令和 5 年度 修士論文

キャッシュ利用に着目した
複数仮想計算機へのメモリ割り当て量の最適化

龍谷大学大学院 理工学研究科 修士課程 情報メディア学専攻

学籍番号 ： T22M057
氏名 ： 青木 雄佐

指導教員 ： 三好 力 教授，芝 公仁 助教

内容梗概

仮想化技術を利用して，複数の仮想計算機を 1 台の物理計算機に集約し運用することが多く行われている．このように，複数の仮想計算機で物理計算機を共有することにより，計算機資源の利用効率の向上が期待される．仮想計算機にはそれぞれ計算機資源が割り当てられ，仮想計算機ごとに管理が行われる．このとき，それぞれの仮想計算機に適切な量のメモリを割り当てることは難しく，十分な効率化の効果を得られないことも多い．これは，システムが効率的に動作するのに必要なメモリ量を，システム構築時に正確に予測することが困難であるためである．また，システムの動作に応じて必要なメモリ量が動的に変化することも多い．本稿では，ディスクキャッシュが有効に機能するよう，仮想計算機へのメモリ割り当て量を設定する手法について述べる．本手法では，仮想計算機の動作を監視し，メモリ割り当て量とキャッシュヒット数の関係のモデルを作成する．また，このモデルを用いてキャッシュが有効に機能するメモリ割り当て量を算出し，各計算機へのメモリ割り当てを行う．本手法により，仮想計算機のキャッシュが有効に機能するようにメモリを配分することが可能になる．これによって，限られた物理メモリ資源を効果的に活用することができ，システム全体の効率化が実現される．

Abstract

Virtualization technology is often used to operate multiple virtual computers by consolidating them into a single physical computer. Sharing a physical computer with multiple virtual computers in this way is expected to improve the efficiency of computer resource use. Each virtual computer is allocated its own computing resources, and each virtual computer is managed individually. It is difficult to allocate an appropriate amount of memory to each virtual computer, and often the efficiency of the system is not sufficiently improved. This is because it is difficult to accurately predict the amount of memory required for efficient system operation at the time of system construction. In addition, the amount of required memory often changes dynamically depending on the system operation. This paper describes a method for setting the amount of memory allocated to virtual computers so that the disk cache functions effectively. This method monitors the operation of virtual computers and creates a model of the relationship between the amount of memory allocated and the number of cache hits. This model is used to calculate the amount of memory allocated for the cache to function effectively. This model is used to calculate the amount of memory allocated for the cache to function effectively and allocate memory to each computer. This method makes it possible to allocate memory so that the caches of virtual computers function effectively. This enables effective use of limited physical memory resources and improves the efficiency of the entire system.

目次

1　はじめに

近年，仮想化技術が広く普及し，様々な分野で利用されるようになってきている．仮想計算機を使用し，1 台の物理計算機上に複数の計算機を集約することによって，計算機資源の利用効率を向上させることができる．また，仮想計算機間で資源が分離され独立性が高くなるため，システムの管理が容易になり，また，運用の柔軟性の向上も期待できる．

しかし，仮想計算機ごとに資源管理が行われるため，限られた計算機資源を複数の仮想計算機で共有する場合には，資源の配分を適切に行う必要がある．特にメモリ資源に関しては，それぞれの仮想計算機に対して適切な量のメモリを割り当てることは難しいといった問題がある．これは，システムが効率的に動作するのに必要なメモリ量を，システム構築時に正確に予測することが困難であるためである．また，システムの動作に応じて必要なメモリ量が動的に変化することも多い．

本論文では，ディスクキャッシュが有効に機能するよう仮想計算機へのメモリ割り当て量を設定する手法と，本手法に基づきメモリ割り当てを行うシステムについて述べる．通常，プロセスにはそれが動作するために必要な量のメモリが割り当てられ，その上で余ったメモリがディスクキャッシュのために使用される．ディスクキャッシュは I/O 性能を高める重要なものであり，ディスクキャッシュの容量はシステムの性能に大きく影響する．提案手法では，仮想計算機の動作を監視し，メモリ割り当て量とキャッシュヒット数の関係のモデルを作成する．また，このモデルを用いてキャッシュが有効に機能するメモリ割り当て量を算出し，各計算機へのメモリ割り当てを行う．

本手法により，仮想計算機のキャッシュが有効に機能するようにメモリを配分することが可能になる．そのため，メモリのオーバーコミットの状態で仮想計算機を起動し，必要なメモリが少ない状態では不要分を解放し，メモリ負荷が高くなった場合には必要分を確保するといったことができる．これにより，限られた物理メモリ資源を効率的に活用することが可能になり，システム全体の性能と運用の柔軟性の向上が実現される．

以下，本論文では，2 章で関連研究について述べ，3 章で提案システムについて述べる．また，6 章で評価を行い，提案システムの有用性を示す．

2　関連研究

2.1　メモリ割り当て

ディスクアクセスのオーバヘッドが大きい仮想計算機では，ディスクキャッシュの効果が大きい．ディスクキャッシュの容量が I/O 性能に大きく影響する [1] ため，仮想計算機へのメモリ割り当て量の設定は重要な課題である．

仮想計算機へのメモリ割り当てに関しては，事前に必要なメモリ量を予測することが難しい，動的な負荷の変化があるなどの課題がある．これに対応するために，メモリのホットプラグやバルーニングの機能を使用し [2, 3]，仮想計算機のメモリ量を動的に変更することが行われる．

実際に割り当てるべきメモリ量を決めるためには，計算機上で動作するシステムのワーキングセットを考慮することが有効である．動作中のシステムのワーキングセットサイズを eBPF を使用して推測する手法が提案されている [4]．

また，仮想計算機上で動作しているシステムに対して，ワーキングセットサイズの推測を行う手法も提案されている [5, 6, 7]．これらは，仮想計算機上で動作しているシステムに対して，メモリ割り当て量を変更したり，キャッシュ領域を監視したりすることで，ワーキングセットサイズの推測を行っている．

提案システムでは，同様に，動作中の仮想計算機のメモリ割り当て量の変更と，動作の監視を行っている．また，このような処理を複数の仮想計算機に対して行い，それらに対する最適なメモリ割り当て量の探索を行っている．

2.2　使用アプリケーション

仮想計算機には QEMU を用いる．QEMU は汎用的なオープンソースのマシンエミュレータであり，仮想化ツールである．QEMU の最も一般的な使用方法はシステムエミュレーションで，ゲスト OS を実行するためのマシン全体の仮想モデル（CPU、メモリ、エミュレートされたデバイス）を提供する．このモードでは，CPU が完全にエミュレートされる場合もあれば，KVM，Xen，Hypervisor.Framework などのハイパーバイザと連携して，ゲストがホスト CPU 上で直接実行できるようにすることもできる．QEMU には，ディスクイメージの作成，変換，変更を可能にする qemu-img ディスクイメージユーティリティなど，スタンドアロンのコマンドラインユーティリティも多数用意されている．

QEMU のディスクイメージフォーマットとして qcow2 がある．qcow2 はオプションの AES 暗号化，zlib または zstd ベースの圧縮，複数の VM スナップショットのサポートに使用できる．qcow2 ディスクイメージをクローンし，差分だけを含むファイルを作成することもできる．複数ゲストを作るときは差分ファイルを複数作る．ベースとなるディスクイメージファイルを複数作る必要がないためストレージ容量への影響を抑える．

QEMU で実行している仮想計算機にメモリを割り当てる方法として，メモリホットプラグとメモリバルーニングがある．メモリホットプラグは，メモリ資源が即座に仮想計算機に提供されアプリケーションの要求に迅速に応答することができるため，特定の仮想計算機に対する直接的かつ細かいリソース管理が求められる場合に有効である．メモリバルーニングは，メモリホットプラグと比較しメモリを変化させる動作が容易なため処理内容が不明なときのメモリ割り当ての自動化に使われる．そのため，複数の仮想計算機間でメモリ資源のバランスを取ることに有効である．

ベイズ最適化は，最適化問題を解くための統計的手法の一つで，特に関数の評価が高コストな場面やノイズの多い場面での最適化タスクに適している．ベイズ最適化は，関数の評価を効率的に行いながら，評価されていない入力点での関数の出力を予測することを目的としている．

目的関数の事前分布を選択した後，実験の選択，実験の評価，事後分布の更新を繰り返す．事前分布は通常，ガウス過程としてモデル化されている．ガウス過程は関数のすべての可能な形を無限次元の他変量正規分布として捉えることができる強力なツールである．実験の選択は，予測された関数の不確実性と予測される関数の平均を考慮して，次に評価すべき入力点を選択する．実験の評価は，選択した入力点での目的関数の真の値を評価する．事後分布の更新は，新たなデータ点を利用して，ガウス過程を更新する．

