環境と同等の設備投資がテスト環境に必要となる可能性がある．

文献 [11] で提案されている手法は，事前に実機を用いた性能評価を実施し，その結果に基づいて，アプリケーション追加時に性能が低下するかを判断する．この手法は，稼働中のアプリケーションに影響を与えない．また，製品環境をコピーする必要がないため，文献 [96] で提案されている手法と比べて設備投資を抑えら

れる．しかし，この手法が判断できることは「性能が低下するかどうか」だけであり，どのくらい性能が低下するかは推定できない．

3.3.3 VM 間の干渉を考慮した性能推定手法

文献 [95] で提案されている手法は，VM 間の性能干渉を考慮して，VM の性能変化を定式化している．文献 [95] では，円周率演算の速度を性能として，提案の式の正しさを検証している．しかし，本論文が想定するクローラの性能 (計測時刻の正確さ) の変化は，文献 [95] で定義されている式に当てはまらなかった．クローラの挙動は通信処理を含み，通信処理は VM 群を管理する仮想化ソフトウェアにも負荷を与えるため，処理が VM 内で完結する円周率演算と比べて，性能の定式化が難しいと考える．本章で提案する手法は，クローラや VM の処理をシミュレーションにより模擬することで，クローラの計測時刻誤差を推定する．

3.4 提案手法

提案手法は，クローラや VM，物理マシンのモデルに基づくシミュレーションにより，計測時刻誤差を推定する．

3.4.1 提案手法に基づくクローラ追加時の運用

提案手法を用いた場合の，クローラの追加の手順を説明する．図 3.2 にクローラを追加する運用のフローチャートを示す．

ビル管理契約を締結した後，まず，新しく追加するクローラ (新規クローラ) が監視することになる監視点の数と，監視点ごとの計測頻度を把握する．この情報に基づいて，新規クローラの計測スケジュールを作成する [43]．計測スケジュールが決まれば，計測頻度が決まる．2 章の評価から，クローラの負荷の傾向は CPU バウンドであり，その CPU 使用率は計測頻度に対し線形であることがわかってい

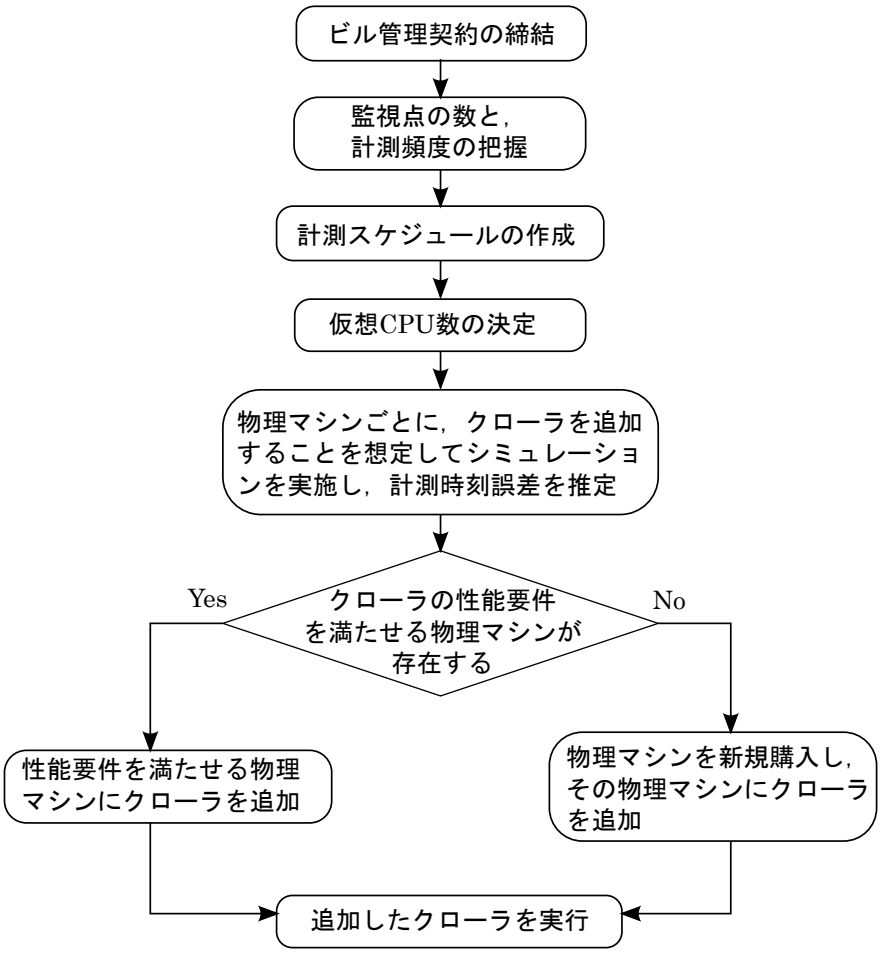


図 3.2: クローラを追加する運用フローチャート

る．すなわち，計測頻度 (r) と CPU 使用率 (U_{cpu}) の関係は以下の一次式で表すことができる．

$$U_{cpu} = ar + b \tag{3.1}$$

a と b は係数であり，実機を用いた評価により得られる計測頻度と CPU 使用率の組を用い，残差のユークリッドノルムに関して最小二乗法を用いることで求められる．この式に基づき，計測頻度から新規クローラに割り当てるべき仮想 CPU の数を求める．

新規クローラの計測スケジュールと仮想 CPU の数が決まったら，物理マシンごとに，新規クローラを追加した状態を想定してシミュレーションを実施する．こ

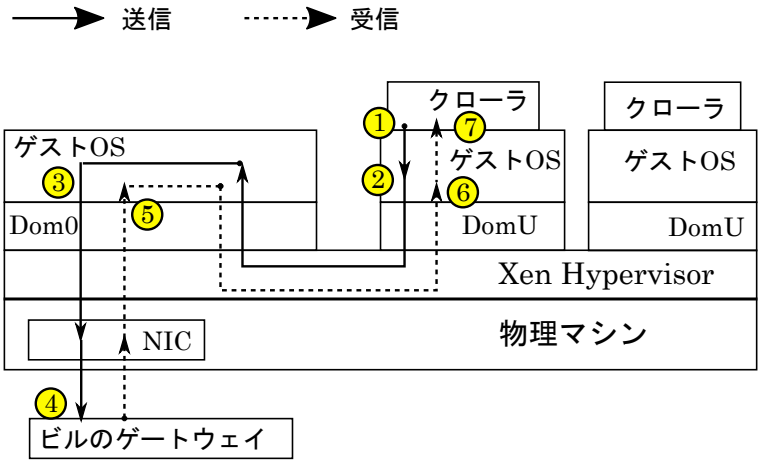


図 3.3: クローラの監視点データの計測処理の流れ

のシミュレーションにより，物理マシン上で動作する全クローラの計測時刻誤差を推定する．そして，推定した計測時刻誤差に基づき，クローラの性能要件を満たせるかを判定する．

新規クローラを追加したとしても，実行中のクローラ群の性能要件を満たせる物理マシンが見つかった場合は，その物理マシンに新規クローラを追加する．クローラ群の性能要件を満たせる物理マシンが複数見つかる場合も考えられる．この場合は，何らかの規則に基づき，新規クローラを追加する物理マシンを一つに絞る必要がある．例えば，ランダムで選択してもよいし，物理マシンの CPU やメモリなどのリソース残量が最も大きい物理マシンを選択してもよい．

クローラ群の性能要件を満たせる物理マシンが見つからなかった場合は，新規に物理マシンを購入する．そして，その物理マシンに新規クローラを追加する．最後に，追加した新規クローラを実行する．提案手法を用いることで，このような手順に基づいて新規クローラを追加することで，クローラ群の性能要件を満たすことを目指す．

3.4.2 シミュレーションのモデルの概要

提案するシミュレータは，クローラの計測処理をシミュレートする．図 3.3 に，クローラの実際の計測処理の流れを示す．

1. クローラはデータ要求を送信する．データ要求はBACnet/WSのgetValuesメッセージである．BACnet/WSはHTTPベースの通信プロトコルであるから，クローラによる送信は，ゲストOSが提供するTCPのAPIを使用することで行われる．

2. ゲストOSは，クローラから受けとったデータ要求を，Dom0のゲストOSに送信する．

3. Dom0は，物理マシンのネットワークインターフェースカードを介して，ビルのゲートウェイに対してデータ要求を送信する．

4. ビルのゲートウェイは，データ要求に対してデータ応答を返す．

5. Dom0のゲストOSはデータ応答を受信する．

6. Dom0のゲストOSは，受信したデータ応答をDomUのゲストOSに渡す．

7. クローラは，ゲストOSからデータ応答を受け取る．

上記のクローラの計測処理は，クローラのデータ要求の生成処理，DomU上のゲストOSのプロセススケジューリングやTCP通信の輻輳制御，同じくDom0上のゲストOSのプロセススケジューリングやTCP通信の輻輳制御，ハイパーバイザによるCPUスケジューリングなど，様々な処理を内包する．これらのそれぞれを詳細にモデル化することで，シミュレーションの精度は向上すると考えられる．しかし，シミュレーションのモデルが，OSやハイパーバイザの実装へ強く依存度するようになるため，実装ごとにモデルを作成する必要が生じる．また，シミュレーションにかかる時間が長くなる．

そこで，提案するシミュレータでは，通信処理をOSやハイパーバイザに依存しないように抽象化する．これにより，OSやハイパーバイザの実装への依存度を低くし，かつ，シミュレーションにかかる時間を短くする．ただし，監視制御アプリケーションは，監視や制御の信号を他の機器とやりとりするタイミングが重要である．クローラであれば，計測時刻誤差が小さいことが重要である．そこで，2

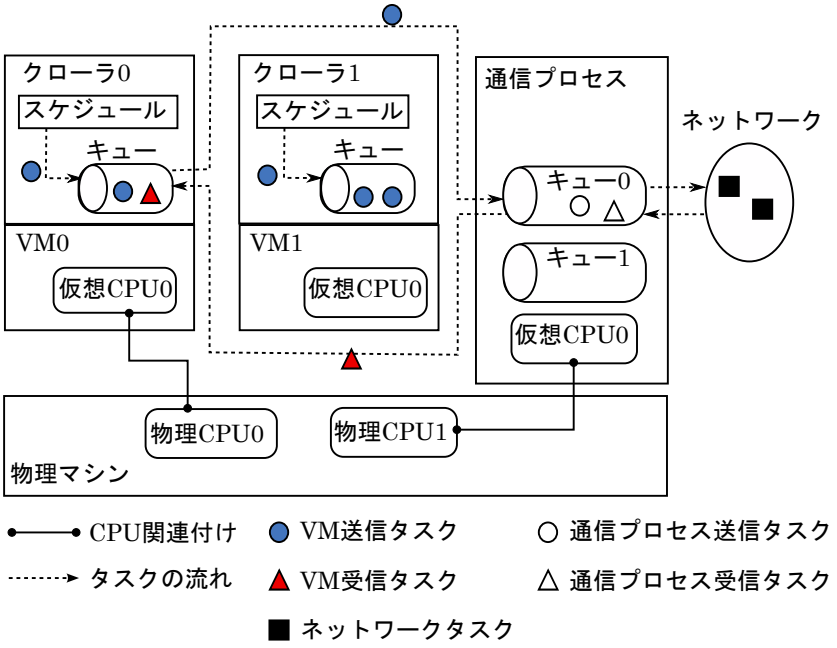


図 3.4: シミュレーションのモデルの全体像

章の評価で判明したクローラの計測時刻誤差を増加させる 2 要因である 1) VM 間の CPU 競合と，2) Dom0 の CPU リソース不足の再現性には留意する．

図 3.4 に，シミュレーションのモデルの全体像を示す．以下，図 3.4 の各要素を説明する．

物理マシン

物理マシンは，1 個以上の物理 CPU を持つ．物理 CPU は，VM の仮想 CPU と関連付けられる場合がある．この関連付けは 1 対 1 である．詳細は 3.4.4 項で説明する．

クローラの処理負荷は CPU バウンドであるため，メモリやストレージなどの他の要素は考慮しない．

VM

VMは，1個以上の仮想CPUと1個のクローラを持つ．仮想CPUは，物理CPUと1対1で関連付けられる場合がある．また，その関連付けは，時間の経過により変化する．物理CPUと関連付けられた場合，仮想CPUは，対応するクローラを実行できる．VMの数が増えると，仮想CPUの数も増えるため，物理CPUと関連付けられていない仮想CPUが発生する．物理CPUと関連付けられていない仮想CPUは，クローラを実行できない．このように物理CPUと仮想CPUの関連付けをモデル化することで，クローラを実行できない時間帯，すなわちCPU競合状態を再現する．詳細は3.4.4項で説明する．

クローラ

クローラは，計測スケジュールとタスクキューを持つ．計測スケジュールは，いつ，どの監視点データを計測するかを定義している情報である．クローラの計測処理は「タスク」としてモデル化する．タスクはタスクキューに格納され，仮想CPUにより処理される．タスクの詳細は3.4.3項で説明する．

通信プロセス

通信プロセスは，物理マシンのネットワークインターフェースカードを使用し，別の物理マシンとの通信を行うプロセスである．実際には仮想化ソフトウェアの一部として実装される場合もあり，例えばXenのDom0や，KVMのホストOSに相当する．通信プロセスは1個以上の仮想CPUを持つ．また，VMごとに有限サイズのタスクキューを持つ．この有限サイズのタスクキューが，未処理のタスクであふれた場合が，Dom0のCPUリソース不足の状態に相当する．詳細は3.4.3項で説明する．

ネットワーク

ネットワークは，データ要求をビルに送信してから，データ応答が返ってくるまでの応答時間を模擬するためのタスクプールである．タスクプールのサイズに上限はない．

シミュレーション時間

提案のシミュレータにおいて，時間は離散的に進む．離散的に進む時間の最小単位を,「シミュレーション単位時間」と呼ぶ．

3.4.3 監視点データの計測処理のモデルの詳細

3.4.2 項で述べたように，クローラの計測処理は，様々な複雑な処理を内包する．提案手法では，これを以下の 5 種類のタスクとしてモデル化する．

- **VM 送信タスク**は，クローラがデータ要求を作成する処理や，ゲスト OS がデータ要求を通信プロセスに渡す処理に相当する．VM 送信タスクを処理するために必要な時間を，T_{vm} と記述する．
- **VM 受信タスク**は，ゲスト OS がデータ応答を通信プロセスから受け取る処理や，クローラがデータ応答を解析する処理に相当する．VM 受信タスクを処理するために必要な時間を，T_{vm} と記述する．
- **通信プロセス送信タスク**は，通信プロセスがデータ要求をゲートウェイに送信する処理に相当する．通信プロセス送信タスクを処理するために必要な時間を，T_{cp} と記述する．
- **通信プロセス受信タスク**は，通信プロセスがデータ応答をゲートウェイから受信する処理に相当する．通信プロセス受信タスクを処理するために必要な時間を，T_{cp} と記述する．

- **ネットワークタスク**は，データ要求やデータ応答がネットワークを流れている時間や，ビルのゲートウェイがデータ応答を作成する処理に相当する．ネットワークタスクを処理するために必要な時間を，T_{net} と記述する．

このように，監視点データの計測処理は，VM 送信タスク，通信プロセス送信タスク，ネットワークタスク，通信プロセス受信タスク，VM 受信タスクの 5 つのタスクで表わされる．これらのタスクは，以下のように処理される．

1. クローラの計測スケジュールに従い，VM 送信タスクが作成され，タスクキューに格納される．

2. タスクキューに格納された VM 送信タスクは，VM の仮想 CPU により処理される．処理された時間が T_{vm} に達すると，VM 送信タスクの処理は完了する．すなわち，クローラが作成したデータ要求が通信プロセスに渡されたことを意味する．

3. VM 送信タスクの処理が完了したら，通信プロセスのタスクキューに通信プロセス送信タスクを格納する．前述のとおり，通信プロセスは VM ごとにタスクキューを持つため，対応するタスクキューに格納する．ただし，通信プロセスのタスクキューがタスクで一杯の場合は，タスクが処理されて，タスクキューに空きが生じるまで待つ必要がある．通信プロセス送信タスクを格納できたら，VM 送信タスクを削除する．

   通信プロセスがタスクを処理する速度よりも速く，クローラ群がタスクを処理すると，通信プロセスのタスクキューは一杯になる．これは，通信プロセス (Dom0) の CPU リソースが不足している状況である．このようなモデル化により，Dom0 の CPU リソース不足を模擬する．

4. 通信プロセス送信タスクは，通信プロセスの仮想 CPU により処理される．処理された時間が T_{cp} に達すると，通信プロセス送信タスクの処理は完了する．すなわち，データ要求がビルのゲートウェイに対して送信されたことを意味する．

5. 処理が完了した通信プロセス送信タスクはタスクキューから削除される．そして，ネットワークタスクが，ネットワークのタスクプールに格納される．

6. ネットワークタスクは，シミュレーション単位時間ごとに処理される．処理された時間がT_{net}に達すると，ネットワークタスクの処理は完了する．すなわち，データ応答がゲートウェイから返ってきたことを意味する．

7. ネットワークタスクは削除される．そして，通信プロセス受信タスクが，通信プロセスのタスクキューに格納される．

8. 通信プロセス受信タスクは，通信プロセスの仮想 CPU により処理される．処理された時間がT_{cp}に達すると，通信プロセス受信タスクの処理は完了する．すなわち，データ応答を VM 上のゲスト OS に対して送信したことを意味する．

9. 処理が完了した通信プロセス受信タスクを削除する．そして，VM 受信タスクを，対応する VM のタスクキューに格納する．

10. VM 受信タスクは，VM の仮想 CPU により処理される．処理された時間がT_{vm}に達すると，VM 受信タスクの処理は完了する．すなわち，データ応答の受信および解析が完了したことを意味する．

11. 処理が完了した VM 受信タスクを削除する．

以上が，シミュレーション上の監視点データの計測処理である．

3.4.4 物理 CPU と仮想 CPU のモデルの詳細

物理マシンのリソースが余っていても，VM 間の CPU 競合が生じれば，アプリケーションの性能は低下する．CPU 競合とは，物理 CPU を使用したい VM の仮想 CPU が，物理 CPU を使用できない状態のことである．ある時点において,「物理 CPU を使用したい仮想 CPU の数」が「物理マシンが備える物理 CPU の数」を