入力点の選択は，最初は範囲内で均等に提案し，情報が集まると評価の高くなる値を推測する．

ベイズ最適化の主な利点は，計算コストが高い関数の最適化問題において，評価を最小限に抑えつつ関数の全体的な構造を効率的に学習する能力だ．ベイズ最適化は，ハイパーパラメータの調整や実験のデザインなど，さまざまなアプリケーションで利用されている．

BoTorch と Ax は，ベイズ最適化を実際の問題に適用する際のフレームワーク及びライブラリで，特に機械学習のハイパーパラメータ最適化などのタスクに使用される．両者は The Fundamental AI Research(旧 Facebook AI Research) によって開発され，連携して動作するよ

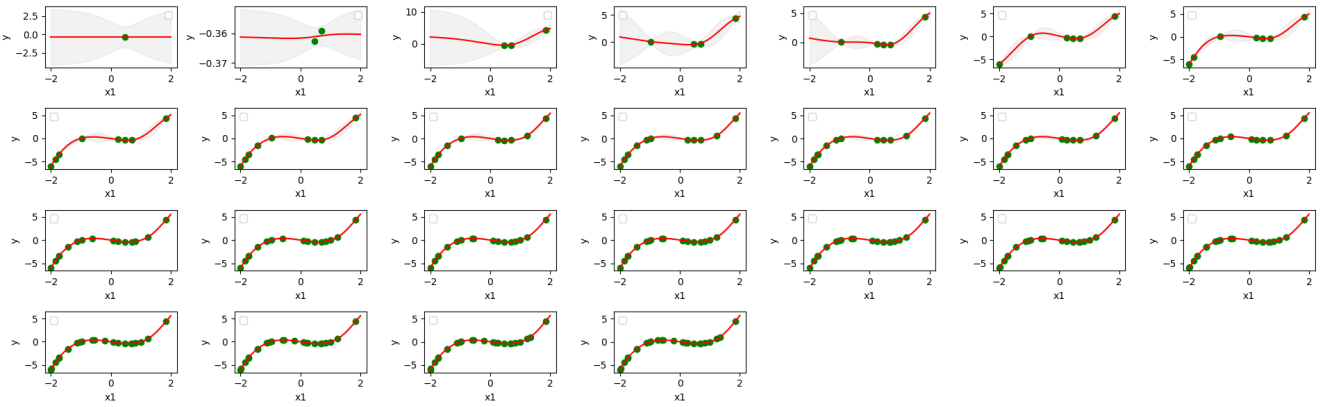

図 1　$x^3 - x$ の最小値調査

うに設計されている．

BoTorch は，モダンなベイズ最適化のアルゴリズムを構築するためのフレームワークだ．PyTorch をベースにしてあり，そのための計算の効率や拡張性に優れている．ガウス過程，多目的最適化，ベイズ最適化に必要な要素をサポートしている．

Ax は，実験や最適化タスクを管理するためのプラットフォームである．BoTorch をバックエンドとして使用しており，ベイズ最適化の実験を用意に設定・実行することができる．ハイパーパラメータの探索空間の定義，実験の結果の保存，最適化の進行状況の視覚化などの機能を持っている．Ax は，最初に初期試行を数回行い，その後最適化プロセスを行う．初期試行は，Sobol シーケンスで行う．Sobol シーケンスは底辺さな乱数生成手法の一つで，多次元の空間を均等にカバーするための手法として設計されている．それ以降は，GPEI と呼ばれる手法を用いる．GPEI は Gaussian Process Expected Improvement の略称で，ガウシアンプロセス回帰モデルを使用して目的関数をモデル化し，次に試すべき最も良い点を提案する．初期試行は使用する変数の数によって変わる．

図 1 は Ax で簡単な関数である式 1 の最小値を求める際の推測グラフである．灰色の部分は 95 % 信頼区間である．試行回数ごとに並んでおり，左上から右へ順番に並んでいる．この試行では最初の 5 回が初期試行であり，最小値となる x が x の範囲の最小値とはいえ 6 回目の時点で最小値を求めることができている．また，グラフも $x^3 - x$ と目測では違いがわからないほどの予測を立てている．

$$x^3 - x \, (-2.0 \leq x \leq 2.0) \tag{1}$$

図 2 は Ax で 2 変数関数である式 2 の最大値を求める際の推測グラフである．n は 2 で，t_1

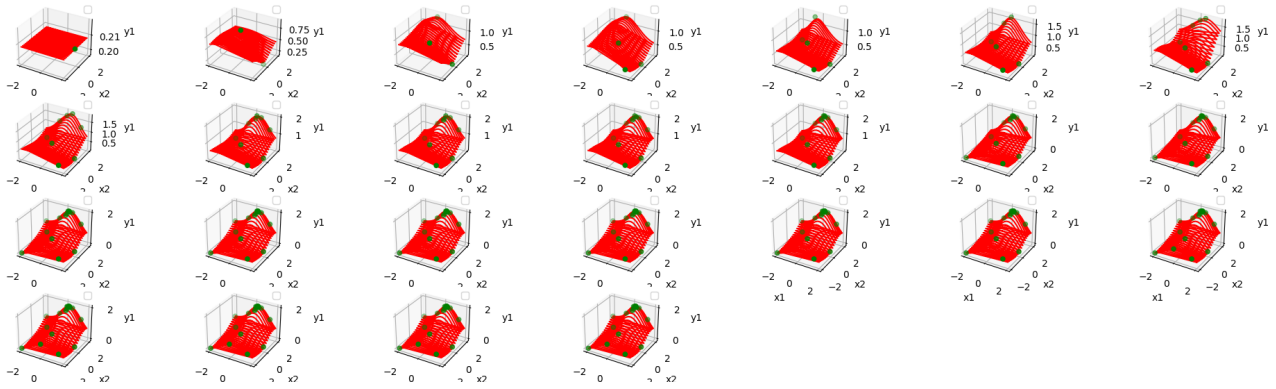

図 2　2 変数関数の最大値調査

は 0.5，t_2 は 1.5 としたためこの関数は，1 つ目の変数が 0.5 に近く，2 つ目の変数が 1.5 に近いほど高い値が出力される．この関数も初期試行が終わったあとの 6 回目の試行で最大値に近い変数を求められた．

$$\sum_{i=1}^{n} e^{-1\cdot(x_i-t_i)^2}\ (-2.0 \leq x_i \leq 2.0)(-2.0 \leq t_i \leq 2.0) \tag{2}$$

図 3 は Ax の学習個数を最新 5 個に制限し，試行回数が 8 回目以降の評価式を変更した際の関数推測グラフである．x が -2.0 から 2.0 の範囲での最小値を求める．7 回目までの試行を式 1 で評価し，8 回目以降の試行を式 3 で評価した．

$$x^4 - x^2\ (-2.0 \leq x \leq 2.0) \tag{3}$$

左上を試行回数 1 とし，左から右，上から下の順番で並んでいる．その試行回数で新しく Ax が提したパラメータを青い点，次の試行で最新 5 個のデータから外れるパラメータを黄色い点，それ以外の使用したデータを緑の点で表す．式 1 での試行は，6 回目で最小値を求める x を導いた．式 3 は x が約 0.71 または約-0.71 のときである．17 回目と 23 回目は x が 0.71 または -0.71 から 0.10 以内の値を提案した．

Linux カーネルの関数を監視するために eBPF を用いる．eBPF はパケットフィルタリング用に設計され，現在はセキュリティ，ネットワーキング，パフォーマンスモニタリングなど幅広いシステムレベルのタスクに応用されている．eBPF はカーネルのソースコードを変更することや，カーネルモジュールを組み込むことなくカーネルの機能を拡張することができる．そのため，バグのあるカーネルモジュールを組み込みクラッシュしてしまう，といったことを防ぐことができる．eBPF はカーネルモジュールで実行を希望するプログラムの一部

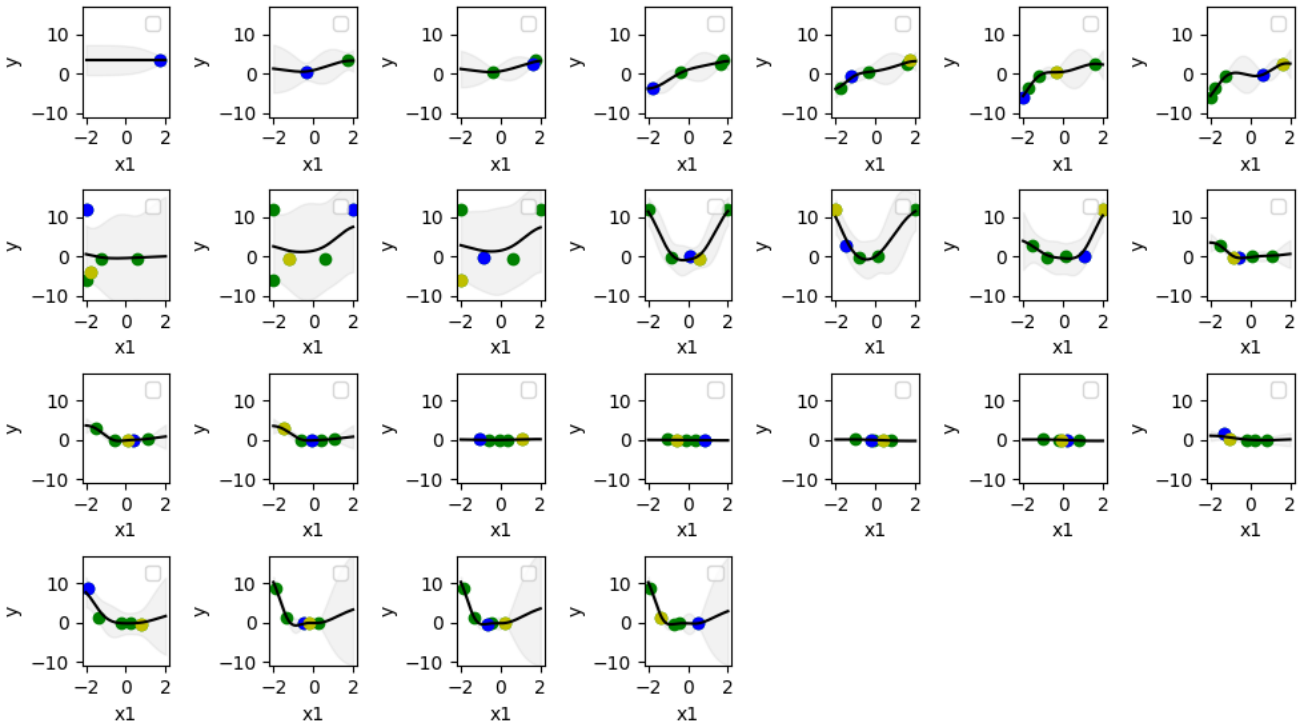

図 3　学習個数，評価関数を変化させた最小値調査

を実行可能にする．そのため，バグの含むカーネルモジュールを組み込みカーネルがクラッシュする，といったことを防ぐ．

BCC は，ユーザが eBPF プログラムを組み込んだ Python プログラムを書くことを可能にするフレームワークである．このフレームワークは，主にアプリケーションやシステムのプロファイリング・トレーシングを含むユースケースを対象としており，eBPF プログラムが統計情報の収集やイベントの生成に使用される．Python プログラムを実行することで eBPF のバイトコードが生成され，カーネルに読み込まれる．本論文では BCC で eBPF を使用し，キャッシュヒット数とキャッシュミス数を取得する．

3　システムの構成

提案システムは，図 4 に示すような単一のホスト計算機上で複数のゲスト計算機が動作する環境において，各ゲスト計算機に適切な量のメモリを割り当てる．ホスト計算機で動作する Linux カーネルが物理メモリの管理を行い，ゲスト計算機にメモリを配分する．このような環境においては，ゲスト計算機間で物理的な計算機資源が共有されているため，このような限られた資源を有効に機能するよう配分することが重要である．

ゲスト計算機は，QEMU による仮想計算機で実現されている．QEMU とゲスト計算機で動作する Linux カーネルは，メモリバルーニングに対応しており，動作中に使用するメモリ量を増減させることができる．この機能を利用すると，使用メモリ量の少ないゲスト計算機のメモリ量を減らし，その分を多くのメモリを使用するゲスト計算機に割り当てることで，限られたメモリ資源を効率的に使用することが可能になる．

各計算機において，メモリはカーネルやプロセスに割り当てられ，それぞれが必要とする量が使用される．メモリが不足した場合には，仮想記憶が使用される．ただし，仮想記憶の使用で必要となるスワップイン，スワップアウトの処理は負荷が高い．したがって，システムを適切に動作させるために，仮想記憶を使用せずにすむよう，少なくともカーネルやプロセスが必要とする量のメモリを割り当て運用することが多い．

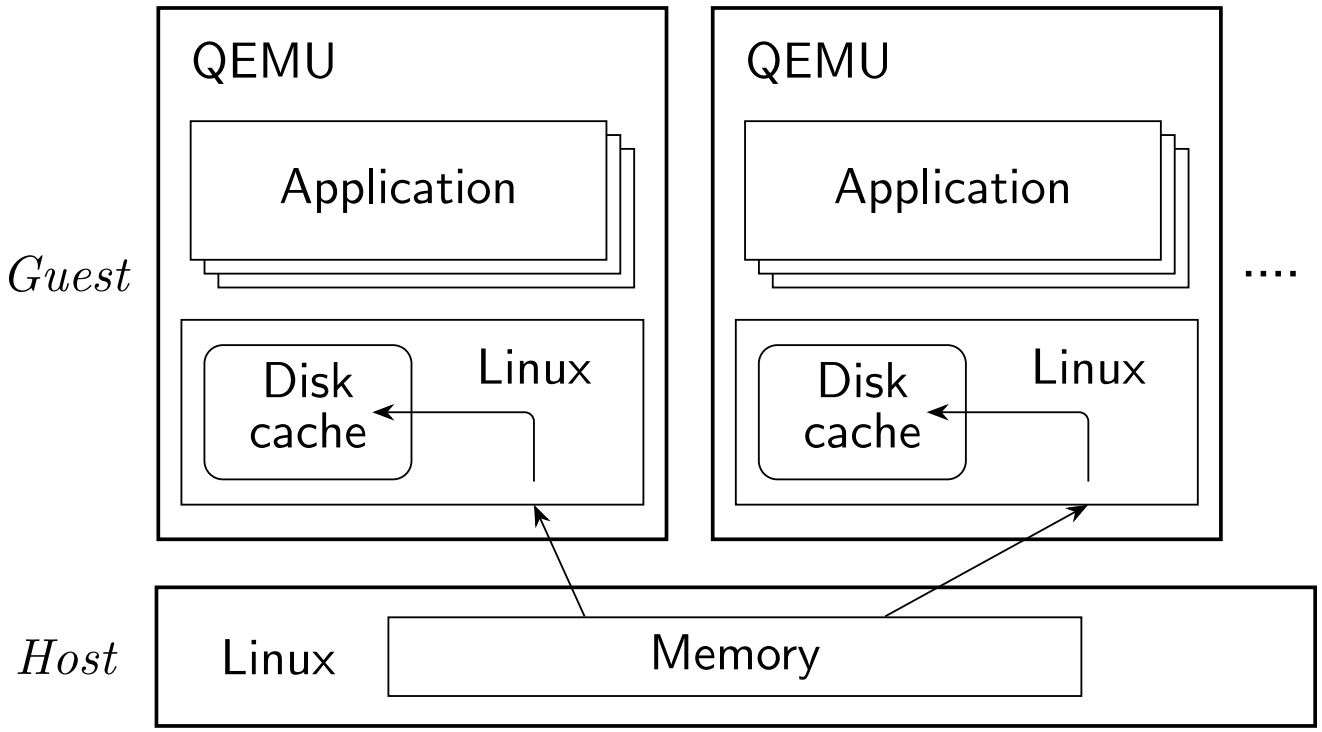

図 4　ゲスト計算機へのメモリ割り当て

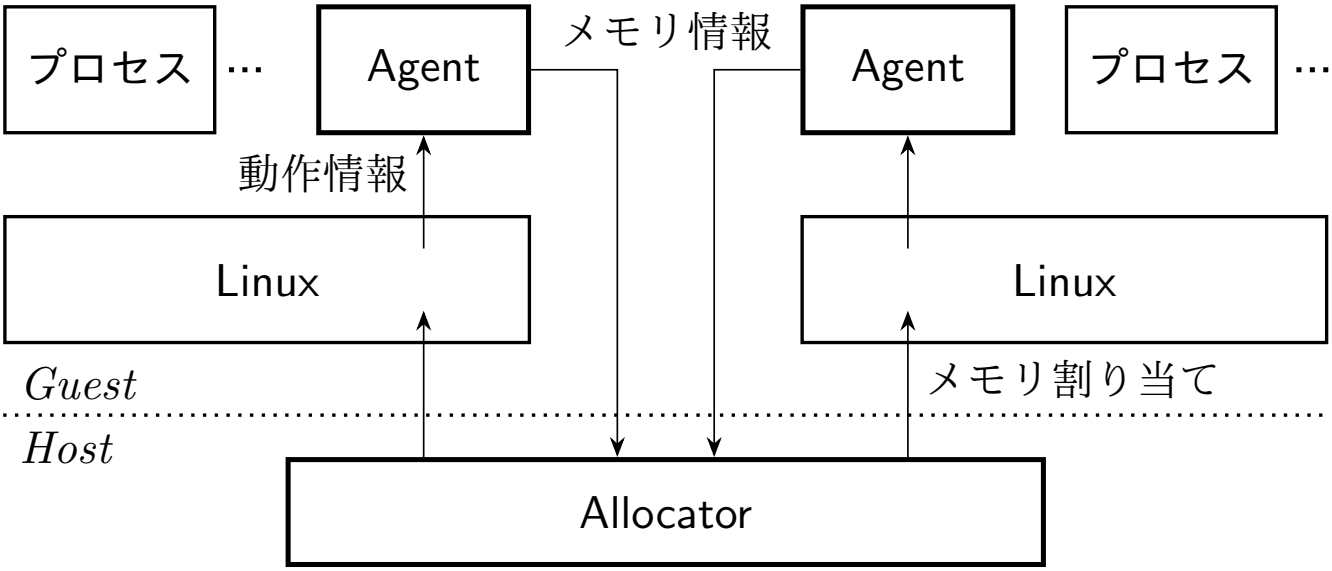

図 5　提案システムの構成

各ゲスト計算機ではそれぞれ固有のプロセスが動作し，それらにより各ゲスト計算機で行われるべき処理が実現される．このとき，プロセスは自身が必要とする量のメモリを使用して動作し，通常，ゲスト計算機が持つメモリ量に応じて動作を変えることはない．