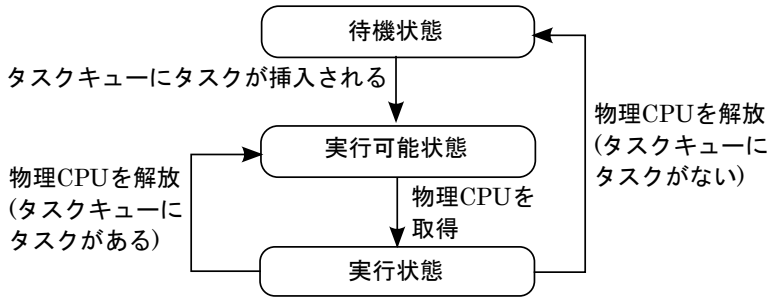


図 3.5: シミュレーションにおける仮想 CPU の状態遷移

超えると，CPU 競合が生じる．CPU 競合はクローラの計測時刻誤差の主要因であるため，これを模擬できるように物理 CPU と仮想 CPU をモデル化する．

図 3.5 はシミュレーションにおける仮想 CPU の状態遷移図である．仮想 CPU の初期状態は「待機状態」である．1 つ以上の VM 送信タスクまたは VM 受信タスクがクローラのタスクキューにあるとき，そのクローラを実行する VM の仮想 CPU は，待機状態から「実行可能状態」に遷移する．同様に，1 つ以上の通信プロセス送信タスクまたは通信プロセス受信タスクがタスクキューにあるとき，通信プロセスの仮想 CPU は，待機状態から「実行可能状態」に遷移する．

仮想 CPU と関連付けられていない物理 CPU は，実行可能状態である仮想 CPU をランダムで 1 つ選ぶ．物理 CPU に選ばれた仮想 CPU は,「実行状態」へと遷移し,「物理 CPU 使用可能時間」が割り当てられる．物理 CPU 使用可能時間とは，物理 CPU を解放することなく使用し続けられる時間である．仮想 CPU に割当てる物理 CPU 使用可能時間の量は，稼働中の物理マシンをモニタリングした情報に基づいて決定する．詳細は 3.4.5 項で述べる．

実行状態の VM の仮想 CPU は，シミュレーション単位時間ごとに，クローラのタスクキューから First-In First-Out (FIFO) でタスク (VM 送信タスクまたは VM 受信タスク) を選び，処理する．これにより，タスクの「処理された時間」が，シミュレーション単位時間分だけ増加する．また，仮想 CPU の物理 CPU 使用可能時間は，シミュレーション単位時間分だけ減少する．前項で述べたように，処理された時間が T_{vm} に達すると，タスクは処理完了となる．

同様に，実行状態の通信プロセスの仮想 CPU は，いずれかのタスクキューから，

FIFO でタスク (通信プロセス送信タスクまたは通信プロセス受信タスク) を取得し，処理する．タスクキューはランダムで選択する．

実行状態の仮想 CPU は，以下のいずれかの条件を満たした場合に，使用中の物理 CPU を解放する．

- タスクキューにタスクが存在しない場合
- 自身に割当てられた物理 CPU 使用可能時間が 0 以下になった場合

物理 CPU を解放した仮想 CPU は，タスクキューにタスクがあるなら実行可能状態に，タスクがないなら待機状態に遷移する．解放された物理 CPU は，再び，実行可能状態の仮想 CPU を選ぶ．

物理 CPU に選ばれなかった仮想 CPU は，タスクキューにあるタスクを処理できない．いずれかの物理 CPU が解放され，次に自身が選ばれることを待つ必要がある．この待ち時間が，CPU 競合時間となる．

3.4.5 モニタリング情報に基づくパラメータの決定

3.4.3 項で述べたタスクの処理にかかる時間や，3.4.4 項で述べた物理 CPU 使用可能時間は，物理マシンの性能や CPU スケジューリングのアルゴリズムに依存する．物理マシンのスペックや CPU スケジューリングの詳細をモデル化することは，提案手法の汎用性を低下させることになる．一方，これらのパラメータは，クローラの性能に影響する CPU 競合状態を再現するために重要である．そこで提案手法では，実際にビル群に対して稼働している物理マシン (製品環境の物理マシン) をモニタリングして得られる情報を用いて，これらのパラメータを決定する方針とする．これにより，物理 CPU や OS の実装に依存することなく，CPU 競合の再現性を高めることを目指す．

タスクの処理時間の決定

T_{vm} と T_{cp} を決定するために，クローラを稼働している物理マシンにて，ある期間 D において，以下のデータを計測する．

- U_{vm}: 物理マシン上の VM 群の CPU 使用時間の合計 (秒)
- U_{cp}: 通信プロセスの CPU 使用時間 (秒)
- N: 物理マシン上のクローラ群が実施した計測処理の回数

VM 群の CPU 使用時間は，仮想化ソフトウェアが提供するコマンドにより計測できる．例えば Xen であれば `xentop` コマンドや `xl` コマンドで計測できる．KVM であれば `ps` コマンドや `top` コマンドで計測できる．

VM 送信タスクと VM 受信タスクを処理するために必要な時間 T_{vm} は，U_{vm} と N から計算する．

$$T_{vm} = \frac{U_{vm}}{2N} \tag{3.2}$$

$\frac{U_{vm}}{N}$ は，一回の計測処理にかかる時間を表している．提案手法は，VM 送信タスクと VM 受信タスクの処理にかかる時間は等しいという前提を置く．そのため，$\frac{U_{vm}}{N}$ を 2 で割った値を T_{vm} とする．

実際には，クローラの送信処理にかかる時間と受信処理にかかる時間は異なると考えられる．両方の処理にかかる時間の比率を正確に分析するためには，アプリケーション (クローラ) をより詳細にプロファイリングする必要がある．アプリケーションごとに詳細なプロファイリングを行うことは手間である．そこで本章では，送信と受信の処理時間が等しいという単純な前提のもとで，どれだけの精度で性能を推定できるかを検証する．このモデルであれば，異なるアプリケーションでも，そのまま適用できる．

同様に，通信プロセス送信タスクと通信プロセス受信タスクを処理するために必要な時間 T_{cp} は，U_{cp} と N から計算する．

$$T_{cp} = \frac{U_{cp}}{2N} \tag{3.3}$$

ここでも，通信プロセス送信タスクと通信プロセス受信タスクの処理にかかる時間は等しいという前提を置く．

T_{net} は，監視点データを計測するごとに，ビルのゲートウェイの応答時間を計測し，その平均値を計算することで求める．

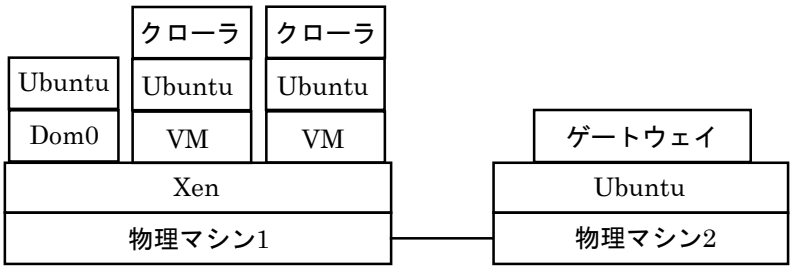


図 3.6: 実機評価の環境

物理 CPU 使用可能時間の決定

クローラを稼働している物理マシンにて，仮想 CPU が物理 CPU を解放するまでの時間をモニタリングする．Xen であれば `xentrace` コマンドを使用することで，この時間をモニタリングできる．仮想 CPU が物理 CPU を解放するまでの時間は，一定ではない．例えば，Xen の credit スケジューラのデフォルトの設定を用いた場合は，数マイクロ秒から 30 ミリ秒の間で変化する．そこで，モニタリング結果から，仮想 CPU が物理 CPU を解放するまでの時間と，その発生頻度の組み合わせを取得し，それを線形補間したものを，物理 CPU 使用可能時間の割り当て量を決定するための確率密度関数とする．

3.5 評価

提案手法の有効性を検証するために，実機の計測時刻誤差と，提案手法により推定した計測時刻誤差を比較する．

3.5.1 実機評価の環境

実機評価の環境を図 3.6 に示す．評価には 2 つの物理マシンを用いる．1 つはクローラ群を実行するために，もう 1 つはビルのゲートウェイ群を模擬するために用いる．2 つの物理マシンは 1000BASE-T で接続する．クローラ用の物理マシンは Intel(R) Xeon(R) 1.80GHz CPU (8 コア)，32GB メモリを備える．仮想メモリアドレスと物理メモリアドレスの変換処理を効率化する Extended Page Tables 機

表 3.1: 実機評価のパラメータ

パラメータ	値
クローラ/VM の数	1 から 20
クローラ/VM の追加間隔 (分)	10
各クローラの計測頻度 (rps)	100,200,300
各 VM の仮想 CPU 数	1
クローラ/VM 群が使用できる物理 CPU の数	7
試行回数	10

能は有効化する．ビル GW 用の物理マシンは Intel(R) Xeon(R) 2.60GHz CPU (32 コア)，80GB メモリを備える．物理マシンと VM の OS として Ubuntu 16.04 LTS (Linux 4.4.0) を使用する．

仮想化ソフトウェアとしてバージョン 4.6.0 の Xen を使用し，デフォルトのスケジューラである credit スケジューラを使用する．credit スケジューラの timeslice パラメータは，デフォルト値である 30 ミリ秒を使用する．cap パラメータによる CPU 使用量の制限は行わない．また，weight パラメータによるクレジット割り振りの重みづけは行わない．2 章の評価から，Dom0 (通信プロセス) に物理 CPU を占有させることで，性能が向上することがわかっている．そこで，CPU affinity (Hard affinity) を使い，Dom0 に物理 CPU を 1 つ占有させる．つまり，VM 群は 7 個の物理 CPU を共有する．また，完全仮想化よりも準仮想化のほうが性能が高いこともわかっているため，準仮想化を使用する．

本評価で用いるクローラは，2 章で使用したクローラと同一である．

3.5.2 実機を用いた評価

実機を用いた評価を実施した．表 3.1 に実機評価のパラメータを示す．実機評価では，クローラ／VM を，10 分ごとに 1 個ずつ追加し，20 個まで追加した．各クローラが管理する監視点数は 100, 200, 300 のいずれかとし，各監視点の計測周期は全て 1 秒とした．つまり各クローラの計測頻度 (rps; Request per Second) は 100,

200, 300 のいずれかとした．100rps は，ビル数棟分の計測に相当する頻度である．また，各クローラの計測頻度が同じ場合と，異なる場合を評価した．計測頻度が異なる場合として，以下を評価した．

- 100rps のクローラと 300rps のクローラを交互に追加する場合
- 200rps のクローラと 300rps のクローラを交互に追加する場合
- 100rps のクローラ, 200rps のクローラ, 300rps のクローラを順番に追加する場合

各 VM の仮想 CPU は 1 個とした．1 個の仮想 CPU は，VM が 1 個であれば，300rps の計測処理に対して十分な計算リソースである．

評価では計測時刻誤差の 99.999 パーセンタイル値を計測した．ただし，クローラ／VM を追加している時間帯は除いて，計測時刻誤差の 99.999 パーセンタイル値を計測した．クローラ／VM を追加している時間帯は，通常とは異なる負荷が物理マシンに加わるため，すでに稼働中のクローラの計測時刻誤差が乱れる可能性があるが，一時的な乱れだと考えられるためである．以降，単に計測時刻誤差と記述する場合は，計測時刻誤差の 99.999 パーセンタイル値を意味する．実機評価で得られた計測時刻誤差は，3.5.5 項でシミュレーションによる推定値と比較する．

実機評価の際，3.4.5 項で述べた方法に基づき，タスクの処理に必要な時間を求めた．T_{vm} と T_{cp} は VM 数や計測頻度に応じて変化した．その様子を図 3.7 に示す．T_{vm} と T_{cp} は，計測頻度が大きくなると小さくなった．Dom0 と各 VM は，専用のメモリ領域を使用してデータを交換する．計測頻度が大きくなると，一度のメモリコピーで複数のデータ要求／データ応答のデータが交換されるようになるため，タスクあたりの処理時間は減少する．T_{cp} が VM 数に対して反比例する理由も同様であり，ネットワークに対して送受信するデータが増加するに従い，Dom0 とネットワークインターフェースカードの間のデータ交換の効率が上がるためである．

また，3.4.5 項で述べた方法に基づき，ネットワークタスクの処理にかかる時間 T_{net} の値を求めた．T_{net} の値は表 3.2 に記す．

また，3.4.5 項で述べた方法に基づき，物理 CPU 使用可能時間の確率密度関数を求めた．仮想 CPU が物理 CPU を解放するまでの時間は，`xentrace` コマンドを用

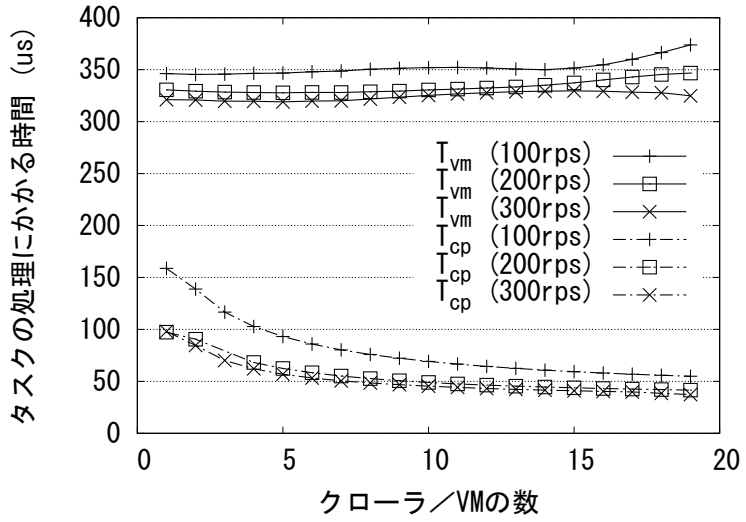


図 3.7: VM 数と計測頻度とタスクの処理時間の関係

いて計測した．計測結果の累積分布関数 (CDF) を図 3.8 に示す．Xen の credit スケジューラの timeslice パラメータを 30 ミリ秒としたため，物理 CPU 解放までの時間の最大値はほぼ 30 ミリ秒となった．この「物理 CPU を開放するまでの時間」を，物理 CPU 使用可能時間とする．

3.5.3 シミュレーションによる評価

次に，シミュレーションによる評価を実施した．シミュレーション評価のパラメータを表 3.2 に示す．VM の数は 2 個から 20 個までとした．1 個目の VM を追加する時点では，稼働中の物理マシンのモニタリング情報が存在しない．したがって，タスクの処理にかかる時間 (T_{vm} と T_{cp}) やネットワークタスクの処理にかかる時間 T_{net} を計算できず，よって計測時刻誤差を推定できないためである．

T_{vm} と T_{cp} は計測頻度と VM 数に応じて変化したため (図 3.7)，最新の値を用いてシミュレーションを実施した．例えば，X 個目のクローラ／VM を追加した場合の計測時刻誤差を推定する場合は，すでに $X-1$ 個のクローラが稼働中であるから，クローラ／VM が $X-1$ 個のときの T_{vm} と T_{cp} を使用してシミュレーションを実施した．通信プロセスが VM ごとに備えるタスクキューのサイズは，Xen

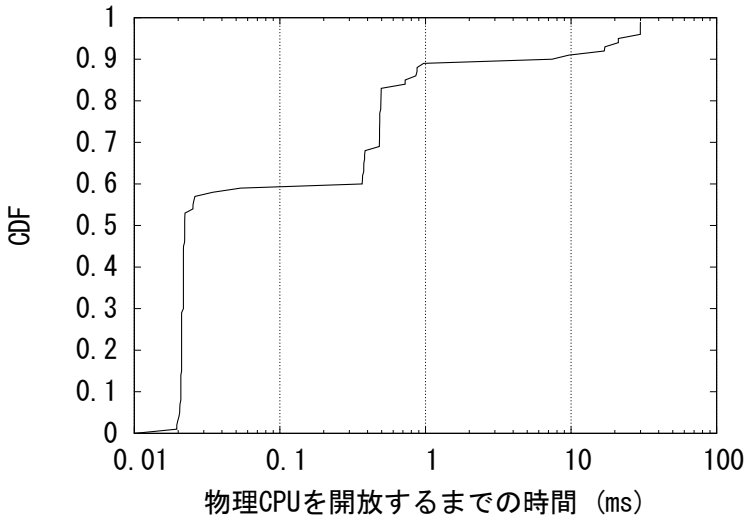


図 3.8: 仮想 CPU が物理 CPU を解放するまでの時間の CDF

の Dom0 と VM がデータ交換するためのリングバッファのサイズと同じ 256 とした．シミュレーションによる推定結果は，3.5.5 項で述べる．

3.5.4 指数関数の近似式による評価

提案手法に対する比較手法として，近似式による推定を実施した．実機評価の結果，計測時刻誤差の実測値は指数的に増加することがわかった．そこで，実測値を最小二乗法を用いて以下の指数関数にフィッティングすることで近似式を求め，その近似式を用いて計測時刻誤差を推定した．

$$y = a\mathrm{e}^{bx} \tag{3.4}$$

x がクローラ／ VM の数，y が計測時刻誤差である．a と b は係数である．

VM を追加するごとにフィッティングをやりなおし，推定した．すなわち，実行中の VM 数が X のとき，VM 数が 1 から X までの計測時刻誤差の実測値から近似式を求め，VM 数が $(X+1)$ のときの計測時刻誤差を推測した．この指数関数近似による推定結果は，3.5.5 項で述べる．

表 3.2: シミュレーションのパラメータ

パラメータ	値
クローラ／VM の数	2 から 20
クローラ／VM が使用できる物理 CPU の数	7
通信プロセスが占有する物理 CPU の数	1
各 VM の仮想 CPU 数	1
計測スケジュール	実機評価と同じスケジュール
T_{vm} (マイクロ秒)	図 3.7 の値
T_{cp} (マイクロ秒)	図 3.7 の値
T_{net} (マイクロ秒)	660.855
通信プロセスのタスクキュー長	256
シミュレーション期間 (秒)	30
シミュレーションの単位時間 (マイクロ秒)	1

3.5.5 実機評価とシミュレーション評価の比較

本節では，実機評価で得られた計測間隔誤差と，シミュレータにより推定した計測時刻誤差と，指数関数近似により推定した計測時刻誤差とを比較した結果について述べる．

各クローラの計測頻度が等しい場合

図 3.9 から図 3.11 は，各クローラの計測頻度が等しい場合の，計測時刻誤差の実測値と推定値の比較結果をまとめる．

計測時刻誤差の実測値は，VM 数が物理 CPU コア数である 8 個に達するまではほぼ横ばいであり，さらに VM が増えると，指数的に増加している．計測頻度が 100rps の場合は，クローラ／VM 数が 20 個の時点でも計測時刻誤差は約 5 ミリ秒だが (図 3.9)，計測頻度が 300rps の場合は，クローラ／VM 数が 18 個の時点で 100 秒を大きく超えた (図 3.11)．そのため，VM が 17 個の時点から，グラフがほぼ真上に伸びている．この計測時刻誤差の増大は，Dom0 がクローラ群から受信