カーネルやプロセスが必要とする以上のメモリが使用できる場合，当該メモリはカーネルがディスクキャッシュとして使用する．すなわち，ゲスト計算機に新たにメモリが割り当てられたとき，そのメモリはディスクキャッシュとして使用される．逆に，メモリを減らす場合には，ディスクキャッシュを破棄し，そのメモリを解放する．

通常，キャッシュ量が多いほどキャッシュヒット率が高くなるため，多くのメモリをディスクキャッシュに使用できると I/O 性能が向上する．提案システムは，ゲスト計算機のキャッシュの動作をもとに，ゲスト計算機へのメモリ割り当て量の設定を行い，システム全体でのメモリ利用の効率化を図る．

図 5 に示すように，提案システムでは，次の 2 つのコンポーネントが協調して動作し，ゲスト計算機へのメモリ割り当てを行う．

- Allocator

  ゲスト計算機へのメモリ割り当て量を決定する

- Agent

  ゲスト計算機のメモリ使用状況を監視する

Allocator は，ホスト計算機で動作するプロセスである．提案システムの中核として動作し，ゲスト計算機の動作状況をもとに割り当てるべきメモリ量を計算し，ゲスト計算機へのメモ

リ配分を行う．Agent は，各ゲスト計算機で 1 個ずつ動作するプロセスである．ゲスト計算機で動作する Linux カーネルの動作を監視し，メモリ使用状況を Allocator に通知する．

図 6 は Agent の動作をあらわしている．Agent は，BCC で eBPF を用いて Linux カーネルの関数を監視し，キャッシュヒット数，キャッシュミス数を数える．ゲスト計算機がマウントしたホスト計算機のディレクトリにあるファイルにキャッシュヒット数，キャッシュミス数を記録することで Allocator への通知を行う．キャッシュヒット数，キャッシュミス数の記録は毎秒行う．

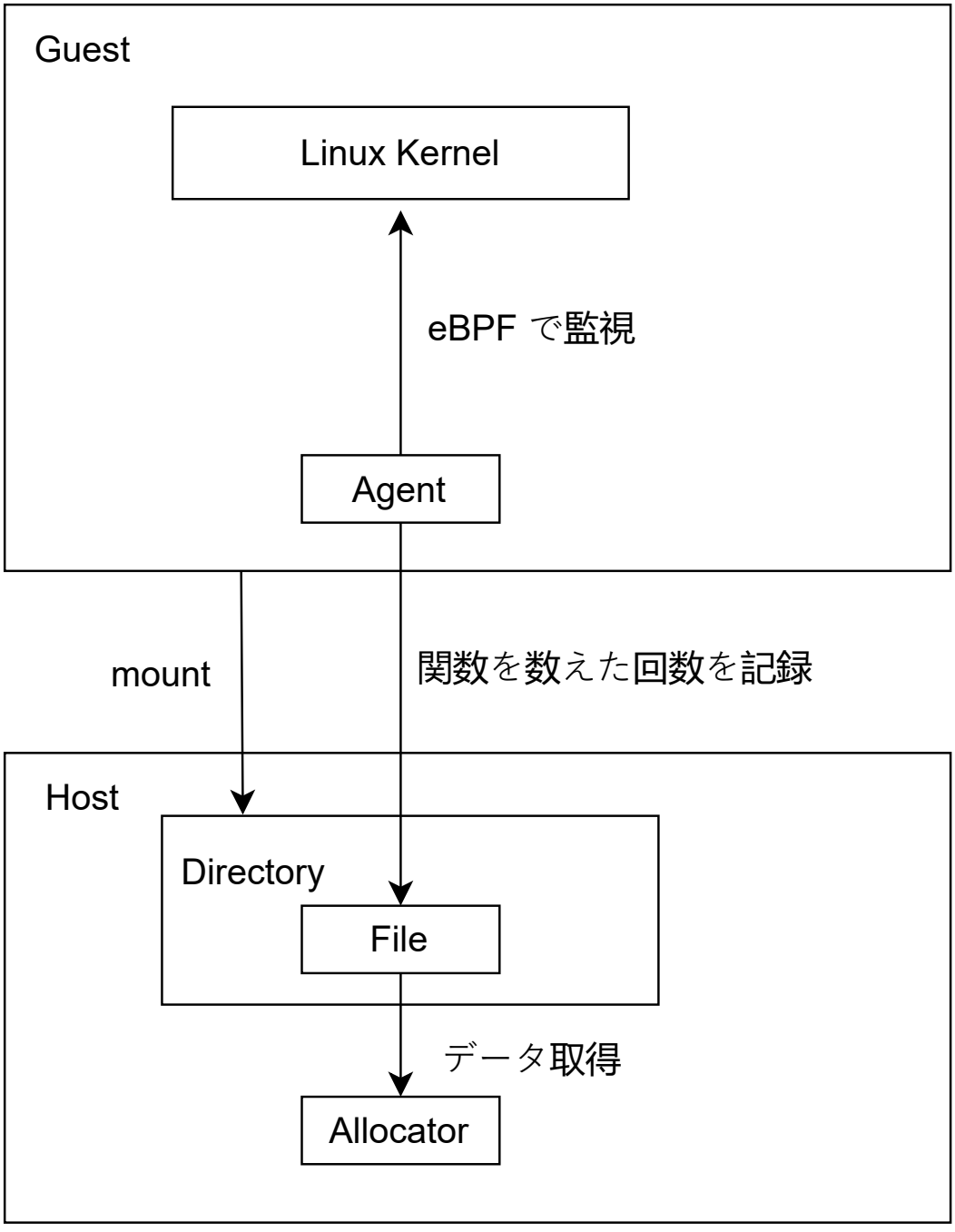

図 6　Agent の動作

4　割当メモリ量の設定

4.1　一定量移動

Allocator が算出する割り当てるメモリ量の算出方法の一つとして，キャッシュヒット数，キャッシュミス数，メモリサイズを使用する．キャッシュヒット数が大きいとキャッシュメモリがデータや命令を保持している回数が多いことを意味する．キャッシュヒット数が大きいとキャッシュメモリがデータや命令を保持していない回数が多いことを意味する．そのため，キャッシュヒット数が大きいゲスト計算機のメモリを減少させたとしても，キャッシュヒット数が減らない可能性がある．キャッシュミス数が大きいゲスト計算機はデータや命令を読み始めたか，メモリが十分にない．メモリが十分にない場合は，キャッシュミス数の多いゲスト計算機のメモリを増加させることでキャッシュミス数が減る．本算出方法では，キャッシュヒット数が大きいゲスト計算機のメモリを減少させ，キャッシュミス数の多いゲスト計算機のメモリを増加させるようメモリを割り当てる．キャッシュヒット数の大きいゲスト計算機のメモリを少しずつ減少させ，キャッシュヒット数が小さくならない最小のメモリサイズを探す．

本項では，式 4 からなる評価値 e を用いる．h がキャッシュヒット数，m がキャッシュミス数，$memsize$ が割り当てられているメモリサイズである．

$$e = \frac{h - m}{memsize} \tag{4}$$

評価値が大きいゲスト計算機のメモリを減少させ，評価値が小さいゲスト計算機のメモリを減少させる．メモリサイズで割っているため，キャッシュヒット数とキャッシュミス数が同じ場合はメモリサイズが小さいほうが評価が大きい．そのため，評価値が最も高いゲスト計算機はメモリサイズで割らない場合に比べて変化しやすい．

Allocator では，以下のことを繰り返す．

(1) 各ゲスト計算機のキャッシュヒット数，キャッシュミス数取得

(2) 各ゲスト計算機の評価値を算出

(3) 各ゲスト計算機の評価値でソート

(4) 評価値が高いゲスト計算機と低いゲスト計算機にわける

(5) ゲスト計算機の数が奇数の場合は 1 台は変更しない．ゲスト計算機が 5 台の場合は評価値が 3 番目に高いゲスト計算機は高いゲスト計算機にも低いゲスト計算機にもならない．

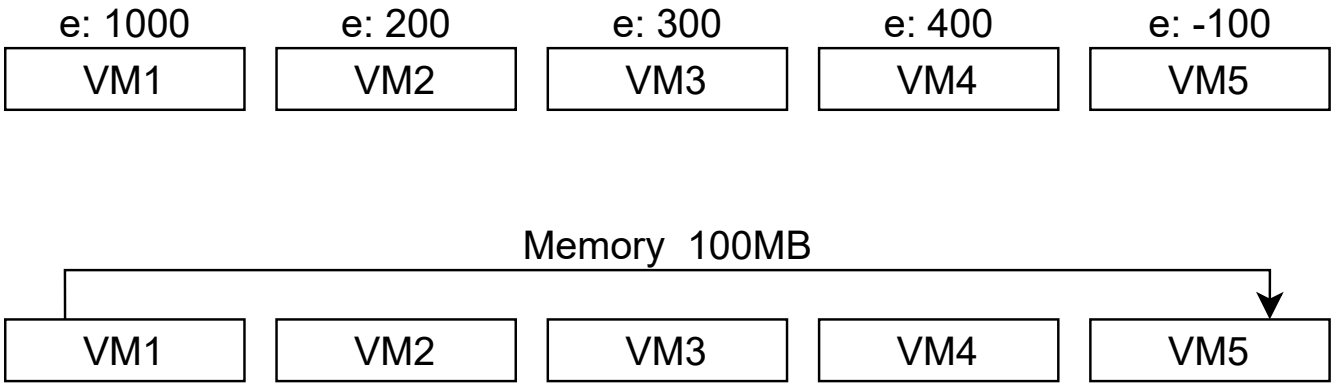

図 7 メモリの移動例

(6) 評価値の最も高いゲスト計算機と低いゲスト計算機，2 番目に高いゲスト計算機と 2 番目に低いゲスト計算機と順番に評価値を比較

(7) 評価値の差が 500 未満だとゲスト計算機のメモリの増減を行わない

(8) 評価値の差が 500 以上だと評価値の高いゲスト計算機のメモリを 100 MB 減少させ，評価値の低いゲスト計算機のメモリを 100 MB 増加させる

(9) 10 秒待機

図 7 の場合を考える．ゲスト計算機が 5 台あり，評価値は VM1 が 1000，VM2 が 200，VM3 が 300，VM4 が 400，VM5 が -100 である．評価値を降順で入れ替えると，VM1，VM4，VM3，VM2，VM5 となる．この場合，最も評価値の高い VM1 と 最も評価値の低い VM5，2 番目に評価値の高い VM4 と 2 番目に評価値の低い VM2 の評価値を比較する．VM1 と VM5 の評価値の差は 500 以上あるので，VM1 のメモリを 100 MB 減少させ，VM5 のメモリを 100 MB 増加させる．VM4 と VM2 の評価値の差は 500 未満なのでメモリの増減は行われない．VM3 はメモリの増減は行わない．メモリサイズ変更後，10 秒後に評価値を算出し同様のことを行う．

提案システムは，メモリ割り当てを繰り返すことで，メモリを減少させたゲスト計算機のキャッシュヒット数が減少せず，メモリを増加させたゲスト計算機のキャッシュミス数が減少しキャッシュヒット数が増加するようにする．その結果各ゲスト計算機のキャッシュヒット率が向上し，処理を効率的に実行できるようになる．

4.2 ベイズ最適化

Allocator は，ベイズ最適化 [8] を用いて，各ゲスト計算機へ割り当てるべき最適なメモリ量の探索を行う．キャッシュヒット率は，ゲスト計算機で動作するプロセスや操作対象となる

ファイルに依存するため，メモリ割り当て量からのみで事前に知ることは困難である．そのため，メモリを割り当て，そのときのゲスト計算機の状態での実際のキャッシュヒット率を観測するといった試行が必要である．提案システムは，このような試行を繰り返すことで，最適なメモリ割り当て量の探索を行う．

提案システムでは，ゲスト計算機へのメモリ割り当て量を変化させながら，最適なメモリ割り当て量を探索する．この探索のために，Allocator は，各ゲスト計算機に割り当てられたメモリ量と，そのときのメモリ利用効率のデータを収集する．また，このデータを用いて，各ゲスト計算機へのメモリ割り当て量を入力，そのときの全ゲスト計算機のメモリ利用効率を出力とするモデルを作成する．このようなモデルを作成することによって，メモリ利用効率が高くなるときの各ゲスト計算機へのメモリ割り当て量を算出することが可能になり，提案システムはその算出結果をもとにメモリ割り当て量の設定を行う．

Allocator と Agent は，次のように協調して，ゲスト計算機へのメモリを割り当て量の設定を行う．

(1) Allocator が，各ゲスト計算機へ割り当てるべきメモリ量を算出し，割り当てを行う．
(2) 各ゲスト計算機上の Agent が，メモリ利用効率を取得し，Allocator に通知する．
(3) Allocator が，メモリ割り当て量とメモリ利用効率の関係を計算する．

Allocator は，これまでの試行で得たメモリ割り当て量とメモリ利用効率のデータをもとに，メモリ利用効率が高くなる可能性の高いメモリ割り当て量を算出する．また，メモリバルーニングの機能を使用して，ゲスト計算機のメモリ割り当て量の変更を行う．各ゲスト計算機は，新たに割り当てられたメモリでディスクキャッシュの容量を増加させたり，ディスクキャッシュを解放しそのメモリをホスト計算機に返却したりする．

各ゲスト計算機上で動作する Agent は，割り当てられたメモリでゲスト計算機がどのように動作するかを観測する．具体的には，eBPF を使用して Linux カーネルの動作を調べ，キャッシュヒット数とキャッシュミス数を取得する．また，これらを Allocator に通知する．

Allocator は Agent から受け取った情報を用いて，全ゲスト計算機のメモリ利用効率を計算する．メモリ利用効率の指標には目的に応じて様々なものが考えられるが，有用なものとして，全ゲスト計算機の一定時間内でのキャッシュヒット数の合計やキャッシュミス数の合計がある．キャッシュヒット数は大きいほど，キャッシュミス数は小さいほど，メモリ利用効率が良いと考えることができる．したがって，すべてのゲスト計算機でのキャッシュヒット数の合計をメモリ利用効率の評価値することができる．このような評価値はシステム全体の

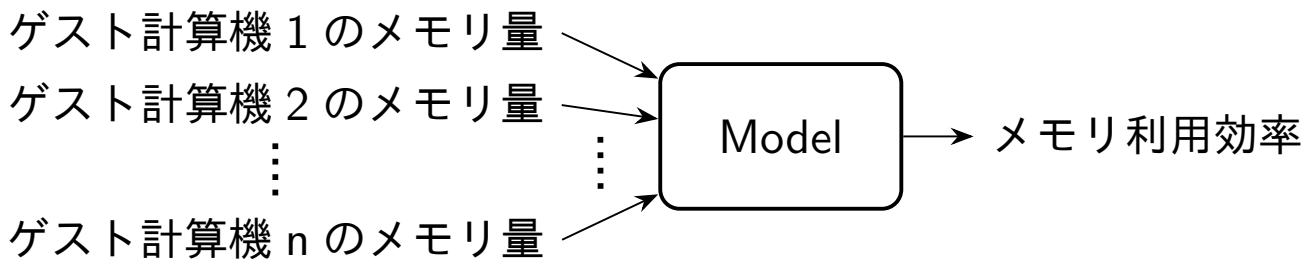

図 8　メモリ割り当て量からのメモリ利用効率の推測

性能を評価するが，すべてのゲスト計算機に公平にメモリが割り当てられているかは評価しない．ゲスト計算機間の差をなくすために，キャッシュミス数の分散が小さくなるほど良いと評価することも考えられる．

Allocator は，各ゲスト計算機に割り当てられたメモリ量と算出したメモリ利用効率を，過去の試行のものと合わせ，ガウス過程回帰によって，メモリ割り当て量とメモリ利用効率の関係を表すモデルを作成する．提案システムは，以上を繰り返すことで，モデルの精度を高め，より適切にメモリ割り当て量を設定できるようにしている．

Allocator が作成するモデルを図 8 に示す．モデルの入力は各ゲスト計算機へのメモリ割り当て量のベクトルであり，出力はそのときの全ゲスト計算機のメモリ利用効率のスカラー値である．このようなモデルを用いることで，メモリ割り当て量に対するメモリ利用効率を計算することができる．そのため，実際にはメモリ割り当てを行い動作を調べていないメモリ割り当て量に対してもメモリ利用効率の推測を行うことができる．