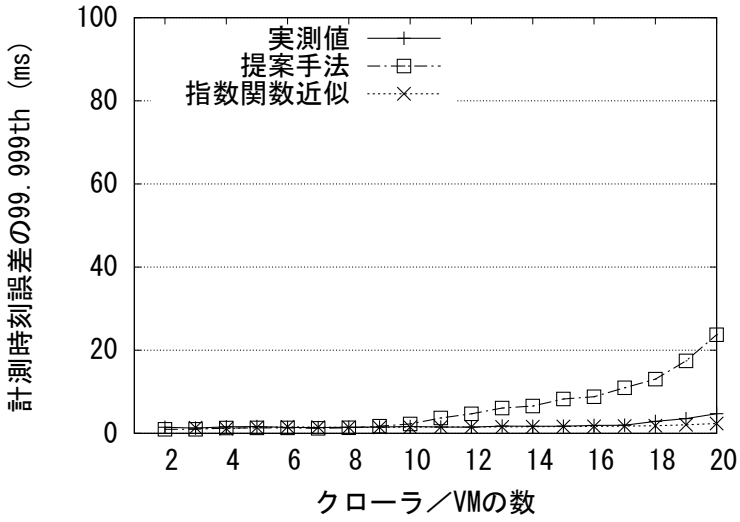


図 3.9: 全クローラの計測頻度が 100rps の場合

したデータ要求と，ビルのゲートウェイから受信したデータ応答を処理しきれなくなったため，生じている．

提案手法による推定値は，実測値と同様に，VM 数に対して指数的に増加しているが，実測値の計測時刻誤差の増加率を正確に再現できていない．計測頻度が 100rps と 200rps の場合は，クローラ／VM が 12 個以上の場合において，実測値よりも大きく推定している．100rps の場合は約 20 ミリ秒，200rps の場合は約 35 ミリ秒，大きく推定している．一方，計測頻度が 300rps の場合は，クローラ／VM が 15 個から 17 個の場合において，実測値よりも小さく推定している．つまり，実測値のほうが推測値よりも，計測頻度の増加に対する計測時刻誤差の増加率が高い．ただし，計測頻度が 300rps の場合において，クローラ／VM 数が 18 個の時点で計測時刻誤差が増大する傾向は再現できた (図 3.11)．提案手法は，通信プロセス (Dom0) に VM ごとの有限長のタスクキューを設けたが，このキューがタスクで一杯になることで，Dom0 のボトルネック状態を再現できたと考える．

指数関数近似式による推定は，実測値よりも低く推定する結果となった．また，計測頻度が 300rps の場合は，実測値との差が大きく，計測時刻誤差が増大する VM 数の推定に失敗している．計測時刻誤差を実測値よりも低く推定する場合，「新たにクローラを追加しても，クローラの性能要件を満たせる」とビル管理システム

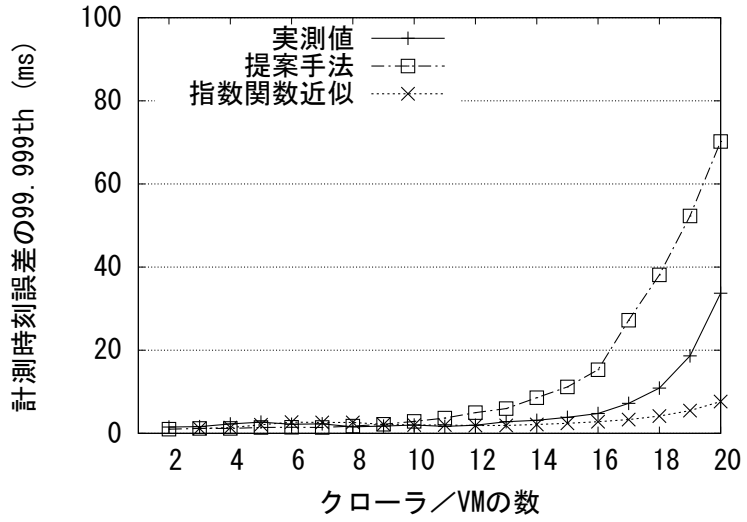


図 3.10: 全クローラの計測頻度が 200rps の場合

の運用者が判断してしまい，本当は計測時刻誤差が大きくなるにも関わらず，クローラを追加してしまう可能性が生じる．また，Dom0 がボトルネックとなり計測時刻誤差が増大する状況は，監視制御機能への影響が大きいことから，推定に失敗することは避けたい．したがって，提案手法は指数関数近似式による推定よりも実運用において役立つと言える．

各クローラの計測頻度が異なる場合

図 3.12 から図 3.14 に，各クローラの計測頻度が異なる場合の，計測時刻誤差の実測値と推定値の比較結果をまとめる．計測頻度が 100rps と 300rps のクローラが混在する場合 (図 3.12) の，各クローラの計測頻度の平均値は 200rps である[1]．Dom0 の処理量は，各クローラの計測頻度が 200rps の場合 (図 3.10) と同等のはずだが，クローラ／VM が 20 個の時点で計測時刻誤差の実測値が増大した．計測頻度が 100rps と 200rps と 300rps のクローラが混在する場合も同様の傾向であり，クローラ／VM が 20 個の時点で，計測時刻誤差の実測値が増大した (図 3.14)．

[1]ただし，クローラ／VM の数が奇数の場合は，各クローラの計測頻度の平均値は 200rps ではない．

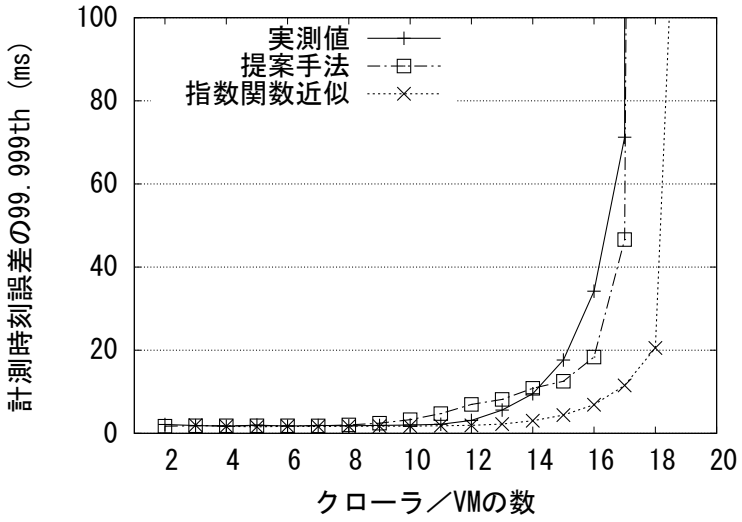


図 3.11: 全クローラの計測頻度が 300rps の場合

今回，Xen の credit スケジューラをデフォルトの状態で使用した．cap パラメータによる CPU 使用量の制限や，weight パラメータによるクレジット割り振りの重みづけは行っていない．つまり，クローラ／ VM の処理負荷の大きさによらず，各 VM に均等に CPU リソースが配分される．そのため，各クローラの計測頻度が 200rps 均一の場合よりも，計測頻度が 100rps のクローラと 300rps とクローラが混在している場合のほうが，300rps のクローラの CPU リソースが不足しやすく，動作が不安定になりやすい．その影響が，クローラ／ VM が 20 個の時点で発生した結果，計測時刻誤差の実測値が増大したと考える．

一方で提案手法のシミュレータは，Xen の credit スケジューラの CPU リソースの配分アルゴリズムを模擬していない．VM の負荷に応じて CPU リソースが配分されるアルゴリズムとなっているため，CPU リソース不足によりクローラの動作が遅れるという状況が，実機よりも起こりづらい．そのため，図 3.14 において，提案手法はクローラ／ VM が 20 個の時点でも計測時刻誤差が増大していない．また，図 3.12 においても，クローラ／ VM が 20 個の時点で，実測値よりも小さく推定している．ただし、クローラ群の計測頻度が同じ場合 (図 3.9〜図 3.11) と同様に，指数関数近似式による推定値と比べると，提案手法による推定値の方が計測時刻誤差の増加傾向を再現できている．

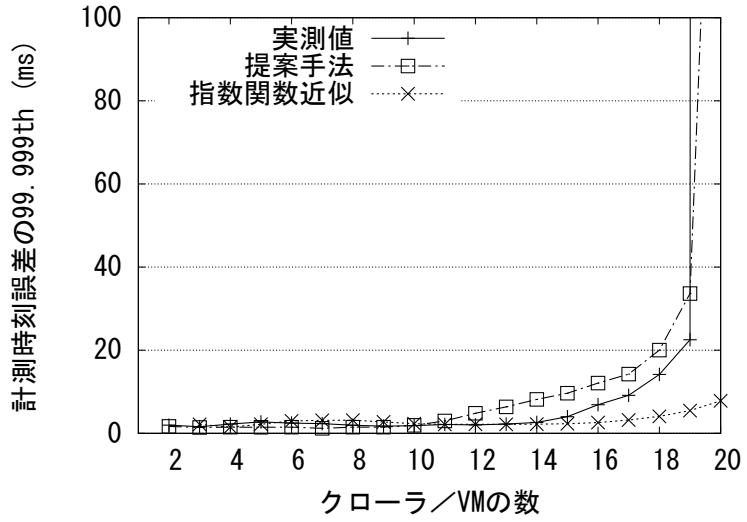


図 3.12: 計測頻度が 100rps と 300rps のクローラが混在する場合

図 3.13 は，計測頻度が 200rps と 300rps のクローラが混在している場合の結果である．この場合，クローラ／ VM 数が 19 個の時点で，Dom0 がボトルネックになり，計測時刻誤差が増大している．このように，Dom0 がボトルネックになる状況が発生している場合は，提案手法は計測時刻誤差が増大する VM 数を推定できている．

指数関数近似式による推定は，実測値よりも低く推定する結果となった．また，Dom0 がボトルネックとなり計測時刻誤差が増大する VM 数の推定に失敗した．各クローラの計測頻度が異なる場合でも，提案手法は指数関数近似式による推定よりも実運用において役立つと言える．

提案手法の効果

2.2.2 項で述べたように，空調や照明の状態を 1 秒間隔で計測する場合であれば，典型的な計測時刻誤差の許容限界は 50 ミリ秒である．この条件に基づいて，提案手法の効果を分析する．

国土交通省の資料に基づき，日本における中小規模ビルの総数を 54 万棟とする[2]．

[2]国土交通省 平成 25 年法人土地・建物基本調査 付表 3-7-2 建物の延べ床面積別建物件数から計算．住宅以外の建物で，延べ床面積が $500m^2$ 以上／ $10000m^2$ 以下の建物を中小規模ビルとした．

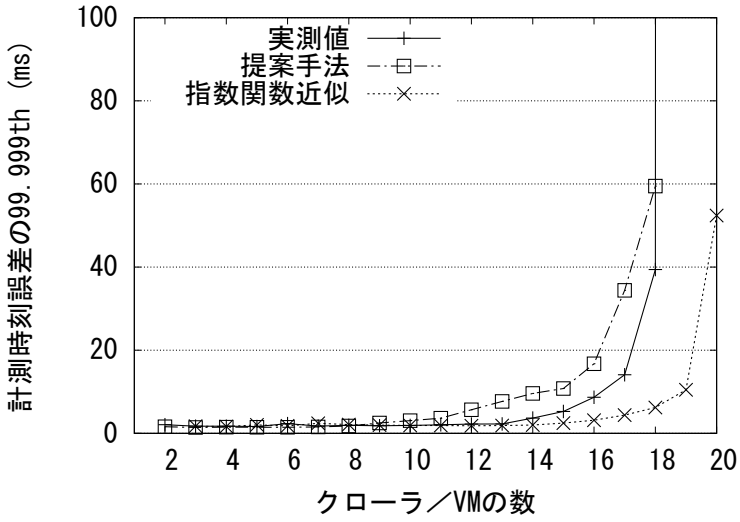


図 3.13: 計測頻度が 200rps と 300rps のクローラが混在する場合

各ビルの監視点数を 100 個とすると，監視点の総数は 540 万点となる．この全てを，1 秒間隔で計測する状況を想定する．各クローラの計測頻度を 200rps とすると，27000 個のクローラが必要である．仮に，本章の評価で用いた物理マシン (8CPU コア) を用いて，単純に 1 コアあたり 1 クローラ／VM とした場合，$\frac{27000}{8} = 3375$ 台の物理マシンが必要となる．一方，図 3.10 に示すように，提案手法を用いることで，クローラが 18 個までであれば，計測時刻誤差は 50 ミリ秒以下であると判定する．つまり提案手法を用いた場合は，$\frac{27000}{18} = 1500$ 個の物理マシンで，国内の中小ビル群を管理できる．1 コアあたり 1 クローラとした場合と比べて，半分以下の物理マシン数で抑えられるため，設備投資を削減できると言える．

評価のまとめ

本評価を通じて，提案手法は，指数関数近似による推定よりも計測時刻誤差の増加の傾向を再現できることがわかった．一方で，推定の精度には改善の余地があることもわかった．特に，クローラごとの計測頻度が異なる場合に，計測時刻誤差が増大する VM 数をより正確に推定できる必要がある．

仮想化ソフトウェアの CPU リソースの配分アルゴリズムをシミュレーションの

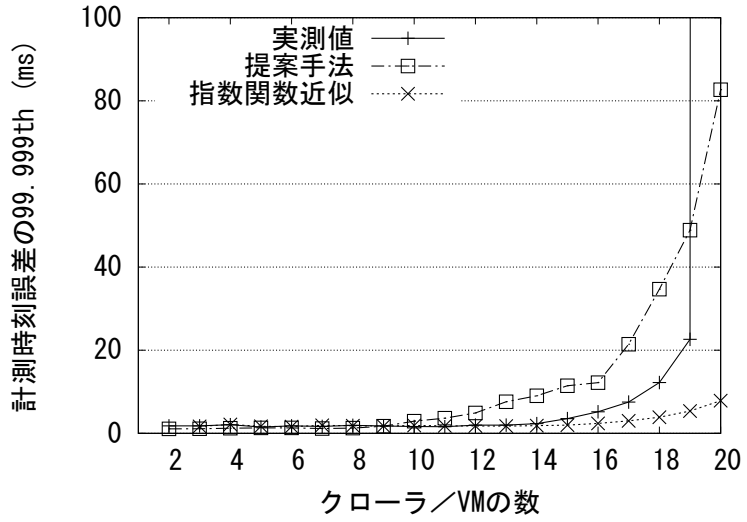


図 3.14: 計測頻度が 100rps と 200rps と 300rps のクローラが混在する場合

モデルに組み込むことで，推定の精度を高められる可能性がある．仮想化ソフトウェアの挙動を正確にモデル化するほど，推定精度は向上すると考えられるが，それは特定の仮想化ソフトウェアに対する提案手法の依存度を高め，汎用性を低下させることを意味する．複数の仮想化ソフトウェアに共通する CPU リソースの配分アルゴリズムを抽象化して，それをシミュレーションのモデルとする必要がある．

3.6 むすび

本章では，監視制御アプリケーションの 1 つであるクローラを想定し，VM 上で動作するクローラの計測時刻の誤差を推定する手法を提案した．実機を用いた評価結果と，提案手法による推定結果と，指数関数の近似式による推定結果を比較した結果，提案手法は指数関数による推定よりは有用であることがわかったが，その推定精度には改善の余地があることもわかった．

今後は，推定精度を改善するために，シミュレーションのモデルを修正し，Xen 以外の仮想化ソフトウェアや，異なる物理マシンを使用した場合の推定精度も評価する．また，図 3.4 に示したモデルの VM と Dom0 のタスクキューを待ち行列と捉え，待ち行列理論に基づく解析的アプローチによる計測時刻誤差の推定手法

も検討する.