5　提案システムの動作

単一のホスト計算機に複数のゲスト計算機を仮想計算機として動作する環境を考える．このとき，3 つのケースが考えられる．全てのゲスト計算機が処理に対してメモリが十分にある場合，メモリが十分にあるゲスト計算機とメモリを必要とするゲスト計算機の両方が存在する場合，全てのゲスト計算機が処理に対してメモリが必要量ない場合である．これら 3 つのケースを提案システムの算出方法である少しずつメモリを移動させる方法とベイズ最適化を利用する方法，提案システムを利用しないときの動作を示す．

5.1　全てのゲスト計算機が処理に対してメモリが十分にある

全てのゲスト計算機が処理に対してメモリが十分にある場合は，提案システムが理想とする状態でありゲスト計算機のメモリの増減は必要ない．しかし提案システムのメモリの増減を行わない条件に入らない限りゲスト計算機のメモリサイズを変更する．

メモリを少しずつ移動させる手法の場合，各ゲスト計算機を評価値でソートしてメモリを増減させるゲスト計算機を決定する．このときメモリを増加させるゲスト計算機とメモリを減少させるゲスト計算機の評価値の差が 500 未満だとメモリの増減はない．500 以上の差がある場合は評価値の高いゲスト計算機はメモリが連続で減少し，評価値の差が 500 未満になるか評価値でソートしたときに評価値が低いグループに入るまで続く．また，評価値でソートした時に評価値が低いグループに入っているゲスト計算機もメモリは十分にあるため，メモリが上昇したとしてもキャッシュヒット数は増加しない．結果として，ゲスト計算機のメモリを増減させない条件に入るか，評価値が高いグループにいるゲスト計算機のキャッシュヒット数が減少することが評価値が小さくなり，評価値が低いグループになったことでメモリを増加させるといったことが繰り返される．

ベイズ最適化を用いる場合は，提案システムを含む機構を動かし始めたときのみ必要のないメモリの増減が行われるが，時間がたつにつれて増減するメモリの量は減っていくと考えられる．ベイズ最適化ツールとして使用した Ax は，ゲスト計算機の数によって変わるが最初数回はソボル列で生成された値で割り当てる．そのため提案システムを動作させ始めたときは必要のないメモリの増減を行ってしまう可能性が高い．時間が立ち学習を繰り返すことでメモリの最適値を発見し，処理に大きな変更がない限りメモリの増減は減ると予想される．

提案システムを使わない場合は，余計なメモリの増減を行わないため，提案システムを実行するよりも早く実行中の処理が終了する．

5.2 メモリが十分にあるゲスト計算機とメモリを必要とするゲスト計算機の両方が存在

メモリが十分にあるゲスト計算機とメモリを必要とするゲスト計算機の両方が存在する場合，提案システムで適切なメモリ割り当てを行い実行中の処理を提案システムを使わない場合と比べて早く終わる．唯一の例外として，少しずつメモリを移動させる手法で処理に必要とするメモリサイズよりも仮想計算機に割り当てるメモリ総量よりも大きい場合は全てのゲスト計算機がメモリを必要とする状態となってしまい割り当てない場合よりも処理速度が落ちる．メモリを少しずつ増減させる手法とベイズ最適化を使用する手法ではどちらが早く終わるかはわからない．

メモリを少しずつ移動させる手法の場合，メモリを必要とするゲスト計算機にメモリを渡し続ける．各ゲスト計算機を評価値でソートしてメモリを増減させるゲスト計算機を決定し，このときメモリを増加させるゲスト計算機とメモリを減少させるゲスト計算機の評価値の差が 500 未満だとメモリの増減はないためメモリを増減させない条件に入るか，評価値の高いゲスト計算機の必要なメモリまで減少しキャッシュヒット数が減り評価値が小さくなるか，評価値の低いゲスト計算機のメモリが増加したことにより評価値の低いゲスト計算機のキャッシュミス数が増加して評価値が増加することでゲスト計算機間の評価値の順位が変更される．最終的に，仮想計算機に割り当てるメモリサイズの総量が仮想計算機で行う処理全ての必要とするメモリサイズよりも大きい場合はゲスト計算機で行う処理に対してメモリが十分にある状態になる．反対に仮想計算機に割り当てるメモリサイズの総量が処理全ての必要とするメモリサイズよりも小さい場合は全てのゲスト計算機がキャッシュミス数を多く出す状態になる．仮想計算機に割り当てるメモリサイズの総量と処理全ての必要とするメモリサイズの差が小さい場合は，メモリが十分にあるゲスト計算機とメモリが十分にないゲスト計算機両方がある状態となり，メモリが十分にあるかどうかはゲスト計算機ごとに入れ替わる．

ベイズ最適化を用いる場合は，効率の良い割り当てるメモリサイズを発見するまでの時間によってメモリを少し増減させる場合と処理速度が変わる．

提案システムを使わない場合，メモリが十分にあるゲスト計算機の処理が早く終わりメモリが必要量ないゲスト計算機の処理に時間がかかる．メモリが必要量ない場合だと CPU 資源の利用も不安定になるためメモリが十分にあるゲスト計算機と比較し著しく遅くなる．

5.3 全てのゲスト計算機が処理に対してメモリが必要量ない

メモリを少しずつ増減させる方法だと，全てのゲスト計算機がキャッシュミスを全てのゲスト計算機に十分にメモリがある場合とどちらもある場合と比べて多く出す．評価値はキャッシュヒット数からキャッシュミス数を減算することでマイナスになる．その結果，キャッシュヒット数とキャッシュミス数が同じでメモリサイズが同じ場合はメモリサイズが大きいゲスト計算機の評価値が高くなる．ゲスト計算機 1 つの実行するメモリを多く使う処理が終わると，割り当てるメモリサイズの最小値になるまでメモリは減少し他のゲスト計算機へと移動する．ゲスト計算機の処理が終わることが早いとベイズ最適化，提案手法を使わない場合と比べて早く終わる．

ベイズ最適化を使用する手法の場合，いくつかのパターンが考えられるが，多くは 2 つのパターンになる．一つのゲスト計算機にメモリを集中させて処理を早く終わらせるか，必要とするメモリサイズの大きい順番にメモリを多く割り当てるかである．一つのゲスト計算機にメモリを集中させる場合は早く処理が終わるので，目的次第では有用である．必要とするメモリサイズの大きい順番にメモリを多く割り当てる場合，メモリが足らないことにより CPU 資源も満足に使うことができないが，少しずつメモリを増減する手法と提案手法を使用しない場合と比べてキャッシュヒット数は大きくなるため全ての処理が比較的早く終了する．

提案システムを使用しない場合，処理次第だが最も時間がかかる．たとえ一つのゲスト計算機が処理を終了したとしても，処理を終了させたゲスト計算機はメモリを他のゲスト計算機へと移動することがないからである．

6 評価

本章では，提案システムによるゲスト計算機へのメモリ割り当て量の設定を評価するために行った実験について述べる．実験に使用した環境を表 1 に示す．ホスト計算機は，十分な量のメモリを搭載しており，仮想記憶の使用はない．同様に，ゲスト計算機でも仮想記憶の使用はない．また，全ゲスト計算機の仮想ディスクのイメージファイルは，同じ物理 SSD 上にある．Allocator が行うベイズ最適化には Ax[9, 10] を使用している．また，メモリ利用効率として，全ゲスト計算機のキャッシュヒット数の合計を使用している．

6.1 QEMU のメモリ扱い

Allocator では，以下のことを繰り返す．各ゲスト計算機で同じ内容のファイルを読み込んだ際にホスト計算機のキャッシュメモリでは同じものとして使用されていないことを確かめる．本実験では，ゲスト計算機 G1 と G2 に同じ内容の 2 GB のファイルを作成し，以下の順番で読み込んだ際の時間を計測した．

(1) ベースファイルに 2 GB のファイルを作成

(2) ベースファイルから差分ファイル G1 と G2 を作成

(3) G1 と G2 のゲスト計算機起動

(4) G1 で 2 GB のファイル読み込み速度計測

(5) G1 で 2 GB のファイル再度読み込み速度計測

(6) G1 のキャッシュ削除後 2 GB のファイル読み込み速度計測

(7) G1 で 2 GB のファイル読み込み速度計測

表 1　実験環境

ホスト計算機	
CPU	Intel Core i5–11400 2.60 GHz
メモリ	64 GB
OS	Linux–6.5.0-1009-oem
Virtualizer	QEMU–6.2.0

ゲスト計算機	
CPU	QEMU Virtual CPU version 2.5+ 2 コア
メモリ	最大 6 GB
OS	Linux–5.15.148

表 2　2 GB 読み込む速度

実験内容	速度 (s)
(4) G1 読み込み	1.306
(5) G1 再度読み込み	0.183
(6) G1 キャッシュ削除後読み込み	1.312
(7) G1 再度読み込み	0.188
(8) G1 再起動後読み込み	1.299
(9) G1 再度読み込み	0.192
(10) G2 読み込み	1.283
(11) G2 再度読み込み	0.185

(8) G1 を再起動．その後 2 GB のファイルを読み込み速度計測

(9) G1 で 2 GB のファイル再度読み込み速度計測

(10) G2 でファイル読み込み速度計測

(11) G2 で再度ファイル読み込み速度計測

表 2 は 2 GB のファイルを読み終えるまでの速度である．使用した 2 GB のファイルはベースファイルに作成したときから変更はない．

G1 で最初にファイルを読み込んだ時，G1 を再起動後ファイルを読み込んだ時，G1 のキャッシュを削除後ファイルを読み込んだ時と G2 で最初にファイルを読み込んだときが近い時間となった．また G1 と G2 で再度読み込んだときも近い時間となった．このことから，同じ内容のファイルを扱っていたとしてもゲスト計算機が違う場合はホスト計算機では別物として扱っていることがわかる．またゲスト計算機を再起動した場合もキャッシュメモリは削除されており，ゲスト計算機ごとにキャッシュメモリを扱っていることがわかった．

6.2　メモリが十分に割り当てられていないときの動作

本項ではメモリが十分に割り当てられている時と割り当てられていない時の動作の違いを述べる．本実験では，負荷プロセスとして 2 GB のファイルにランダムアクセスを行うことを 100 回繰り返した．このとき，乱数は正規分布であり，偏りのあるアクセスが行われる．ゲスト計算機として G1 と G2 の 2 台で実験する．2 台にそれぞれ 3 GB メモリを割り当てた時と，1 GB 割り当てた時を比較する．

図 9 は G1，G2 にそれぞれ 3 GB メモリを割り当てたときの CPU 使用率であり，図 10 は 1 GB メモリを割り当てたときの CPU 使用率を示している．グラフの横軸は経過時間であ

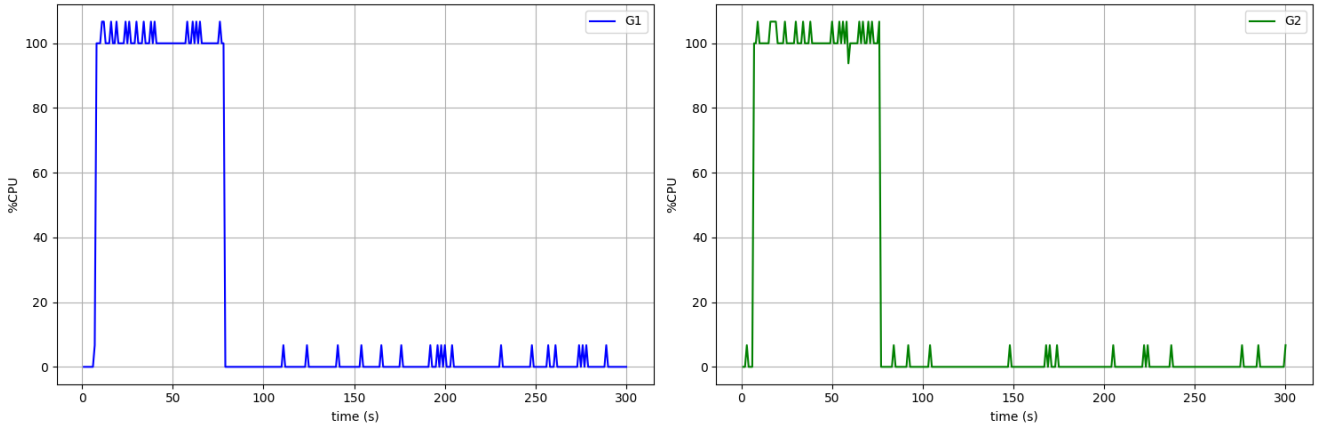

図 9　メモリが十分にあるときの CPU 使用率

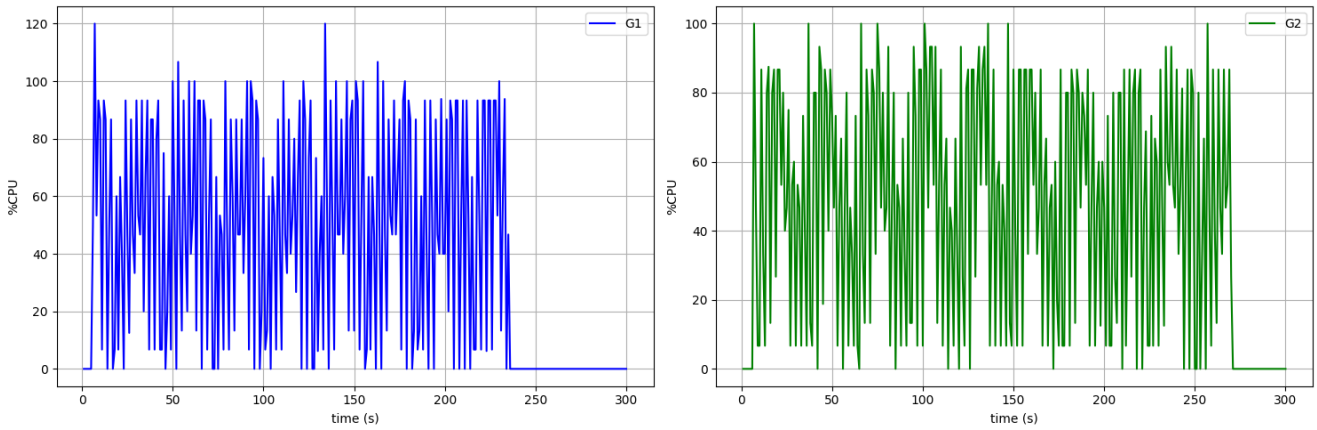

図 10　メモリが十分にないときの CPU 使用率

り，縦軸は CPU 使用率である．3 GB 割り当てた際は 96% から 103% の間で変化しており，1 GB 割り当てた際は 0% から 120% の間で変化している．CPU 使用率は 100% が CPU コア 1 つ分であり，100% 越えることがある．3 GB 割り当てた時と比べて 1 GB 割り当てた際は CPU 使用率が大きく変化している．1 GB 割り当てられたゲスト計算機はメモリを十分に使うことができず，I/O 処理の負荷が増大したことで CPU 使用率が大きく変化したのだと考えられる．

表 3 と表 4 は同様の実験をゲストの数を変化させて行ったときの平均処理時間である．小数点以下の秒数は切り捨てた．メモリが十分にあるときは処理時間はゲストの数が変わっても変化しないが，メモリが十分にないときはゲストが増えると処理時間が増大した．メモリが十分にないときは，処理をすべてのゲストで同時に実行する時と比べ，ゲスト 1 台ずつ処理を実行するほうが速く終わる可能性がある．

提案システムではメモリが十分にない状態を減らし，処理速度を増加させる．

表 3　メモリが十分にあるときのゲスト数の違いによる処理時間

ゲスト数	平均処理時間	最大処理時間	最小処理時間
1	1m20s	1m20s	1m20s
2	1m12s	1m11s	1m13s
3	1m21s	1m19s	1m22s
4	1m23s	1m22s	1m25s

表 4　メモリが十分にないときのゲスト数の違いによる処理時間

ゲスト数	平均処理時間	最大処理時間	最小処理時間
1	2m7s	2m7s	2m7s
2	4m16s	3m50s	4m25s
3	6m42s	6m26s	7m11s
4	9m29s	9m8s	10m17s

6.3　メモリ割り当て量の設定

提案システムの基本的な性能を確認するための実験を行った．本実験では，ファイルの読み出し処理を行う負荷プロセスが動作するゲスト計算機 G1 と G2 の 2 台に対して，メモリ割り当て量の設定を行った．負荷プロセスは，3 GB のファイルにランダムアクセスを行う．このとき，乱数は正規分布であり，偏りのあるアクセスが行われる．Allocator は，6 GB のメモリをこの 2 台のゲスト計算機に分配する．ただし，各ゲスト計算機には少なくとも 2 GB のメモリが割り当てられる．すなわち，各ゲスト計算機のメモリ量は，2 GB から 4 GB までの間で変化することになる．

図 11 は，各試行において，各ゲスト計算機に割り当てられたメモリ量を示している．グラフの横軸は試行回数であり，縦軸はメモリ量である．この実験においては，初めの 5 回の試行では，ソボル列で生成された値で割り当てるメモリ量が決められている．これらの試行によって，探索範囲全体でのメモリ利用効率の概略が取得される．6 回目以降の試行では，ガウス過程回帰によって生成されたモデルをもとに，ベイズの最適化によって試行すべき設定値が算出され，それによる試行が行われる．

本実験では，各ゲスト計算機は 3 GB のファイルにアクセスするが，これらのゲスト計算機には合計 6 GB のメモリが割り当てられる．そのため，一方のゲスト計算機に極端に多くのメモリを偏って割り当てないかぎり，各ゲスト計算機はディスクキャッシュのメモリを十分に確保することができる．Allocator は，メモリ割り当て量を様々に変化させたが，多くの場