第4章　パブリッククラウドにおける監視制御アプリケーションの稼働率の向上

4.1　まえがき

第3章では，仮想化環境における監視制御アプリケーションの性能を保証することを目指し，その性能を推定する手法を提案した．これにより，プライベートクラウドにおいて監視制御アプリケーションを構築し，安定して稼働させることが可能となる．本章では，パブリッククラウドにおいて監視制御アプリケーションを実行することを想定する．パブリッククラウドでは，プライベートクラウドと異なり，ユーザが物理マシンなどのハードウェアを自由に選定できない．すなわち，ユーザがハードウェアの性能や稼働率を調節することが難しい．

一部のパブリッククラウドのプロバイダは，パブリッククラウド上の物理マシンを占有するサービスを提供している．このサービスを利用すれば，性能を調節することはできると考えられる．一方，パブリッククラウドのプロバイダが保証する典型的な稼働率は，99.95%である．これは，IEC 61508 の Safety Integrity Level 1 (SIL1) で定義される監視制御システムの稼働率 99.999%に及ばない値である．監視制御システムをパブリッククラウドで実行するためには，この稼働率のギャップを解消する必要がある．

アプリケーションの稼働率を向上させるための方法は広く研究されている．Web サーバのように，状態を保持しなくても動作を継続できるアプリケーションであれば，冗長化は容易であり，それによって稼働率を向上できる．例えば，Keepalived

という冗長化のためのオープンソースのソフトウェアが存在する[1]．2 章や 3 章で扱ってきたクローラも，Web サーバと同様，冗長化は容易である．一方，状態を保持することが重要なアプリケーションの場合は，冗長化のために，主系 (プライマリ) と従系 (バックアップ) との間で，状態の同期を取る必要がある．監視制御システムにおいては，フィードバック制御を行うコントローラが，この種類のアプリケーションに相当する．

監視制御システムにおいて，フィードバック制御コントローラを対象とした稼働率向上の研究は存在する [57, 81]．しかし，これらの研究は，フィードバック制御コントローラと制御対象の機器が，物理的に近くに存在する環境を想定している．そのため，フィードバック制御コントローラ間で機器の制御の状態を共有することが容易である．パブリッククラウドにてフィードバック制御コントローラを実行する前提で，稼働率を向上する手法を提案している文献もある [36, 94]．これらの手法では，複数のフィードバック制御コントローラを別々のクラウド上で実行し，いずれか一つのフィードバック制御コントローラをプライマリとして選択する．プライマリコントローラの故障の検出は，ビル／工場に設置した専用のゲートウェイを通じて行う．

本章では，パブリッククラウドにおけるフィードバック制御コントローラの稼働率を向上するための手法を提案する．提案手法は，既存手法と同様に，複数のフィードバック制御コントローラを異なるクラウド上で実行する．ただし，異常の検出や，新たなプライマリコントローラの選択に，専用のゲートウェイを利用しない．フィードバック制御コントローラ同士がハートビートを交換することで，互いの異常を検出し，新たなプライマリコントローラを選択する．

以降，4.2 節で想定環境について説明する．次に 4.3 節にて関連研究をまとめる．4.4 節にてパブリッククラウドの性能の安定性について調査した結果を述べる．4.5 節にて提案手法を説明し，4.6 節にて評価結果を述べる．4.7 節でパブリッククラウドを利用した場合の設備投資の削減効果について考察し，最後に，4.8 節で本章のまとめを述べる．

[1] http://www.keepalived.org

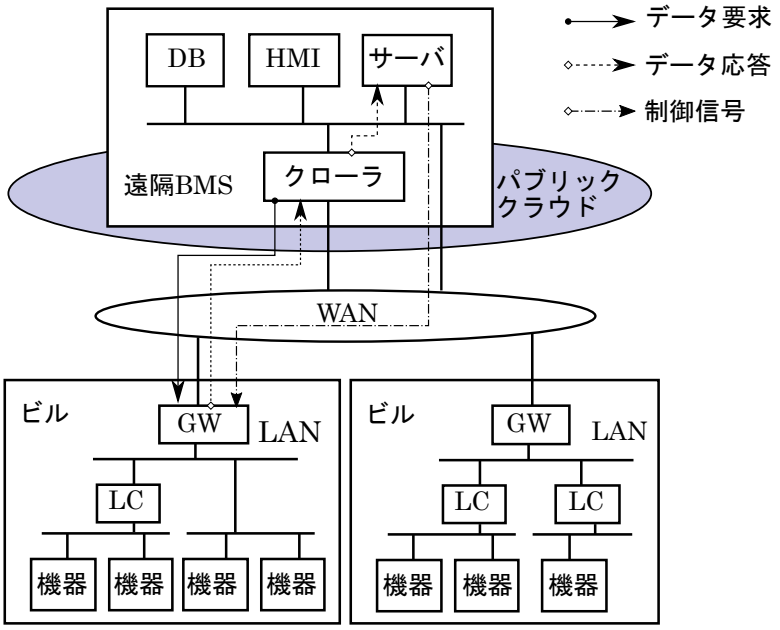


図 4.1: 本章で想定する環境

4.2 想定環境

本章では，第 2 章や第 3 章と同様に，遠隔ビル管理システムを想定する．ただし，本章で注目する監視制御アプリケーションは，クローラではなく，フィードバック制御コントローラである．

図 4.1 に，本章で想定する遠隔ビル管理システムを示す．本章では，遠隔ビル管理システムの事業者が，遠隔ビル管理システムをパブリッククラウド上で構築することを想定する．クローラの挙動は 2 章や 3 章と同様である．フィードバック制御コントローラは，遠隔ビル管理システムのサーバ上で動作し，クローラが計測した監視点データを受け取り，制御値を計算し，ビルのゲートウェイに送り返す．制御信号のやりとりは，クローラと同様に，BACnet/WS [7] や IEEE1888 [40]，oBIX [71] などの遠隔監視制御向けの通信プロトコルを使う．図では，クローラやフィードバック制御コントローラは，左側のビルのみと通信しているが，実際には複数のビルと通信する可能性がある．

4.3　関連研究

本章では，関連研究としてパブリッククラウドの性能の安定性に関する研究と，監視制御アプリケーションの冗長化に関する研究について述べる．

4.3.1　パブリッククラウドの性能の安定性

パブリッククラウドの性能の安定性は，広く研究されている．多くの研究において，パブリッククラウドの性能は不安定であると述べられている．性能とは，具体的には以下である．

- **VM を作成する時間**
  パブリッククラウドでは，まずユーザは VM を作成する．作成にかかる時間が短く，安定しているほど，ユーザは自身のシステムを運用しやすくなる．文献 [73] は，Amazon EC2 において，VM の作成にかかる時間がばらつくことを検証している．

- **VM を起動する時間**
  ユーザは，VM を作成した後に，その VM を起動する．起動にかかる時間が短く，安定しているほど，ユーザは自身のシステムを運用しやすくなる．文献 [42, 63] は，Amazon EC2 や Google Compute Engine，Microsoft Azure において，VM を起動する時間がばらつくことを検証している．

- **計算性能**
  文献 [1] は，Amazon EC2 において，Hadoop の Map/Reduce ジョブを完了するまでの時間がばらつくことを検証している．

- **ストレージアクセスの性能**
  文献 [42, 61, 89] は，Amazon EC2 と Google Compute Engine において，ストレージのアクセス性能がばらつくことを検証している．

上記の性能の不安定さは，パブリッククラウドならでの，以下の要因によるものと考えられている．

- **VM 間の干渉**
  パブリッククラウドでは，ユーザが作成した VM は，一般的には「共有ホスト」に配置される．共有ホストとは，他のユーザが作成した VM も配置される可能性がある物理マシンである．単一の物理マシン上で複数の VM を実行した場合，VM 間の干渉により，互いの性能が低下することが知られている [39, 56, 76, 83]．

- **ライブマイグレーション**
  パブリッククラウドのプロバイダが，ユーザが作成した VM に対して「ライブマイグレーション」を実行している可能性がある．ライブマイグレーションとは，ある物理マシン上で稼働している VM のメモリや仮想デバイスの状態を，別の物理マシンに転送する技術である．これにより，稼働状態の VM 上の OS やアプリケーションを，稼働状態のまま，別の物理マシンに移動できる．ライブマイグレーションを実行中の VM の性能は低下することが知られている [2, 35]．

  パブリッククラウドのプロバイダがライブマイグレーションを実行する理由は 2 つ考えられる．1 つ目の理由は，メンテナンスのためである．ある物理マシンをメンテナンスするためには，その物理マシンを停止する必要がある．物理マシンを停止する前に，その物理マシン上で稼働している VM を，ライブマイグレーションにより別の物理マシンに移動する．もう 1 つの理由は，物理マシンのリソースの利用率向上のためである．パブリッククラウドのプロバイダとしては，稼働している物理マシンの数を減らしたい．そのほうが，物理マシン群のための電気料金を減らせるためである．そのためには，物理マシンのリソース利用率が最大になるように，VM を適切に物理マシンに配置する必要がある．VM が消費するリソース量は一定とは限らないため，リソース消費量の変動に応じてライブマイグレーションを実行することで，物理マシンのリソース利用率を向上できる．

- **Dynamic Voltage and Frequency Scaling (DVFS)**
  パブリッククラウドのプロバイダが，DVFS と呼ばれる技術により，物理マ

シンの CPU クロックを動的に調整している可能性がある．クロックを落としたほうが，消費電力を抑えられるためである．一方で，クロックを落とすと，VM の性能は低下する．

一部のパブリッククラウドのプロバイダは，ユーザに物理マシンを占有させるサービスを開始している．この物理マシン占有サービスを利用することで，上述の性能変動の要因を避けることができると考えられる．しかし，物理マシン占有サービスを利用した場合の性能評価を実施している研究は存在しない．そこで本章では，4.4 節で占有サービスの性能を調査する．

4.3.2　監視制御アプリケーションの冗長化

フィードバック制御コントローラの冗長化手法は，フィードバック制御コントローラと制御対象機器が物理的に近接している環境であれば，実用レベルに到達している [57, 81]．フィードバック制御コントローラと制御対象機器が近接しているため，冗長化構成にあるコントローラ同士も近接している．したがって，監視制御の状況を共有することや，フィードバック制御コントローラに異常が発生したことを検出することが容易である．また，監視制御を正確に行うためには，フィードバック制御コントローラと制御対象機器間の通信遅延を考慮する必要があるが，両者が近接していれば，通信遅延は無視できるほど小さくなる．したがって，各フィードバック制御コントローラが，自身と制御対象機器との間の通信遅延を把握する必要がない．

クラウド上で冗長化する場合，フィードバック制御コントローラ同士や，フィードバック制御コントローラと制御対象機器が近接するとは限らない．したがって，冗長構成にあるフィードバック制御コントローラ間で，監視制御の状況や，異常が発生したことを，どのように共有するかが重要となる．また，フィードバック制御コントローラと制御対象機器の間の通信遅延を無視できなくなるため，フィードバック制御コントローラごとに制御対象機器との間の通信遅延を把握する必要が生じる．通信遅延を考慮して遠隔地から監視制御を行う Networked Control System (NCS) の研究は多く存在する [3, 20, 27, 28, 30, 32, 41, 59, 91]．ただし，これらの

研究では，制御の品質は考慮しているが，フィードバック制御コントローラの冗長化は考慮していない．

Hegazy らは文献 [36] にて，ビル側に「制御情報」を記憶するゲートウェイを設置することで，フィードバック制御コントローラの冗長化を実現する手法を提案している．本手法では，クラウド上で複数のフィードバック制御コントローラを稼働させ，そのうちの 1 つを主系 (プライマリコントローラ) とする．プライマリコントローラは制御を行なう．ゲートウェイは制御情報として，いつ，どのフィードバック制御コントローラから制御信号を受けたかを記憶する．全てのコントローラは，ゲートウェイを経由して機器の状態と制御情報を取得する．制御情報を取得することで，各コントローラは，プライマリコントローラが最後に制御を実施した時刻を把握できる．そして，プライマリコントローラによる制御が一定時間行われてない場合に，プライマリコントローラを切り替える．この切り替え処理をフェイルオーバーと呼ぶ．各コントローラには固定の優先度が設定されており，その優先度に基づいて次のプライマリコントローラを決定する．

文献 [36] の手法は，ビル側に設置されるゲートウェイの機能に依存しており，ゲートウェイが Single Point Of Failure (SPOF) となりうる．ゲートウェイを冗長化することで SPOF ではなくなるが，その場合は，ゲートウェイ間で制御情報の一貫性を維持する仕組みが必要となる．また，文献 [36] では論じられていないが，実際にはゲートウェイは数百から数万の監視点を収容する可能性がある．また，監視制御を行うフィードバック制御コントローラは複数存在する．よってゲートウェイは最大で数万点分の制御情報を維持し，どの制御情報をどのコントローラに返すべきかを判断しなければならない．これらの要件は，ゲートウェイの実装を複雑化させるし，またゲートウェイに必要な計算資源を増加させる．よって，設備投資の削減や，システムの修正・更新が容易になるというクラウドコンピューティングのメリットを低減させる．

文献 [94] は，各フィードバック制御コントローラが自身の情報をブロードキャストし，その情報に基づいて，管理ノードがプライマリコントローラを選択する方法である．自身の情報として，CPU 使用状況を告知する点が特徴である．管理ノードは CPU に余裕があるフィードバック制御コントローラをプライマリコント

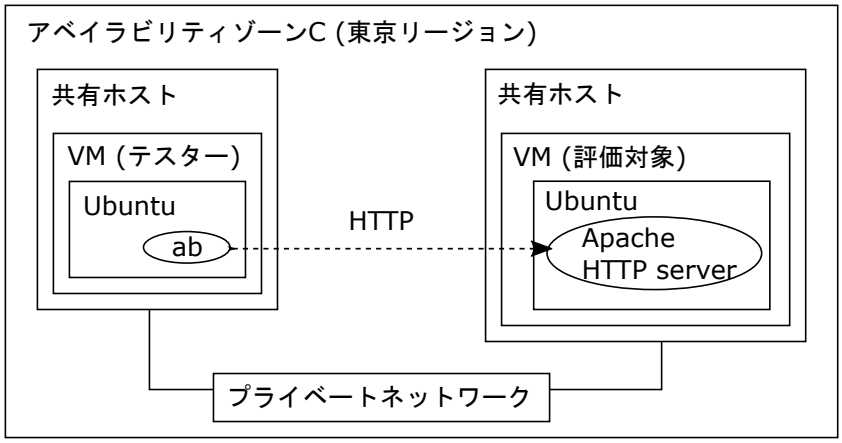


図 4.2: 共有ホストを用いた Apache HTTP サーバの性能評価の環境

ローラとして選択する．文献 [94] の手法は，管理ノードが SPOF となりうる．また，少なくとも管理ノードは全てのフィードバック制御コントローラのブロードキャストを受信し，状態を管理する必要があるため，管理ノードに処理負荷が集中する．さらに，全フィードバック制御コントローラが，同一のブロードキャストドメインに存在する必要がある．よって，ブロードキャストが届く範囲内でしか，フィードバック制御コントローラを冗長化できないという制約が生じる．

4.4　パブリッククラウドにおける性能の安定性の調査

本節では，パブリッククラウドにおける性能の安定性を調査する．パブリッククラウドとして Amazon EC2 を使う．性能を計測するためのアプリケーションとして，ウェブサーバを用いる．

4.4.1　共有ホストを用いた性能評価

ウェブサーバの性能を，共有ホストを用いて計測する．図 4.2 は，共有ホストを使用した場合の実験環境を表している．この環境では，評価対象の VM と，テスター用の VM を，異なる共有ホストにて起動する．どちらの VM も，共有ホストにて起動しているため，その性能は，別の VM の動作により干渉を受ける．評価

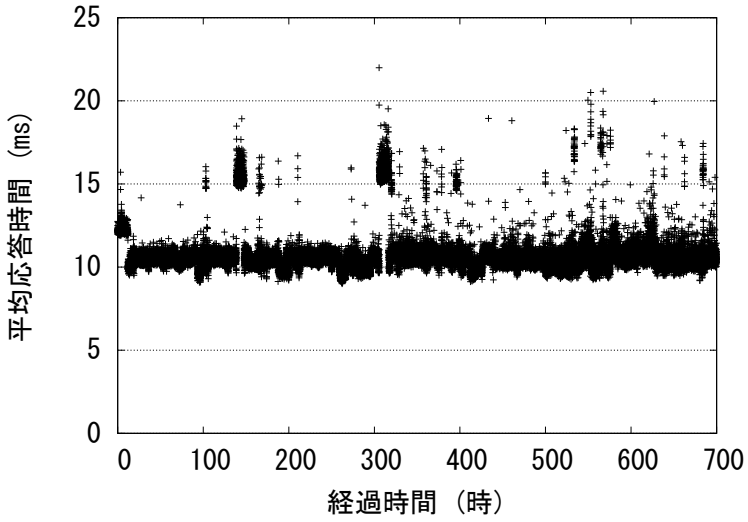


図 4.3: 共有ホスト上で稼働するウェブサーバの平均応答時間の推移

対象の VM のインスタンスタイプは m3.large，テスター用の VM のインスタンスタイプは m4.xlarge とする．また，どちらの VM も OS は Ubuntu 14.04.4 LTS とする．また，どちらの VM も東京リージョンのアベイラビリティゾーン C にて起動する．アベイラビリティゾーンは，複数のデータセンターから構築される．2 つの VM は，アベイラビリティゾーン C 内のプライベートネットワークにより接続される．

ウェブサーバとして Apache HTTP サーバ (バージョン 2.4.7) を使用する．BACnet/WS や IEEE1888 は機器の状態を XML でやりとりするため，それを想定した XML ファイルを用意する．XML ファイルのサイズは 3KB とする．

テスターには Apache HTTP サーバに付属するベンチマークツールである ab コマンドを用いる．ab コマンドは，毎分 100 個のコネクションを接続し，コネクションごとに 3000 個の HTTP GET リクエストを送信する．つまり，30 万個の HTTP GET リクエストを送信する．HTTP GET リクエストの対象は前述の XML ファイルとする．そして，30 万回のファイル取得における平均応答時間を計測する．この評価を 30 日間，続けて実施する．

図 4.3 に共有ホスト上で稼働するウェブサーバの，平均応答時間の推移を示す．図 4.3 から，多くの時間において，平均応答時間が 10 ミリ秒から 11 ミリ秒になっ

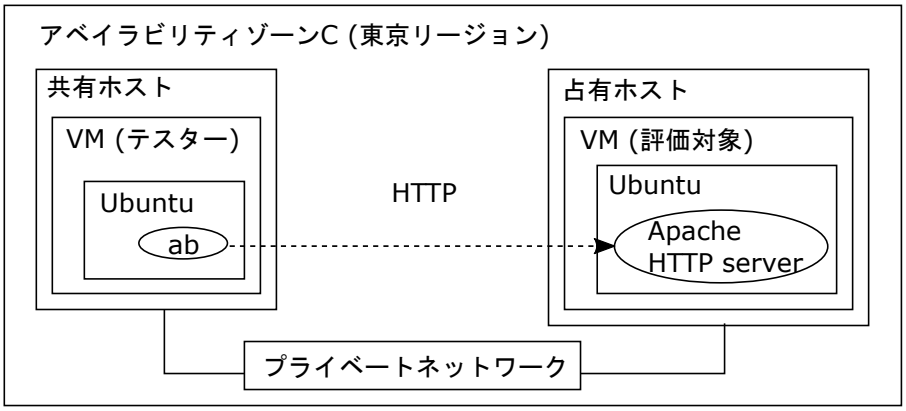


図 4.4: 占有ホスト上で稼働するウェブサーバの平均応答時間の推移

ていることがわかる．一方で，平均応答時間が 15 ミリ秒を超える場合や，20 ミリ秒を超える場合も存在した．この結果から，共有ホスト上のアプリケーションの性能は，一時的に低下する場合があると言える．

4.4.2　占有ホストを用いた性能評価

ウェブサーバの性能を，占有ホストを用いて計測する．図 4.4 は，占有ホストを使用した場合の実験環境を表している．この環境では，評価対象の VM は占有ホストにて起動し，テスター用の VM を共有ホストにて起動する．したがって，評価対象の VM の性能は，別の VM の動作により干渉は受けないが，テスター用の VM の性能は，別の VM の動作により干渉を受ける．評価対象の VM のインスタンスタイプは m3.large，テスター用の VM のインスタンスタイプは m4.xlarge とする．また，どちらの VM も OS は Ubuntu 14.04.4 LTS とする．また，どちらの VM も東京リージョンのアベイラビリティゾーン C にて起動する．2 つの VM は，アベイラビリティゾーン C 内のプライベートネットワークにより接続される．Apache HTTP Server や `ab` コマンドの使い方は，4.4.1 項と同じである．

図 4.5 に占有ホスト上で稼働するウェブサーバの，平均応答時間の推移を示す．図 4.5 から，多くの場合において，平均応答時間が約 11 ミリ秒になっていることがわかる．平均応答時間が 12 ミリ秒を超えた回数は 16 回であり，約 0.03%の確率であった ($\frac{16}{60\times24\times30}$)．

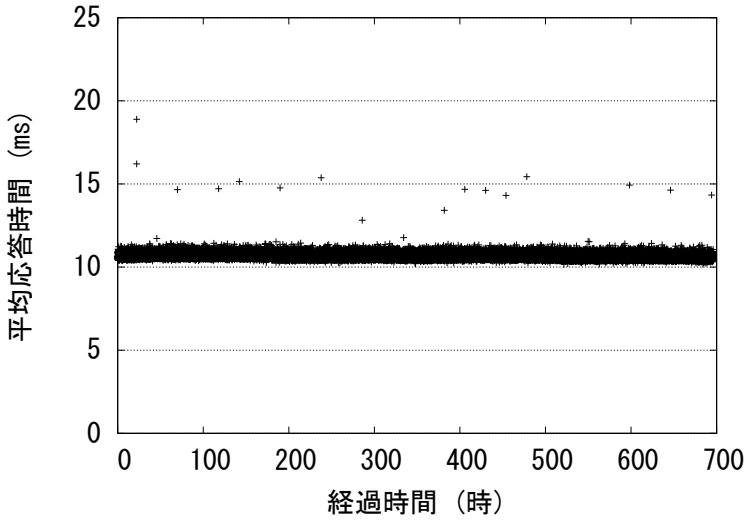


図 4.5: 占有ホスト上で稼働するウェブサーバの平均応答時間の推移

表 4.1: 共有ホストと占有ホストの平均応答時間の統計値 (単位はミリ秒)

	最小値	平均値	最大値	標準偏差値
共有ホスト	8.999	10.719	21.991	1.141
占有ホスト	10.169	10.639	18.889	0.191

4.4.3 共有ホストと占有ホストの性能の比較

図 4.6 に，共有ホストを使用した場合と占有ホストを使用した場合の平均応答時間の Cumulative Distribution Function (CDF) を示す．また，表 4.1 に平均応答時間の統計値を示す．占有ホストを使用した場合の平均応答時間の多くは，10 ミリ秒から 12 ミリ秒の間であり，共有ホストを使用した場合の平均応答時間と比べてばらつきが小さい．4.4.1 項と 4.4.2 項で実施した 2 つの評価では，テスターは共有ホスト上の同一の VM を使用した．そのため，平均応答時間のばらつきの差は，共有ホストと占有ホストの違いにより生じていると考えられる．共有ホストの場合は，別のユーザの VM が稼働する可能性があり，別のユーザの VM の挙動によっては，ウェブサーバの応答処理速度が低下する．一方で占有ホストの場合は，別のユーザの VM が稼働することはない．

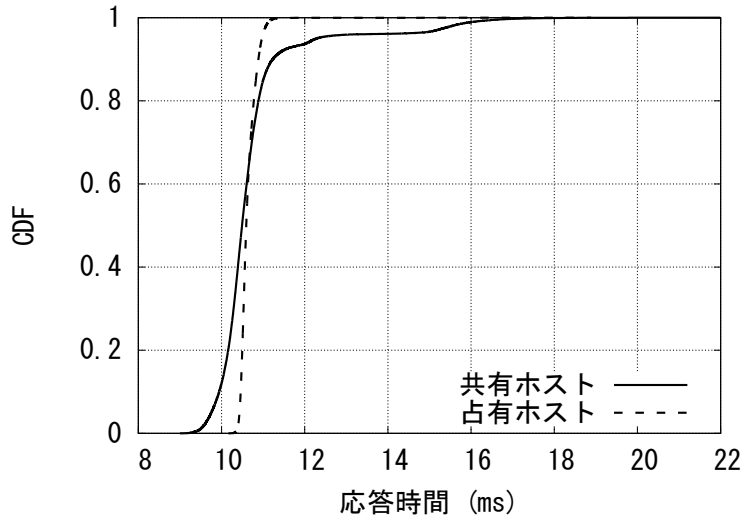


図 4.6: 共有ホストと占有ホストの応答時間の CDF

本節の評価から，共有ホストと占有ホストの性能の安定性には差があることを確認できた．監視制御アプリケーションは性能の安定性を要求するため，監視制御アプリケーションをパブリッククラウドにおいて実行する場合は占有ホストを利用することが望ましいと言える．

4.5　提案手法

本節では，フィードバック制御コントローラを冗長化することで，その稼働率を向上させる方法を説明する．

4.5.1　提案手法の概要

提案手法を適用したシステム構成の一例を図 4.7 に示す．HMI や DB サーバなどは省略している．また，データ要求の矢印も省略している．文献 [36] で提案されている手法と同様に，複数のクラウドのリージョンを利用してフィードバック制御コントローラを冗長化する．各リージョンのクラウドのプロバイダは同一で

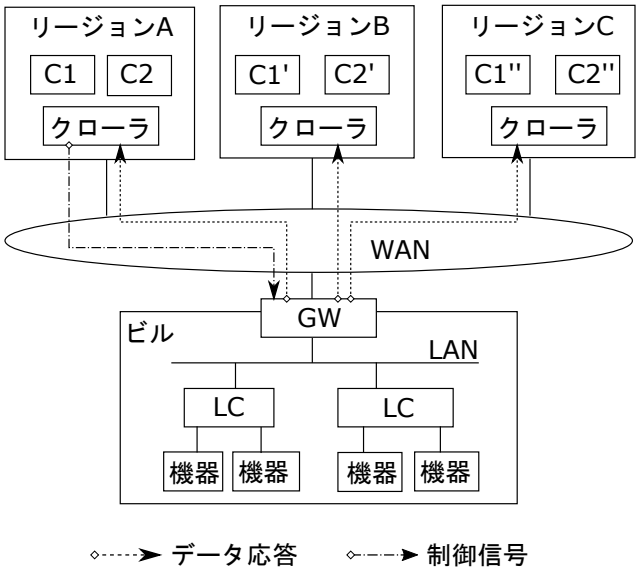


図 4.7: 提案手法を適用したシステムの構成図

よい．VM の稼働率を個別に保証できればよい．例えば Amazon EC2 はリージョンごとに稼働率を保証するため，複数のリージョンを利用すればよい．

図 4.7 において，C1 や C2 はフィードバック制御コントローラであり，個別の VM 上で稼働する．C1’ と C1” は C1 の，C2’ と C2” は C2 のバックアップ用のフィードバック制御コントローラである．各リージョンにはクローラが動作しており，全てのフィードバック制御コントローラはクローラから監視点データを受け取る．そして，プライマリコントローラだけが制御を行う．

提案手法では，フィードバック制御コントローラ間で生存確認用のメッセージ (ハートビート) を定期的にユニキャストで交換することで，他のコントローラが正常に動作しているかを把握する．例えば，図 4.7 では，C1 と C1’ と C1” がハートビートを交換する．そして，プライマリコントローラの異常や，プライマリコントローラとゲートウェイの間のネットワーク異常を検出したらフェイルオーバーする．例えば WAN やコントローラ C1 に障害が発生したら，C1 の代わりに C1’ または C1” が監視制御を引き継ぐ．また，各フィードバック制御コントローラは，自身とゲートウェイの間のネットワーク (制御用ネットワーク) の通信遅延などを把握しておき，それをハートビートに含めて他のコントローラと共有する．通信

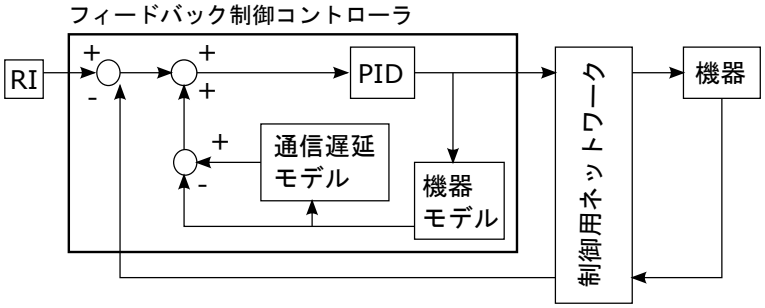


図 4.8: フィードバック制御コントローラの制御ロジック

遅延を考慮して新しいプライマリコントローラを選択することにより，フェイルオーバー時の制御品質の低下を抑える．

このように提案手法では，フィードバック制御コントローラ同士の通信のみでフェイルオーバーが可能であるため，ゲートウェイが制御情報を記録しなくてよい．これにより，ゲートウェイは単なるルータまたはプロトコル変換装置でよくなり，Keepalived や Virtual Router Redundancy Protocol (VRRP) [66] などの既存手法により冗長化できる．よって，システムから SPOF を除外できる．

4.5.2　フィードバック制御処理

各フィードバック制御コントローラは，スミス予測器 [82] を用いた PID 制御を行う．図 4.8 に，制御ロジック図を示す．RI は Reference Input の略で，制御で目指すべき状態 (目標値) を意味する．スミス予測器は，通信遅延が制御品質に与える悪影響を除外するために，制御対象の機器の状態推移モデルと，通信遅延の値を必要とする．各コントローラは，フィードバック制御処理として，以下の処理を定期的に行う．

1. 監視点データの取得

   監視点データは，クローラが計測してくれているため，それを取得する．クローラから直接取得してもよいし，データベースを介して取得してもよい．最新の監視点データを取得できなければ，正確な制御は行えない．そこで，N_{data} 回連続で監視点データを取得できない場合に，フィードバック制御コ

ントローラは自身を異常だと判定することとする．プライマリコントローラが，自身を異常だと判定した場合，プライマリコントローラの選択処理を行う．プライマリコントローラの選択処理は 4.5.4 項で述べる．

N_{data} の値が小さいと，ネットワークの不調やアプリケーションの動作の乱れに対して敏感になる．フェイルオーバーを素早く開始するため，異常による制御品質の低下を抑えやすい．しかし，異常が一時的なものである場合は，フェイルオーバーしないほうが，制御品質の低下を抑えられる可能性もある．一時的な異常に対してフェイルオーバーしないためには，N_{data} の値を大きくすればよいが，フェイルオーバーを開始するまでの時間が遅くなる．制御品質の低下をできるだけ抑えられるよう，N_{data} の値を適切に定める必要がある．

2. **通信遅延の取得**
   スミス予測器は現在の通信遅延の値を使用して制御値を調整するため，通信遅延を取得する必要がある．通信遅延を取得する方法はいくつか考えられる．クローラが通信遅延を計測して，それをフィードバック制御コントローラに伝える方法がある．フィードバック制御コントローラが制御値を送信する際に，通信遅延を計測してもよい．また，`ping` コマンドなどを用いた計測を，別途実行してもよい．

3. **制御値の計算**
   自身がプライマリコントローラであれば，取得した監視点データと通信遅延から，制御値を計算する．そして，制御値をゲートウェイに送信する．

4.5.3 生存確認処理

各フィードバック制御コントローラは，生存確認処理として，以下の処理を一定間隔 T_{hb} で実施する．

1. **ハートビートの送信**
   フィードバック制御コントローラは，他のフィードバック制御コントローラ

群に，ハートビートを送信する．ハートビートには，タイムスタンプや自身の識別子，自身の役割，自身の状態を含める．自身の役割とは，プライマリコントローラまたはバックアップコントローラのどちらかである．自信の状態とは，正常か異常かのどちらかである．また，フェイルオーバー時の制御品質の低下を抑えるため，自身の制御用ネットワークの通信遅延や，制御の情報もハートビートに含める．制御の情報とは,「制御の目標値」と「制御対象機器の状態」の差の積算値である．

2. ハートビートの受信
   ハートビートを受信したフィードバック制御コントローラは，ハートビートに含まれている情報を参照し，記憶する．ハートビートの送信元がプライマリコントローラで，かつ，異常状態であると記されている場合は，プライマリコントローラの選択処理を行う．プライマリコントローラの選択処理は 4.5.4 節で述べる．

3. 受信確認の返信
   ハートビートを受信したフィードバック制御コントローラは，ハートビートの送信元に受信確認を返す．

4. 受信確認の受信
   ハートビートを送信したフィードバック制御コントローラは，受信確認を受信することで，相手のコントローラが正常であることを確認する．一方，N_{ack} 回連続で受信確認を得られない場合は，相手のコントローラが異常であると判定する．ハートビートを送信してからタイムアウト時間 T_{ack} が経過しても受信確認が返信されてこない場合に，受信確認を得られないと判断する．

4.5.4　プライマリコントローラの選択処理

各コントローラは，プライマリコントローラ自身の異常や，プライマリコントローラの制御用ネットワークの異常を検知した場合に，次のプライマリコントローラ (新プライマリコントローラ) の選択処理を行う．状態が正常なフィードバック

表 4.2: 新プライマリコントローラを決定するための式 4.1 の変数

変数	説明
D_c	候補のコントローラの制御用ネットワークの平均通信遅延
D_p	現在のプライマリコントローラの制御用ネットワークの平均通信遅延
S_c	候補のコントローラの制御用ネットワークの通信遅延の標準偏差
E_c	候補のコントローラの制御用ネットワークにおける通信エラー率
a, b, c	チューニング用の係数

制御コントローラが，新プライマリコントローラの候補となる．プライマリコントローラの候補が複数存在する場合は，通信品質を考慮して，新プライマリコントローラを決定する．通信品質は，フィードバック制御の品質に影響を与えるためである．各コントローラの通信品質は，ハートビートに付加することで共有する．

各フィードバック制御コントローラは，自身も含めた全てのコントローラに対して，スコア S を計算する．そしてスコア S が最も小さいコントローラを，新プライマリコントローラとして決定する．スコア S は以下の式で計算する．

$$S = a|D_c - D_p| + bS_c + cE_c \tag{4.1}$$

式 (4.1) の変数の説明を表 4.2 に示す．第一項は，現在のプライマリコントローラと，候補のコントローラの制御用ネットワークの平均通信遅延の差である．第二項は，候補のコントローラの制御用ネットワークの通信遅延の標準偏差値を考慮する．第三項は通信エラー率である．いずれの項も，小さいほうが，フィードバック制御の品質はよくなる．通信エラーについては，アプリケーションレベルで観測してもよいし，OSI の 7 階層におけるトランスポートレイヤにて観測してもよい．例えば，Transmission Control Protocol (TCP) の再送が発生した頻度から計算してもよい．

表 4.3: 提案システムの稼働率に関わる変数

変数	説明
P_r	パブリッククラウドのプロバイダが保証する，リージョンの稼働率
P_w	WAN の稼働率
D_r	リージョンの冗長度

4.5.5　提案システムの稼働率

本項では図 4.7 で示した提案システムの稼働率について述べる．2 章や 3 章では，WAN の信頼性は非常に高いという環境を想定したが，本節ではその稼働率を具体的に想定する．

稼働率の計算に関連する変数を表 4.3 にまとめる．リージョンの稼働率の定義はパブリッククラウドのプロバイダにより異なるが，ここでは Amazon EC2 の定義である「月間使用可能時間の割合」を採用する[2]．すなわち，リージョンに対して外部からアクセスできる時間に基づいて稼働率を計算する．

ビル内の機器やネットワークの稼働率は従来と同じであるから，ここでは，それらの稼働率は十分に高いとする．したがって，提案手法を適用した場合，いずれかのリージョンと，WAN が稼働していれば，システム全体として動作できると言える．いずれかのリージョンが稼働している確率は $1-(1-P_r)^{D_r}$ である．よってシステム全体の稼働率 P_s は式 (4.2) で計算できる．

$$P_s = (1-(1-P_r)^{D_r})P_w \tag{4.2}$$

リージョンの冗長度として 1 から 3，リージョンの稼働率として 99.95%と 99.99%，WAN の稼働率として 99.9990%と 99.9995%と 99.9999%を想定し，P_s を試算した．その結果を表 4.4 に示す．各リージョンとビルは WAN を介して接続されるため，P_s は P_w よりも小さくなる．つまり P_w が 99.999%よりも大きくなければ，99.999%を達成することはできない．今回の試算結果では，P_w が 99.9995%で，かつ，リージョンの冗長度が 2 であれば 99.999%を達成できた．

[2]https://aws.amazon.com/jp/ec2/sla/ (2017 年 10 月時点)

表 4.4: システム全体の稼働率 (99.999%以上の稼働率は太字で表記)

D_r	P_r	P_w		
		0.999990	0.999995	0.999999
1	0.9995	0.999490	0.999495	0.999499
2	0.9995	0.999989	**0.999994**	**0.999998**
3	0.9995	0.999989	**0.999994**	**0.999998**
1	0.9999	0.999890	0.999895	0.999899
2	0.9999	0.999989	**0.999994**	**0.999998**
3	0.9999	0.999989	**0.999994**	**0.999998**

99.9995%の稼働率を保証する WAN のサービスは，本論文の執筆時点では，我々が調べた限りでは見つからなかった．ただし信頼性の向上に向けた研究開発は進められている．例えば，Internet Engineering Task Force (IETF) で標準化が進められている Deterministic Networking (DetNet) [25, 31] は，PCE [23] や MPLS [78] などの経路制御および帯域予約の手法を活用して，ビル管理システムや電力系統網監視システムに使用できる品質の WAN を実現しようとしている．そのため，将来的には 99.9995%や 99.9999%の稼働率を保証する WAN サービスが実現されると考える．その WAN サービスと提案手法を組み合わせることで，パブリッククラウドを利用した稼働率 99.999%以上の監視制御システムを実現できると考える．

4.6 評価

本節では，提案手法の評価について述べる．

4.6.1 評価の環境

提案手法を用いた場合のフェイルオーバー時の制御品質を評価するために，フィードバック制御コントローラと機器を試作した．評価環境を図 4.9 に示す．フィードバック制御コントローラと監視制御対象の機器は，ソフトウェアで実装した．パ

表 4.5: 評価における各パラメータの値

	値	概要
計測周期	1	監視点データの計測周期 (秒)
目標値	1	フィードバック制御の目標値
N_{data}	3	制御用ネットワークを異常だと判断する連続計測失敗数
N_{ack}	5	コントローラを異常だと判断する受信確認の連続受信失敗数
T_{hb}	1	ハートビートの送信間隔 (秒)
T_{ack}	1	受信確認が得られないと判断するまでのタイムアウト時間 (秒)

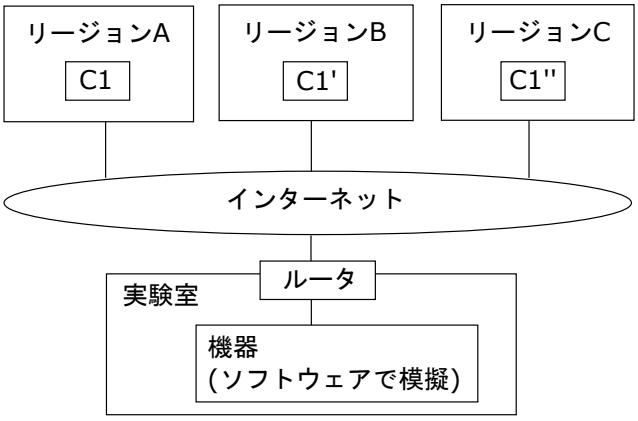


図 4.9: 評価環境

ブリッククラウドとして Amazon EC2 を使用し，フィードバック制御コントローラは t2.micro インスタンス上で実行した．VM 上で実行する OS は Ubuntu 14.04.3 とした．機器は，神奈川県にある実験室にて設置および実行した．

試作したフィードバック制御コントローラは，機器の状態を目標値に維持するよう，PID 制御を行う．PID 制御のチューニングには AMIGO 法 [34] を用いる．本来であればクローラが監視点データを計測し，フィードバック制御コントローラは制御のみを行うが，本評価では監視点データの計測機能をフィードバック制御コントローラに組み込む．コントローラは監視点データを計測するたびに通信遅延を測定し，都度，AMIGO 法によるチューニングを行う．

プライマリコントローラを選択するための式式 4.1 の a，b，c は全て 1 とした．

表 4.6: ゲートウェイと各リージョンの Round Trip Time (RTT) の統計値 (ミリ秒)

宛先のリージョン	最小値	平均値	最大値	標準偏差
東京	20.921	22.439	43.021	0.888
北カリフォルニア	115.249	117.586	152.450	3.167
フランクフルト	292.482	294.305	321.668	1.491

表 4.7: ゲートウェイと各リージョンの通信路の信頼性

宛先のリージョン	正常に ICMP エコー応答が得られた確率
東京	99.9977%
北カリフォルニア	99.9945%
フランクフルト	99.9701%

そのほかの評価におけるパラメータは表 4.5 に示す.

機器は，水冷式の空調の水流を調整するバルブを想定した．水流調整バルブの制御モデルは，以下の伝達関数で表される 1 次遅れ系とした.

$$G(s) = \frac{K}{1 + sT} \tag{4.3}$$

K はゲイン，T は時定数である．評価において，$K = 0.7$，$T = 20$ とし，10 ミリ秒で離散化した.

4.6.2 評価の結果

本項では，評価結果について述べる.

各リージョンとゲートウェイの間の通信品質

提案手法の評価を行う前に，まず，ゲートウェイと Amazon EC2 の各リージョンとの間の通信品質を調査した．ここでの通信品質とは，通信遅延と通信エラー率である．いずれも，フィードバック制御の品質に影響を与える要因である．

リージョンとしては，東京リージョン，北カリフォルニアリージョン，フランクフルトリージョンの 3 リージョンを選定した．各リージョンで VM を実行した状態で，`ping` コマンドを使用し，ゲートウェイから VM に ICMP エコー要求を送信し，Round Trip Time (RTT) を計測した．また，ICMP エコー応答を受け取れなかった場合を通信エラーとした．ICMP エコー要求は 1 秒間隔で送信し，評価期間は一週間とした．すなわち，ICMP エコー要求を 604800 回送信した．

評価により得られた RTT の統計値を表 4.6 に示す．このように，各リージョンとゲートウェイとの間の通信遅延には，差があることがわかった．ゲートウェイは神奈川に設置されているため，物理的に近い東京リージョンとの間の通信遅延が最も小さく，そのばらつきも最も小さかった．また，ICMP エコー応答を受信できた確率を表 4.7 に示す．通信遅延が小さいリージョンのほうが，ICMP エコー応答を受信できた確率も高かった．

制御情報の共有が制御品質に与える影響

PID 制御やスミス予測器の情報 (制御情報) の共有が，フェイルオーバー時の制御品質に与える影響を評価する．これらの状態は，制御値の計算に使用されるため，制御品質に影響すると考えられる．この評価では，プライマリコントローラを東京リージョンに，バックアップコントローラを北カリフォルニアリージョンとフランクフルトリージョンに配置した．そして，評価開始から 10 秒後に，プライマリコントローラとゲートウェイの間の通信を故意に遮断した．コントローラ間の通信は遮断しないため，ハートビートは正しく送受信できる．監視点データの計測間隔は 1 秒，$N_{data} = 3$ であるから，約 13 秒の時点でプライマリコントローラが自身を異常と判断し，そのあとにフェイルオーバーが発生すると想定される (4.5.2 項)．

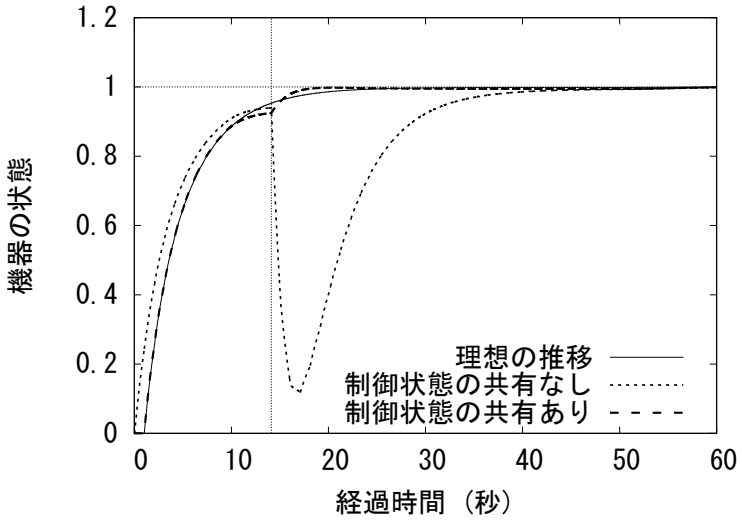


図 4.10: 制御情報の共有の有無による，フェイルオーバー時の機器状態の変動の比較

図 4.10 は評価時の制御対象機器の状態の推移を表している．図中の「理想の推移」とは，プライマリコントローラに異常が発生しない場合の，機器の状態推移である．この状態推移に近いほど，制御品質が高いと言える．

評価では，まず，約 13 秒の時点で東京リージョンのプライマリコントローラが自身を異常だと判定した．そして，プライマリコントローラの選択処理を開始した．プライマリコントローラからのハートビートを受けた北カリフォルニアリージョンとフランクフルトリージョンのバックアップコントローラも，プライマリコントローラの選択処理を開始した．式 4.1 を用いたスコア計算を行った結果，北カリフォルニアリージョンのバックアップコントローラがプライマリコントローラとして選択された．北カリフォルニアリージョンはフランクフルトリージョンよりも，ゲートウェイとの間の通信遅延の値が，東京リージョンに近いためである．

制御情報を共有しない場合は，バックアップコントローラがプライマリコントローラに昇格した直後に，制御対象機器の状態が大きく変化した．一方，制御情報を共有することで，制御対象機器の状態推移を，理想の推移に近づけることができた．制御情報を共有することで，新しいプライマリコントローラが，これまでの制御の経緯を考慮して制御値を計算できたためである．本結果から，制御情

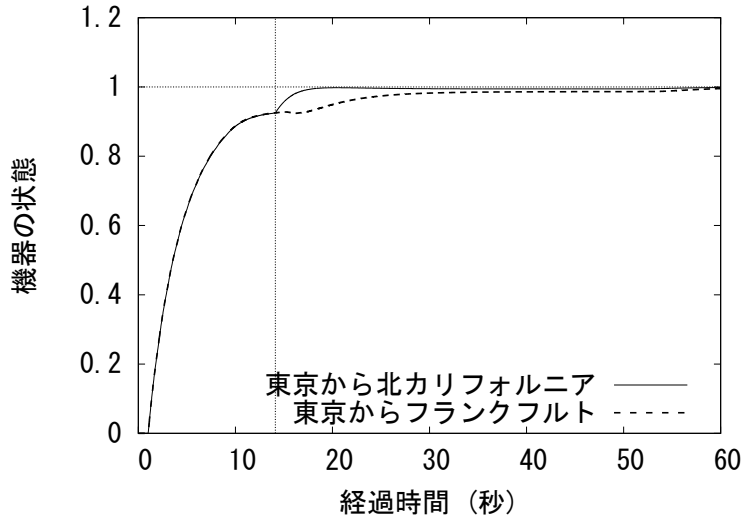


図 4.11: 東京リージョンから別のリージョンにフェイルオーバーする場合の機器の状態

報を共有しておくことで，フェイルオーバー時の機器状態の変動を軽減できると言える．

通信遅延が制御品質に与える影響

表4.6で示したように，Amazon EC2の各リージョンとゲートウェイとの間の通信遅延には差がある．この遅延の差が，フェイルオーバー時の制御品質に影響を与えるかを評価する．

本評価では，評価開始から10秒後に、プライマリコントローラとゲートウェイの間の通信を故意に遮断した．図4.11に，東京リージョンから北カリフォルニアリージョンにフェイルオーバーした場合の機器状態の推移と，フランクフルトリージョンにフェイルオーバーした場合の機器状態の推移を示す．どちらの場合も、目標値に収束させることはできているが，北カリフォルニアリージョンにフェイルオーバーした場合のほうが速く収束させられている．

図4.12に，フランクフルトリージョンから北カリフォルニアリージョンにフェイルオーバーした場合の機器状態の推移と，東京リージョンにフェイルオーバー

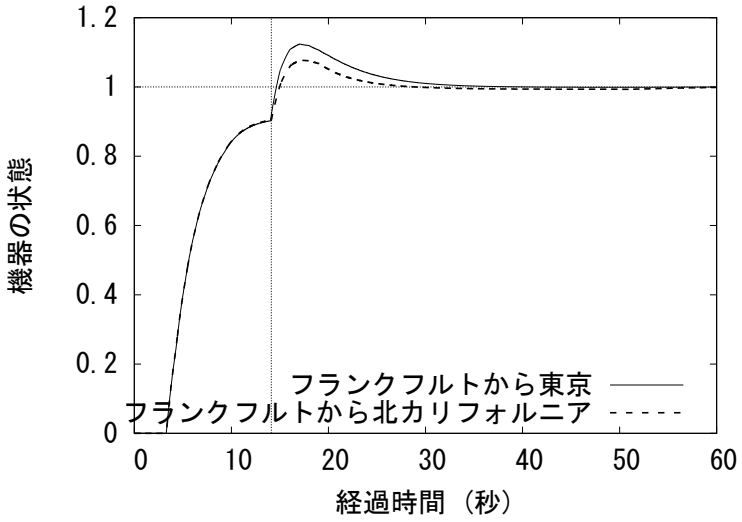


図 4.12: フランクフルトリージョンから別のリージョンにフェイルオーバーする場合の機器の状態

した場合の機器状態の推移を示す．どちらもフェイルオーバー直後の機器状態の変動は大きいが，北カリフォルニアリージョンにフェイルオーバーした場合のほうが変動が小さいことがわかる．今回，PID 制御のパラメータチューニングには AMIGO 法 [34] を使用している．AMIGO 法は，通信遅延が小さいほどゲインパラメータが大きくなる特徴がある．そのため，通信遅延が小さいリージョンにフェイルオーバーした直後は，フェイルオーバー前よりも制御値を大きく設定する傾向にあり，機器状態の大きな変化につながると考える．図 4.11 と図 4.12 の結果から，フェイルオーバー先は，制御対象機器までの通信遅延が小さい場所よりも，現状と比べて，制御対象機器までの通信遅延の変化が小さい場所が適切であると言える．

プライマリコントローラの異常停止が制御品質に与える影響

これまでの評価では，プライマリコントローラと制御対象機器の間の通信は遮断されているが，プライマリコントローラのハートビートはバックアップコントローラに届く状況を想定した．そのため，プライマリコントローラのハートビー

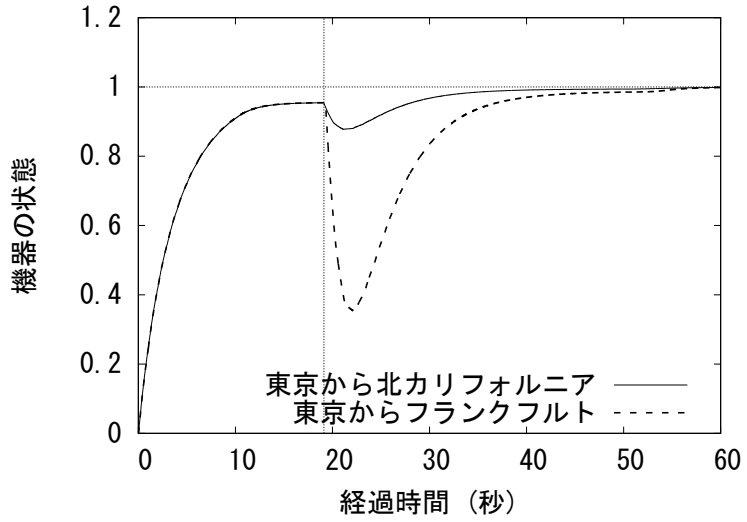


図 4.13: プライマリコントローラの異常停止時が制御品質に与える影響

トを受信することで，バックアップコントローラはプライマリコントローラの異常を知ることができた．この評価では，プライマリコントローラそのものが異常となり，ハートビートを送信できなくなる状況を想定する．そこで，評価開始から10秒後に，プライマリコントローラを故意に停止する．北カリフォルニアリージョンにフェイルオーバーする場合と，フランクフルトリージョンにフェイルオーバーする場合を評価する．

図4.13に評価結果を示す．T_{hb} を1秒，T_{ack} を1秒，N_{ack} を5としたため，想定通り，約19秒の時点で，5回目の受信応答待ちのタイムアウトとなり，フェイルオーバーが行われた．図4.11と比べて，異常が発生してからフェイルオーバーを実行するまでの時間が長い．その間，制御信号は送られないため，理想の状態推移との乖離が大きくなっていく．そのため，フェイルオーバーを完了した直後の制御による機器状態の変動が大きい．4.5.2項で述べたように，ネットワークやプライマリコントローラを異常だと判定するための閾値 (N_{data} や N_{ack}) を適切に設定することで，制御品質の低下を抑えることが重要である．

4.7 設備投資の削減効果の検証

クラウドコンピューティングを利用するメリットの 1 つは，設備投資の削減である．設備投資の削減は定量化しやすく，わかりやすいメリットである．そこで本節では，Amazon EC2 を使用した場合に，本当に設備投資が減るかを試算する．なお，Amazon EC2 の料金はドルで計算されるため，試算はドルを基準通貨として行う．また 1 ドル 100 円とする．

4.7.1 設備投資のモデル

コストのモデルを説明する．パブリッククラウドを用いたビル管理システムにおいて，一年間に発生するコスト C_{all} は，VM を実行することで発生するコスト C_{av}，VM が通信することにより発生するコスト C_{ac}，WAN を利用することにより発生するコスト C_{aw} の合計となる．

$$C_{all} = C_{av} + C_{ac} + C_{aw} \tag{4.4}$$

総設備投資 C_{all} の試算に必要となる変数を表 4.8 にまとめる．C_{dh} は，占有ホストの使用により発生するコストである．Amazon EC2 の課金モデルに従い，一時間ごとにコストが発生するモデルとする．Amazon EC2 の占有ホストのコストは，占有ホスト上で稼働する VM 数に依存しないが，占有ホスト上で実行できるインスタンス数には上限がある[3]．C_{ex} は，Amazon EC2 からビルに対して通信した際に発生するコストである．Amazon EC2 からインターネットに出ていくトラヒック量に対してのみ課金される．ビルあたりのフィードバック制御コントローラ数や監視点データの計測間隔は，実際にはビルや監視点ごとに異なる可能性があるが，ここでは単一の値とする．

表 4.8 の変数に基づき，C_{av} は以下の式で計算される．

$$C_{av} = 365 \times 24 N_{dh} C_{dh} \tag{4.5}$$

[3]インスタンスの種類ごとに，上限は異なる．

表 4.8: コストの試算に使用する変数

変数	説明
N_b	管理するビルの数
N_{cpb}	ビルあたりのフィードバック制御コントローラ数
N_c	フィードバック制御コントローラの総数 ($N_b N_{cpb}$)
N_s	各フィードバック制御コントローラが収容する監視点数
D_r	フィードバック制御コントローラの冗長度
N_{vm}	全 VM 数 ($N_c D_r$)
N_{vmdh}	占有ホストが実行できる VM 数．占有ホストのスペックにより決まる．
N_{dh}	全占有ホスト数 ($\lceil \frac{N_c}{N_{vmdh}} \rceil D_r$)
C_{dh}	1 台の占有ホストを使用することで発生するコスト (ドル/時)
I_d	監視点データの計測間隔 (秒)
I_{hb}	ハートビートの送信間隔 (秒)
F_d	1 度の計測処理で，クラウド外部に対して発生するトラヒック (バイト)
F_{hb}	1 度のハートビートと受信応答の送受信で発生するトラヒック (バイト)
C_{ex}	VM がクラウド外部に対して通信するためのコスト (ドル/ギガバイト)
C_{in}	VM がリージョン間で通信するためのコスト (ドル/ギガバイト)
C_{wan}	WAN の 1ヶ月分の契約料

C_{ac} は以下のように，監視点データを計測することで発生するコスト C_{as} とハートビートを交換することで発生するコスト C_{ah} の合計となる．

$$C_{ac} = C_{as} + C_{ah} \tag{4.6}$$

C_{as} は総監視点データ数と計測頻度から計算できる．

$$C_{as} = \frac{365 \times 24 \times 60 \times 60}{1000000000 I_d} F_d N_{vm} C_{ex} \tag{4.7}$$

同様に，C_{ah} は以下の式で計算される．

$$C_{ah} = \frac{365 \times 24 \times 60 \times 60}{1000000000 I_{hb}} F_{hb} N_{vm} D_r C_{in} \tag{4.8}$$

C_{aw} はビルの数と WAN の契約費から，以下のように計算される．

$$C_{aw} = 12N_bC_{wan} \tag{4.9}$$

一方，パブリッククラウドを使用せずに，ビル内にビル管理システムを構築した場合，一年間に発生する設備投資 C_{ab} は，コントローラのハードウェア費 C_{ctrl} とその耐用年数 Y から計算できる．

$$C_{ab} = \frac{N_cD_rC_{ctrl}}{Y} \tag{4.10}$$

上記の式において，コントローラ数を N_cD_r としている理由は，ビル内においてもコントローラを冗長化することを想定するためである．

4.7.2 設備投資の試算

表 4.9 に，設備投資の試算に用いる変数の値をまとめる．試算にあたり，ビルのコントローラの数 N_b として 1 棟から 100 棟までを想定する．中小ビルを想定し，1 ビルあたり 10 個のフィードバック制御コントローラとする．また，1 コントローラあたり 10 個の監視点を監視制御する設定とする．4.5.5 項で考察したように，WAN の稼働率が高ければ，リージョンの冗長度は 2 で十分であるため，D_r は 2 とする．パブリッククラウドの料金は Amazon EC2 の現行の料金を使用する．占有ホストとしてタイプ「m3」を選択する．m3 は，mediam インスタンスを 32 個実行できる．この数は，占有ホストの中では最大である．F_d と F_{hb} は，試作したフィードバック制御コントローラの実装に合わせて 200 バイトとする．現時点で，本論文が想定する高信頼な WAN のサービスは存在しないため，WAN の使用料 C_{wan} は複数の値を想定する．

設備投資の試算結果を図 4.14 に示す．ビル内にビル管理システムを構築する場合は，コントローラの耐用年数 Y を 5 年，コントローラの価格を 50 万円 (5000 ドル) として試算した．ビルの数が増えると，必要なコントローラの数が増えるため，設備投資が増加することは明らかである．図 4.14 から，WAN の契約費が 1500 ドル/月の場合は，ビル内にビル管理システムを設置する場合よりも設備投資が高く

表 4.9: 設備投資の試算時の各変数の値

変数	説明
N_b	1 から 100
N_{cpb}	10
N_c	10 から 1000
N_s	10
D_r	2
N_{vm}	20 から 2000
N_{vmdh}	32 (m3 は，medium インスタンスを 32 個実行可能)
C_{dh}	1.292 ドル/一時間 (Amazon EC2 の占有ホストの m3 タイプの 3 年間契約)
N_{dh}	2 から 64
I_d	1 秒
I_{hb}	1 秒
F_d	200 バイト
F_{hb}	200 バイト
C_{ex}	0.14 ドル/ギガバイト
C_{in}	0.09 ドル/ギガバイト
C_{wan}	500 ドル/月，1000 ドル/月，1500 ドル/月

なることがわかる．一方，WAN の契約費が 1000 ドル/月または 500 ドル/月の場合は，パブリッククラウドを利用したほうが安くなった．

図 4.15 に，設備投資の内訳の比率を示す．図 4.15 から，契約費が 1000 ドル/月または 1500 ドル/月の場合は，総コストにおいて WAN の契約費が支配的であることがわかる．例えば，KDDI の国内イーサネット専用サービスは 10Mbps の帯域保証型の WAN サービスであり，月当たりの契約費は 170,000 円 (1700 ドル) である[4]．また，CTC の広域イーサーネットサービス「CTC EtherLINK」も，同様のサービスであり，10Mbps の帯域を保証できる[5]．このサービスの月当たりの契約

[4]http://www.kddi.com/business/network/intranet/ethernet-senyo/charge/10mbps/
[5]http://business.ctc.jp/service/network/etherlink/charge.html

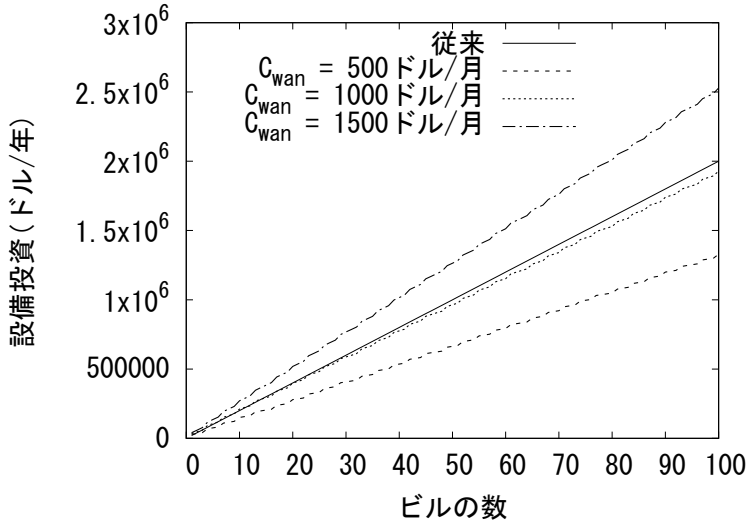


図 4.14: 設備投資の試算結果

費は 200,000 円 (2000 ドル) である．つまり現状の WAN サービスでは，パブリッククラウドを利用するほうが設備投資が増える．また，WAN サービスを提供可能な国が限られるため，利用できるパブリッククラウドのリージョンが限定される．パブリッククラウドを利用することで設備投資の削減効果を得るためには，WAN サービスの契約費の低価格化が進み，現状の 50%程度 (1000 ドル程度) になる必要がある．

図 4.15 では，通信により発生するコストはほぼ見えなくなっている．監視制御の通信は，整数値や小数値をやりとりするだけであるため，発生するトラヒックは小さい．ハートビートも同様であり，トラヒックは小さく，結果として，通信により発生するコストは，VM の使用料や WAN の契約費と比べて小さくなった．トラヒック量を減らす方法として，1) 複数の監視点データをまとめて計測する，2) ハートビートの送信頻度を下げる，などが考えられる．これらの方法を適用することで，通信により発生するコストを下げることはできるが，総設備投資の削減の観点からは効果が薄いと言える．

本節では，ビル管理システムを構築するための設備投資のみを試算し，比較した．設備投資以外では，システムの保守費が考えられる．ビル内にビル管理システムを構築する場合は，物理的なコントローラが地理的に分散するビル群に設置

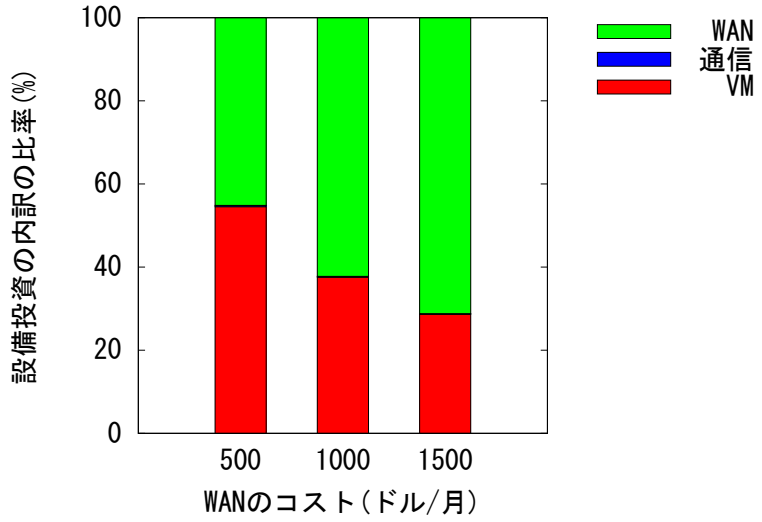


図 4.15: 設備投資の内訳の比率

される．そのため，コントローラを保守・修理するための人員を各地に配置する必要があり，そのための人件費が発生する．一方，パブリッククラウドを利用する場合は，コントローラはソフトウェア化され，ネットワーク経由で管理できる．したがって，パブリッククラウドを利用したほうが保守費を下げられると考えられる．

4.8　むすび

本章では，パブリッククラウドにおける監視制御アプリケーションの稼働率を向上するための冗長化手法を提案した．監視制御アプリケーションとして，フィードバック制御コントローラを想定した．提案手法では，複数のプライマリコントローラとバックアップコントローラを，異なるリージョンで実行させ，コントローラ間で生存確認のためのハートビートメッセージを交換する．ハートビートに，制御に関する情報や，制御用ネットワークの通信品質の情報を含めることで，異常発生時に，制御品質の低下を抑えられる新プライマリコントローラを選択できる．

また，パブリッククラウドの一種である Amazon EC2 を用いて提案手法を評価した．その結果，制御に関する情報や通信品質を共有することで，制御品質の低

下を抑えつつフェイルオーバーできることを確認した．

今後は，プライマリコントローラの選択方法を改善する．現状の提案手法のプライマリコントローラの選択方法は決定論的ではない．つまり，複数のバックアップコントローラの制御用ネットワークの通信品質が同等であった場合，ハートビートを交換するタイミングや，プライマリコントローラの選択処理を実行するタイミングによっては，各コントローラが異なるプライマリコントローラを選択してしまう可能性が存在する．この問題を解決するためには，Paxos [58] や Raft [72] などの分散合意形成アルゴリズムを適用してプライマリコントローラを決定する必要があると考える．また，待ち行列理論などに基づき想定環境をモデル化し，フェイルオーバーが制御品質に与える影響の理論値を得るための検討を行う．これにより，パブリッククラウドやインターネットというブラックボックスが制御品質に与える影響を分析しやすくなると考える．

第5章　結論

5.1　本論文のまとめ

本論文では，監視制御システムにクラウドコンピューティングを適用した場合に，監視制御システムの高い安定性を保証するための方法について議論した．

まず第1章では，監視制御システムの概要を述べた．また，クラウドコンピューティングの実現形態としてプライベートクラウドとパブリッククラウドが存在することと，それぞれの特徴を述べた．そして，監視制御システムをクラウドコンピューティング上で構築する際の課題として，安定した性能の保証と稼働率の向上の2つがあることを述べた．

第2章では，プライベートクラウド上で監視制御システムを構築することを想定し，監視制御アプリケーションをVM上で動作させた場合の性能への影響を把握した．VMに割り当てるCPUリソース量や，単一の物理マシン上で複数のVMを同時に実行することが監視制御アプリケーションの性能に与える影響を調べるための評価を実施した．また，物理マシンのCPUコアの一部を仮想化ソフトウェアに占有させることが性能へ与える影響や，CPUスケジューリングのアルゴリズムが性能へ与える影響を調べるための評価を実施した．評価結果から，VM間のCPU競合や，仮想化ソフトウェアのCPU不足が，監視制御アプリケーションの性能を低下させる主要因であることを確認した．

第3章では，第2章で得られた知見に基づき，VM上で動作する監視制御アプリケーションの性能を推定する手法を提案した．提案手法は，物理マシンやVMの挙動をモデル化し，そのモデルに基づくシミュレーションにより，複数の監視制御アプリケーション／VMを単一の物理マシン上で実行した場合の性能を推定する．モデルにおける物理マシンの性能に依存するパラメータは，実機をモニタリング

した結果から値を求めることで，シミュレーションの精度を向上する．複数のVMによるCPU競合を考慮して性能を推定する点と，仮想化ソフトウェアのCPU不足を考慮して性能を推定する点が，提案手法の特徴である．提案手法の有効性を示すために，実機を用いた性能評価と，提案手法による推定結果を比較し，提案手法の推定精度を検証した．その結果，監視制御アプリケーションの性能が著しく低下する状況を推定できることを確認した．

第4章では，監視制御アプリケーションをパブリッククラウド上で実行する環境を想定し，その稼働率を向上させる手法を提案した．提案手法では，各クラウド上で実行される監視制御アプリケーションが，定期的に生存確認用のメッセージを交換することで，互いに生存確認を行う．また，単に生存確認を行うだけでなく，それぞれが実施している監視制御処理の情報や，監視制御アプリケーションと監視制御対象との間のネットワークの品質を共有することで，障害発生時の制御品質の低下を抑える．パブリッククラウドを使用した評価を行い，提案手法が制御品質の低下を抑えられるかを検証した．その結果，提案手法は，監視制御処理の情報やネットワーク品質を共有しない場合と比べて，制御品質の低下を抑えられることを確認した．

監視制御システムの低コスト化と柔軟性向上のため，監視制御システムをクラウド上で構築するための研究が注目されている．しかし，既存の研究の多くは，少数の監視制御アプリケーションの性能を仮想化環境において評価したものであり，多数の監視制御アプリケーションを同時に実行する状況や，「監視制御アプリケーションを追加する」という運用フローを考慮していない．これに対して，本論文で提案した手法は，多数の監視制御アプリケーションを同時に実行する状況における性能を推定し，監視制御アプリケーションを追加できるかを判断できる．また，通信の品質を考慮した冗長化手法により，パブリッククラウド上でも高い稼働率と制御品質を実現する．本研究の成果は，クラウドコンピューティングを利用した監視制御システムの実現に向けて，大きな進展をもたらすものである．

5.2 検討課題

クラウドコンピューティングを利用した監視制御システムを真に実現するためには，各章のむすびで述べた課題に加えて，WAN サービスの高信頼化と低コスト化が今後の検討課題となる．

2.2.2 項で述べたように，ビルの火災報知器の状態を計測する場合は，計測時刻誤差を 5 ミリ秒以下に抑えたい．これを実現するためには，通信帯域を保証できる WAN サービスが必要である．ビル管理システムや電力系統網監視システムに使用できる品質の WAN の実現を目指し，Internet Engineering Task Force (IETF) にて Deterministic Networking (DetNet) [25, 31] という標準規格の策定が進んでいる．DetNet は，PCE [23] や MPLS [78] などの経路制御および帯域予約の手法を活用して，信頼性や通信遅延を制御する．このようなオープンな標準規格が策定・実用化されれば，WAN サービスの低価格化が進むと考える．ただし，DetNet において，どのような経路を選択し，どれだけの帯域を予約するかは，規格の範囲外である．したがって，監視制御アプリケーションの要件を考慮して，適切な経路を選択し，適切な帯域を予約するアルゴリズムが必要となる．

謝辞

本研究を推進するにあたり，懇切なる御指導と惜しみない御助言を頂きました大阪大学大学院情報科学研究科マルチメディア工学専攻 原 隆浩教授に謹んで御礼申し上げます．

本論文をまとめるにあたり，大変有益な御指導と御助言を多数賜りました大阪大学大学院情報科学研究科マルチメディア工学専攻 下條真司教授，松下康之教授に心より感謝申し上げます．

講義，学生生活を通じて，学問に取り組む姿勢をご教授頂きました大阪大学大学院情報科学研究科マルチメディア工学専攻 藤原 融教授，鬼塚 真教授，に厚く感謝申し上げます．

本研究において，多大なる御助言，御協力，御支援を頂きました大阪大学サイバーメディアセンター 義久智樹准教授，大阪大学大学院情報科学研究科マルチメディア工学専攻 前川卓也准教授，天方大地助教に深謝致します．

本研究において，多大なる御助言，御協力，御支援を頂きました大阪大学大学院情報科学研究科マルチメディア工学専攻 横山正浩氏，株式会社東芝 青木 恒氏、村井信哉氏、前川智則氏、伊藤俊夫氏に深く御礼申し上げます．

本研究を進めるにあたり，多くの御討論や御助言を頂きました大阪大学大学院情報科学研究科マルチメディア工学専攻 原研究室の諸氏に心より感謝申し上げます．

最後に，私のこれまでの人生，そして研究生活を送る上で，暖かい支援と理解を頂いた両親，妻，子供たち，友人に，心から感謝致します．

参考文献

[1] Aida, K., Abdul-Rahman, O., Sakane, E., and Motoyama, K.: Evaluation on the Performance Fluctuation of Hadoop Jobs in the Cloud, in *Proc. of IEEE International Conference on Computational Science and Engineering*, pp. 159–166 (2013).

[2] Akoush, S., Sohan, R., Rice, A., Moore, A., and Hopper, A.: Predicting the Performance of Virtual Machine Migration, in *Proc. of IEEE International Symposium on Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS 2010)*, pp. 37–46 (2010).

[3] Al-Omari, H., Wolff, F., Papachristou, C., and McIntyre, D.: An Improved Algorithm to Smooth Delay Jitter in Cyber-Physical Systems, in *Proc. of International Conference on Scalable Computing and Embedded Computing*, pp. 81–86 (2009).

[4] Apparao, P., Iyer, R., Zhang, X., Newell, D., and Adelmeyer, T.: Characterization & Analysis of a Server Consolidation Benchmark, in *Proc. of ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE 2008)*, pp. 21–30 (2008).

[5] Apparao, P., Makineni, S., and Newell, D.: Characterization of network processing overheads in Xen, in *Proc. of International Workshop on Virtualization Technology in Distributed Computing*, pp. 2–2 (2006).

[6] ASHRAE: Annex J to ANSI/ASHRAE Standard 135-1995 - BACnet/IP (1999).

[7] ASHRAE: Addendum C to ANSI/ASHRAE Standard 135-2004 - BACnet/WS (2006).

[8] Babu, S., Hareesh, M., Martin, J., Cherian, S., and Sastri, Y.: System Performance Evaluation of Para Virtualization, Container Virtualization, and Full Virtualization Using Xen, OpenVZ, and XenServer, in *Proc. of International Conference on Advances in Computing and Communications (ICACC 2014)*, pp. 247–250 (2014).

[9] Bai, J., Xiao, H., Yang, X., and Zhang, G.: Study on integration technologies of building automation systems based on web services, in *Proc. of ISECS International Colloquium on Computing, Communication, Control, and Management*, pp. 262–266 (2009).

[10] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A.: Xen and the Art of Virtualization, *SIGOPS Oper. Syst. Rev.*, Vol. 37, No. 5, pp. 164–177 (2003).

[11] Bin, J., Girbal, S., Prez, D. G., Grasset, A., and Merigot, A.: Studying corunning avionic real-time applications on multi-core COTS architectures, in *Proc. of Embedded Real Time Software and Systems (ERTS 2014)*, pp. 1–10 (2014).

[12] Breivold, H., Jansen, A., Sandstrom, K., and Crnkovic, I.: Virtualize for Architecture Sustainability in Industrial Automation, in *Proc. of IEEE International Conference on Computational Science and Engineering*, pp. 409–415 (2013).

[13] Breivold, H., and Sandstrom, K.: Virtualize for test environment in industrial automation, in *Proc. of IEEE International Conference on Emerging Technology and Factory Automation (ETFA 2014)*, pp. 1–8 (2014).

[14] Camargos, F. L., Girard, G., and des Ligneris, B.: Virtualization of Linux servers, in *Proc. of Linux Symposium 2008*, pp. 63–76 (2008).

[15] Chaiboonruang, P., Pora, W., Ochiai, H., and Esaki, H.: Small buildings energy management system based on IEEE1888 standard with data compression, in *Proc. of International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON 2014)*, pp. 1–6 (2014).

[16] Che, J., Yu, Y., Shi, C., and Lin, W.: A Synthetical Performance Evaluation of OpenVZ, Xen and KVM, in *Proc. of IEEE Asia-Pacific Services Computing Conference (APSCC 2010)*, pp. 587–594 (2010).

[17] Debbag, Y., and Yilmaz, E.: Internet based monitoring and control of a wind turbine via PLC, in *Proc. of International Istanbul Smart Grid Congress and Fair (ICSG 2015)*, pp. 1–5 (2015).

[18] Deshane, T., Shepherd, Z., Matthews, J. N., Ben-Yehuda, M., Shah, A., and Rao, B.: Quantitative Comparison of Xen and KVM, in *Proc. of Xen Summit 2008*, pp. 1–3 (2008).

[19] Dhingra, M., Lakshmi, J., Nandy, S., Bhattacharyya, C., and Gopinath, K.: Elastic Resources Framework in IaaS, Preserving Performance SLAs, in *Proc. of IEEE International Conference on Cloud Computing (CLOUD 2013)*, pp. 430–437 (2013).

[20] Dritsas, L., Tsoulkas, V., Pantelous, A., and Tzes, A.: Robust Performance Issues in Tracking Control over Networks, in *Proc. of International Conference on Intelligent Systems, Modelling and Simulation (ISMS 2010)*, pp. 264–269 (2010).

[21] Dua, R., Raja, A., and Kakadia, D.: Virtualization vs Containerization to Support PaaS, in *Proc. of IEEE International Conference on Cloud Engineering (IC2E 2014)*, pp. 610–614 (2014).

[22] Dunlap, G.: Scheduler development update, in *Proc. of Xen Summit Asia 2009*, pp. 1–3 (2009).

[23] Farrel, A., Vasseur, J.-P., and Ash, J.: A Path Computation Element (PCE)-Based Architecture, IETF RFC 4655 (2006).

[24] Felter, W., Ferreira, A., Rajamony, R., and Rubio, J.: An updated performance comparison of virtual machines and Linux containers, in *Proc. of IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS 2015)*, pp. 171–172 (2015).

[25] Finn, N., and Teener, M. J.: Deterministic Networking Architecture, IETF Internet-Draft (2015).

[26] Fraser, K., Hand, S., Neugebauer, R., Pratt, I., Warfield, A., Warfield, A., Williamson, M., and Williamson, M.: Reconstructing I/O, Technical Report UCAM-CL-TR-596 (2004).

[27] Gilani, S. S., Jungbluth, F., Flatt, H., Wendt, V., and Jasperneite, J.: Alternative controls for soft real-time industrial control services in case of broken cloud links, in *Proc. of IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2016)*, pp. 1–4 (2016).

[28] Givehchi, O., Imtiaz, J., Trsek, H., and Jasperneite, J.: Control-as-a-service from the cloud: A case study for using virtualized PLCs, in *Proc. of IEEE Workshop on Factory Communication Systems (WFCS 2014)*, pp. 1–4 (2014).

[29] Givehchi, O., Trsek, H., and Jasperneite, J.: Cloud computing for industrial automation systems - A comprehensive overview, in *Proc. of IEEE Conference on Emerging Technologies Factory Automation (ETFA 2013)*, pp. 1–4 (2013).

[30] Goldschmidt, T., Murugaiah, M. K., Sonntag, C., Schlich, B., Biallas, S., and Weber, P.: Cloud-Based Control: A Multi-tenant, Horizontally Scalable Soft-PLC, in *Proc. of IEEE International Conference on Cloud Computing (CLOUD 2015)*, pp. 909–916 (2015).

[31] Grossman, E., Gunther, C., Thubert, P., Wetterwald, P., Raymond, J., Korhonen, J., Kaneko, Y., Das, S., and Zha, Y.: Deterministic Networking Use Cases, IETF Internet-Draft (2015).

[32] Gupta, R., and Chow, M.-Y.: Networked Control System: Overview and Research Trends, *IEEE Transactions on Industrial Electronics*, Vol. 57, No. 7, pp. 2527–2535 (2010).

[33] Han, R., Guo, L., Ghanem, M., and Guo, Y.: Lightweight Resource Scaling for Cloud Applications, in *Proc. of IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2012)*, pp. 644–651 (2012).

[34] Hgglund, T., and strm, K. J.: Revisting the Ziegler-Nichols Tuning Rules for PI Control, *Asian Journal of Control*, Vol. 4, No. 4, pp. 364–380 (2002).

[35] He, S., Guo, L., Ghanem, M., and Guo, Y.: Improving Resource Utilisation in the Cloud Environment Using Multivariate Probabilistic Models, in *Proc. of IEEE International Conference on Cloud Computing (CLOUD 2012)*, pp. 574–581 (2012).

[36] Hegazy, T., and Hefeeda, M.: Industrial Automation as a Cloud Service, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 26, No. 10, pp. 2750–2763 (2015).

[37] Hnarakis, R., and Nico, P.: In Perfect Xen, A Performance Study of the Emerging Xen Scheduler, Master's thesis, California Polytechnic State University (2013).

[38] Hong, I., Byun, J., and Park, S.: Cloud Computing-Based Building Energy Management System with ZigBee Sensor Network, in *Proc. of International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS 2012)*, pp. 547–551 (2012).

[39] Huang, Q., and Lee, P. P.: An Experimental Study of Cascading Performance Interference in a Virtualized Environment, *SIGMETRICS Perform. Eval. Rev.*, Vol. 40, No. 4, pp. 43–52 (2013).

[40] IEEE: IEEE 1888-2011 Standard for Ubiquitous Green Community Control Network Protocol.

[41] Imai, R., and Kubo, R.: Cloud-based remote motion control over FTTH networks for home robotics, in *Proc. of IEEE Global Conference on Consumer Electronics (GCCE 2014)*, pp. 565–566 (2014).

[42] Iosup, A., Yigitbasi, N., and Epema, D.: On the Performance Variability of Production Cloud Services, in *Proc. of IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2011)*, pp. 104–113 (2011).

[43] 伊藤俊夫, 米良恵介, 金子雄, 松澤茂雄：通信エンドの負荷ピークを低減するためのビル設備情報収集スケジュール作成方法, 電子情報通信学会技術研究報告 情報ネットワーク, Vol. 111, No. 197, pp. 77–82 (2011).

[44] Jung, M., Weidinger, J., Kastner, W., and Olivieri, A.: Building Automation and Smart Cities: An Integration Approach Based on a Service-Oriented Architecture, in *Proc. of International Conference on Advanced Information Networking and Applications Workshops (WAINA 2013)*, pp. 1361–1367 (2013).

[45] Kaneko, Y., and Ito, T.: A Reliable Cloud-Based Feedback Control System, in *Proc. of IEEE International Conference on Cloud Computing (CLOUD 2016)*, pp. 880–883 (2016).

[46] Kaneko, Y., Ito, T., and Hara, T.: A measurement study on virtualization overhead for applications of industrial automation systems, in *Proc. of IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2016)*, pp. 1–8 (2016).

[47] Kaneko, Y., Ito, T., and Hara, T.: Performance Estimation Method for Virtualized Building Facilities Monitoring Applications, in *Proc. of IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2017)* (presented, 2017).

[48] Kaneko, Y., Ito, T., Ito, M., and Kawazoe, H.: Virtual Machine Scaling Method Considering Performance Fluctuation of Public Cloud, in *Proc. of IEEE International Conference on Cloud Computing (CLOUD 2017)*, pp. 782–785 (2017).

[49] 金子雄, 伊藤俊夫, 原隆浩：仮想化が機器状態監視の定刻性に与える影響の一評価, マルチメディア，分散協調とモバイルシンポジウム (DICOMO 2016) 論文集, pp. 1566–1577 (2016).

[50] 金子雄, 伊藤俊夫, 原隆浩：ビル設備機器の状態監視における仮想化技術の適用可能性の検討, 情報処理学会論文誌 ネットワークサービスと分散処理特集, Vol. 58, No. 2, pp. 461–470 (2017).

[51] 金子雄, 伊藤俊夫, 原隆浩：仮想化された機器監視アプリケーションの計測時刻の誤差推定手法, 情報処理学会 マルチメディア通信と分散処理研究会 研究報告, pp. 1–8 (2017).

[52] 金子雄, 伊藤俊夫, 原隆浩：仮想化を利用した設備機器の状態監視における計測時刻誤差の推定手法, 情報処理学会論文誌 ネットワークサービスと分散処理特集 (採録決定，2018 年予定).

[53] 金子雄, 前川智則, 山田孝裕, 松澤茂雄 : 地域エネルギー管理システムの通信ソフトウェアの開発と運用—実証実験により得られた知見—, 情報処理学会 デジタルプラクティス, Vol. 5, No. 3, pp. 213–218 (2014).

[54] Kim, J.-S., Lee, S.-H., and Jin, H.-W.: Fieldbus virtualization for Integrated Modular Avionics, in *Proc. of IEEE Conference on Emerging Technologies Factory Automation (ETFA 2011)*, pp. 1–4 (2011).

[55] Kivity, A., Kamay, Y., Laor, D., Lublin, U., and Liguori, A.: kvm: the Linux Virtual Machine Monitor, in *Proc. of Linux Symposium 2007*, pp. 225–230 (2007).

[56] Koh, Y., Knauerhase, R., Brett, P., Bowman, M., Wen, Z., and Pu, C.: An Analysis of Performance Interference Effects in Virtual Environments, in *Proc. of IEEE International Symposium on Performance Analysis of Systems Software (ISPASS 2007)*, pp. 200–209 (2007).

[57] Komiya, H., Takizawa, H., Matsukawa, H., and Kozakai, K.: FFCS compact control station in centum CS3000 R3, *Yokogawa Technical Report*, No. 38, pp. 5–8 (2004).

[58] Lamport, L.: ACM SIGACT News Distributed Computing Column, *SIGACT News*, Vol. 32, No. 4, pp. 51–58 (2001).

[59] Li, H., Chow, M.-Y., and Sun, Z.: Speed control of a networked DC motor system with time delays and packet losses, in *Proc. of IEEE Annual Conference of Industrial Electronics (IECON 2008)*, pp. 2941–2946 (2008).

[60] Li, W., and Kanso, A.: Comparing Containers versus Virtual Machines for Achieving High Availability, in *Proc. of IEEE International Conference on Cloud Engineering (IC2E 2015)*, pp. 353–358 (2015).

[61] Li, Z., OBrien, L., Ranjan, R., and Zhang, M.: Early Observations on Performance of Google Compute Engine for Scientific Computing, in *Proc. of IEEE*

International Conference on Cloud Computing Technology and Science, pp. 1–8 (2013).

[62] Mahmud, N., Sandstrom, K., and Vulgarakis, A.: Evaluating industrial applicability of virtualization on a distributed multicore platform, in *Proc. of IEEE International Conference on Emerging Technology and Factory Automation (ETFA 2014)*, pp. 1–8 (2014).

[63] Mao, M., and Humphrey, M.: A Performance Study on the VM Startup Time in the Cloud, in *Proc. of IEEE International Conference on Cloud Computing (CLOUD 2012)*, pp. 423–430 (2012).

[64] Mei, Y., Liu, L., Pu, X., and Sivathanu, S.: Performance Measurements and Analysis of Network I/O Applications in Virtualized Cloud, in *Proc. of IEEE International Conference on Cloud Computing (CLOUD 2010)*, pp. 59–66 (2010).

[65] Meng, X., Isci, C., Kephart, J., Zhang, L., Bouillet, E., and Pendarakis, D.: Efficient Resource Provisioning in Compute Clouds via VM Multiplexing, in *Proc. of International Conference on Autonomic Computing (ICAC 2010)*, pp. 11–20 (2010).

[66] Nadas, S.: Virtual Router Redundancy Protocol (VRRP) Version 3 for IPv4 and IPv6, IETF RFC 5798 (2010).

[67] Nathuji, R., Kansal, A., and Ghaffarkhah, A.: Q-clouds: Managing Performance Interference Effects for QoS-aware Clouds, in *Proc. of European Conference on Computer Systems (EuroSys 2010)*, pp. 237–250 (2010).

[68] Nguyen, H., Shen, Z., Gu, X., Subbiah, S., and Wilkes, J.: AGILE: Elastic Distributed Resource Scaling for Infrastructure-as-a-Service, in *Proc. of International Conference on Autonomic Computing (ICAC 2013)*, pp. 69–82 (2013).

[69] 西澤茂隆, 中村匡秀, 井垣宏, 松本健一, 三浦健次郎：ビル管理システムにおけるサービス指向アーキテクチャの適用：異種システムの連携と安全性に関する考察, 電子情報通信学会技術研究報告, Vol. 107, No. 261, pp. 3–8 (2007).

[70] Novaković, D., Vasić, N., Novaković, S., Kostić, D., and Bianchini, R.: DeepDive: Transparently Identifying and Managing Performance Interference in Virtualized Environments, in *Proc. of USENIX Conference on Annual Technical Conference (USENIX ATC 2013)*, pp. 219–230 (2013).

[71] OASIS: oBIX 1.0 - Committee Specification 01 (2006).

[72] Ongaro, D., and Ousterhout, J.: In Search of an Understandable Consensus Algorithm, in *Proc. of USENIX Annual Technical Conference (USENIX ATC 2014)*, pp. 305–319 (2014).

[73] Ostermann, S., Iosup, A., Yigitbasi, N., Prodan, R., Fahringer, T., and Epema, D.: A Performance Analysis of EC2 Cloud Computing Services for Scientific Computing, in *Proc. of International Conference on Cloud Computing (CloudComp 2009)*, pp. 115–131 (2009).

[74] Parker, P., and Chadwick, S.: Scada approaches to Remote Condition Monitoring, in *Proc. of IET Conference on Railway Condition Monitoring and Non-Destructive Testing (RCM 2011)*, pp. 1–6 (2011).

[75] Patnaik, D., Krishnakumar, A. S., Krishnan, P., Singh, N., and Yajnik, S.: Performance Implications of Hosting Enterprise Telephony Applications on Virtualized Multi-core Platforms, in *Proc. of International Conference on Principles, Systems and Applications of IP Telecommunications*, pp. 1–11 (2009).

[76] Pu, X., Liu, L., Mei, Y., Sivathanu, S., Koh, Y., Pu, C., and Cao, Y.: Who Is Your Neighbor: Net I/O Performance Interference in Virtualized Clouds, *IEEE Transactions on Services Computing*, Vol. 6, No. 3, pp. 314–329 (2013).

[77] Qin, B., and Yan, D.: Remote SCADA System Based on 3G VPN Services for Secondary Pressurization Pump Station, in *Proc. of International Conference on Intelligent System Design and Engineering Application (ISDEA 2010)*, Vol. 2, pp. 132–135 (2010).

[78] Rosen, E., Viswanathan, A., and Callon, R.: Multiprotocol Label Switching Architecture, IETF RFC 3031 (2001).

[79] Schachinger, D., Stampfel, C., and Kastner, W.: Interoperable integration of building automation systems using RESTful BACnet Web services, in *Proc. of IEEE International Conference on Industrial Electronics Society (IECON 2015)*, pp. 3899–3904 (2015).

[80] Shahzad, A., Musa, S., Aborujilah, A., and Irfan, M.: A Performance Approach: SCADA System Implementation within Cloud Computing Environment, in *Proc. of International Conference on Advanced Computer Science Applications and Technologies (ACSAT 2013)*, pp. 274–277 (2013).

[81] Singh, S., Chary, V., and Rahman, P.: Dual Redundant Profibus Network Architecture in hot standby fault tolerant control systems, in *Proc. of International Conference on Advances in Engineering and Technology Research (ICAETR 2014)*, pp. 1–5 (2014).

[82] Smith, O. J.: A controller to overcome dead time, *ISA J.*, Vol. 6, No. 2, pp. 28–33 (1959).

[83] Somani, G., and Chaudhary, S.: Application Performance Isolation in Virtualization, in *Proc. of IEEE International Conference on Cloud Computing (CLOUD 2009)*, pp. 41–48 (2009).

[84] Soriga, S., and Barbulescu, M.: A comparison of the performance and scalability of Xen and KVM hypervisors, in *Proc. of RoEduNet International Conference on Networking in Education and Research*, pp. 1–6 (2013).

[85] Suresh, K., Kirubashankar, R., and Krishnamurthy, K.: Research of Internet based supervisory control and information system, in *Proc. of International Conference on Recent Trends in Information Technology (ICRTIT 2011)*, pp. 1180–1185 (2011).

[86] Tafa, I., Beqiri, E., Paci, H., Kajo, E., and Xhuvani, A.: The Evaluation of Transfer Time, CPU Consumption and Memory Utilization in XEN-PV, XEN-HVM, OpenVZ, KVM-FV and KVM-PV Hypervisors Using FTP and HTTP Approaches, in *Proc. of International Conference on Intelligent Networking and Collaborative Systems (INCoS 2011)*, pp. 502–507 (2011).

[87] Tomás, L., and Tordsson, J.: Improving Cloud Infrastructure Utilization Through Overbooking, in *Proc. of ACM Cloud and Autonomic Computing Conference (CAC 2013)*, pp. 1–10 (2013).

[88] Venkataramanan, V.: *Optimization of CPU Scheduling in Virtual Machine Environments*, PhD thesis, Ottawa University (2015).

[89] Wang, J., Varman, P., and Xie, C.: Avoiding performance fluctuation in cloud storage, in *Proc. of International Conference on High Performance Computing*, pp. 1–9 (2010).

[90] Wang, Z., Zhu, X., Padala, P., and Singhal, S.: Capacity and Performance Overhead in Dynamic Resource Allocation to Virtual Containers, in *Proc. of IFIP/IEEE International Symposium on Integrated Network Management (IM 2007)*, pp. 149–158 (2007).

[91] Wu, H., Lou, L., Chen, C.-C., Hirche, S., and Kuhnlenz, K.: Cloud-Based Networked Visual Servo Control, *IEEE Transactions on Industrial Electronics*, Vol. 60, No. 2, pp. 554–566 (2013).

[92] Xi, S., Wilson, J., Lu, C., and Gill, C.: RT-Xen: Towards real-time hypervisor scheduling in Xen, in *Proc. of International Conference on Embedded Software (EMSOFT 2011)*, pp. 39–48 (2011).

[93] Xi, S., Xu, M., Lu, C., Phan, L., Gill, C., Sokolsky, O., and Lee, I.: Real-time multi-core virtual machine scheduling in Xen, in *Proc. of International Conference on Embedded Software (EMSOFT 2014)*, pp. 1–10 (2014).

[94] Xia, Y.: Cloud control systems, *IEEE/CAA Journal of Automatica Sinica*, Vol. 2, No. 2, pp. 134–142 (2015).

[95] Zhao, H., Zheng, Q., Zhang, W., Chen, Y., and Huang, Y.: Virtual machine placement based on the VM performance models in cloud, in *Proc. of IEEE International Performance Computing and Communications Conference (IPCCC 2015)*, pp. 1–8 (2015).

[96] Zheng, W., Bianchini, R., Janakiraman, G. J., Santos, J. R., and Turner, Y.: JustRunIt: Experiment-based Management of Virtualized Data Centers, in *Proc. of USENIX Annual Technical Conference (USENIX ATC 2009)*, pp. 1–16 (2009).

[97] Zhou, W., Yang, S., Fang, J., Niu, X., and Song, H.: VMCTune: A Load Balancing Scheme for Virtual Machine Cluster Using Dynamic Resource Allocation, in *Proc. of International Conference on Grid and Cooperative Computing (GCC 2010)*, pp. 81–86 (2010).