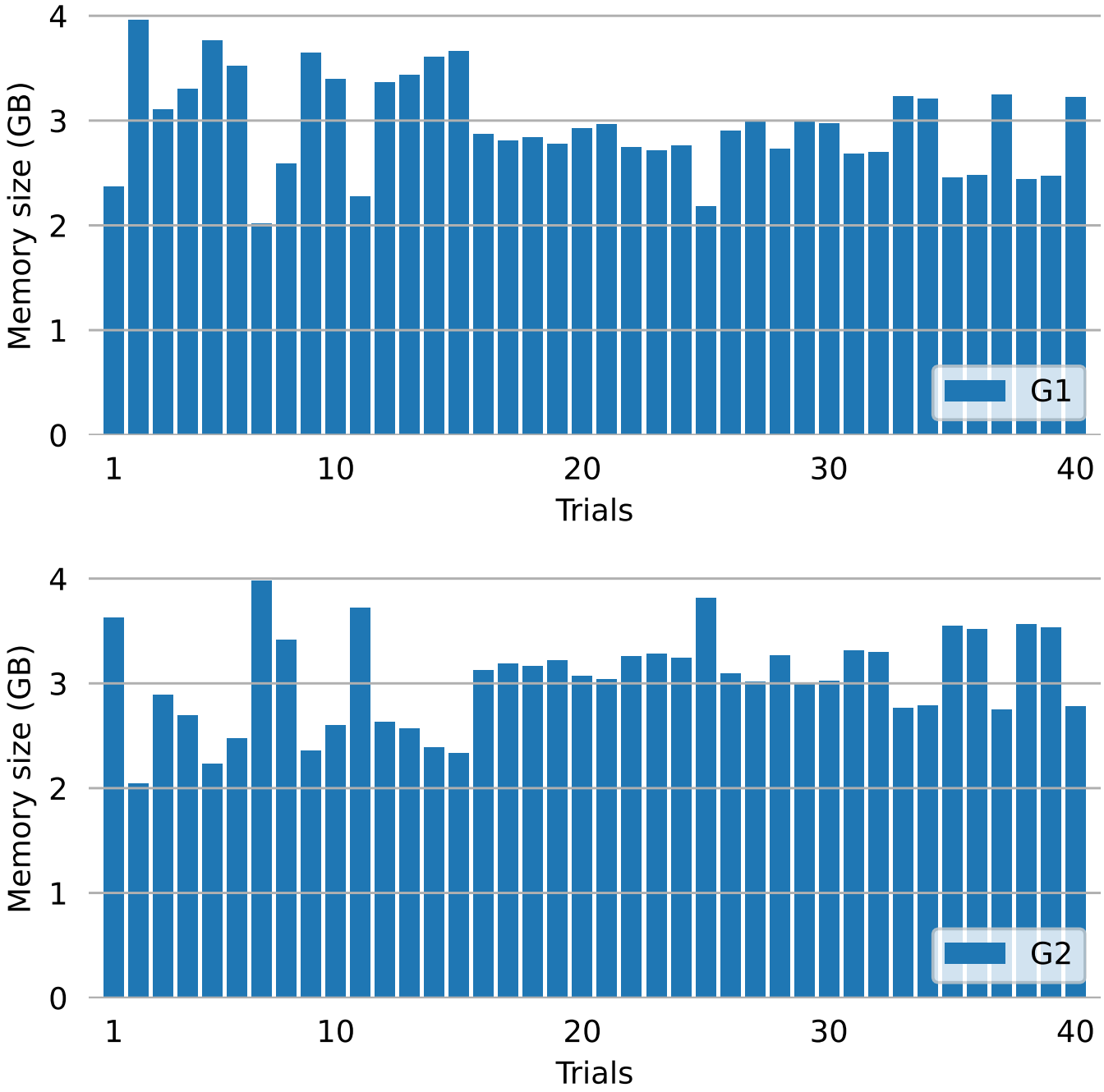

図 11　各ゲスト計算機へのメモリ割り当て量

合において，負荷プロセスが行うファイルアクセスのほとんどでキャッシュが使用された．

各試行でのキャッシュヒット数を図 12 に示す．グラフの横軸は試行回数であり，縦軸はキャッシュヒット数である．多くの試行において，キャッシュヒット数が 10^7 回程度になった．ファイルアクセスのほとんどすべてでキャッシュヒットした場合，この程度のキャッシュヒット数になることから，Allocator は，ほとんどの試行において，メモリ利用効率が高くなるメモリ割り当てを行えたといえる．

40 回目の試行を終えた時点での，メモリ割り当て量に対するキャッシュヒット数の実測値

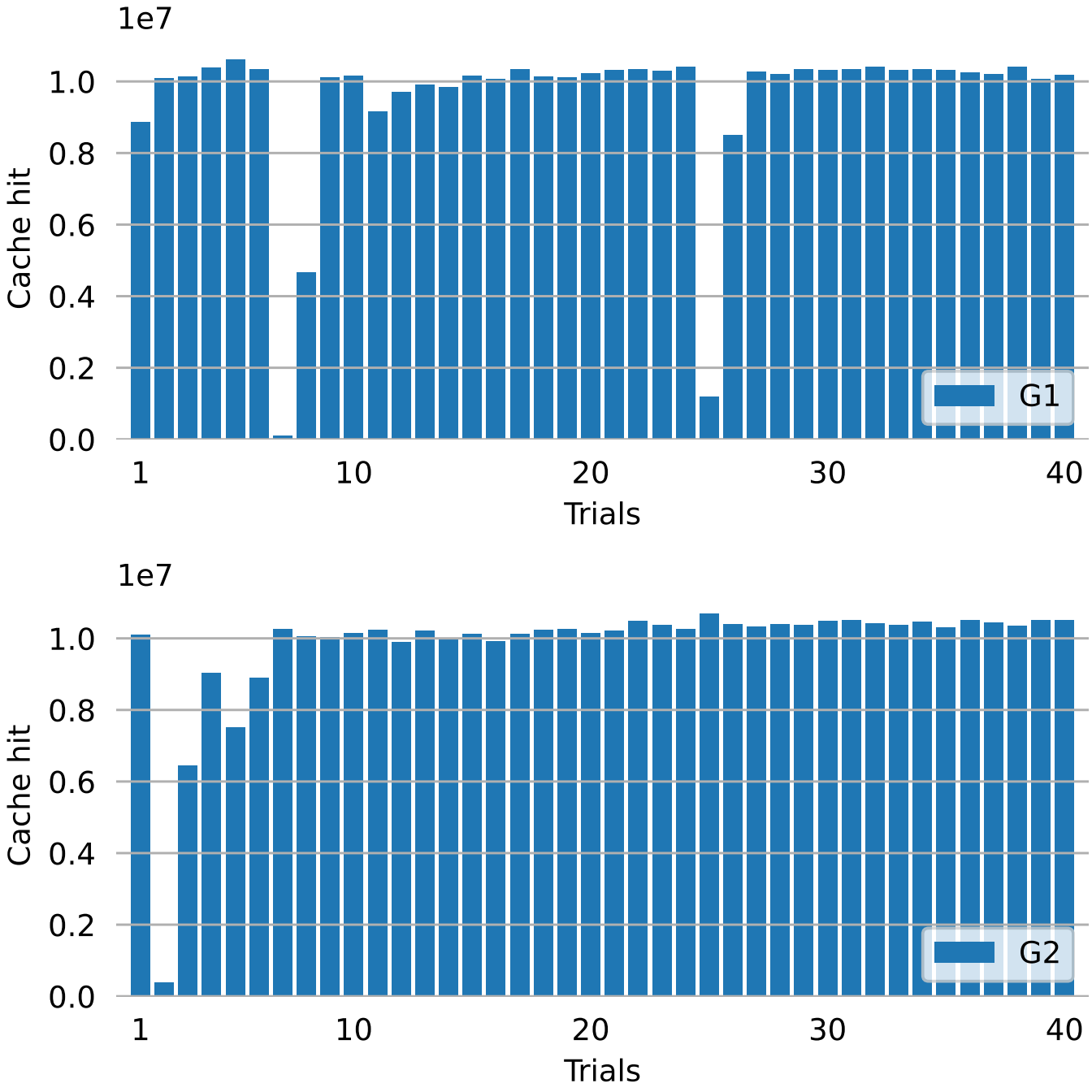

図 12　各試行でのキャッシュヒット数

と推測値を図 13 に示す．グラフの横軸はゲスト計算機 G2 へのメモリ割り当て量，縦軸はキャッシュヒット数である．グラフ内にプロットされている点は，各試行において実測されたキャッシュヒット数である．また，曲線は，メモリ割り当て量とキャッシュヒット数の関係のモデルから算出されるキャッシュヒット数の推測値である．

どちらか一方のゲスト計算機に偏って多くのメモリを割り当てると，メモリ利用効率が低下する．これはメモリ割り当て量が少なかったゲスト計算機では，十分なディスクキャッシュを確保することができず，負荷プロセスによるファイルアクセスにおいて，キャッシュ

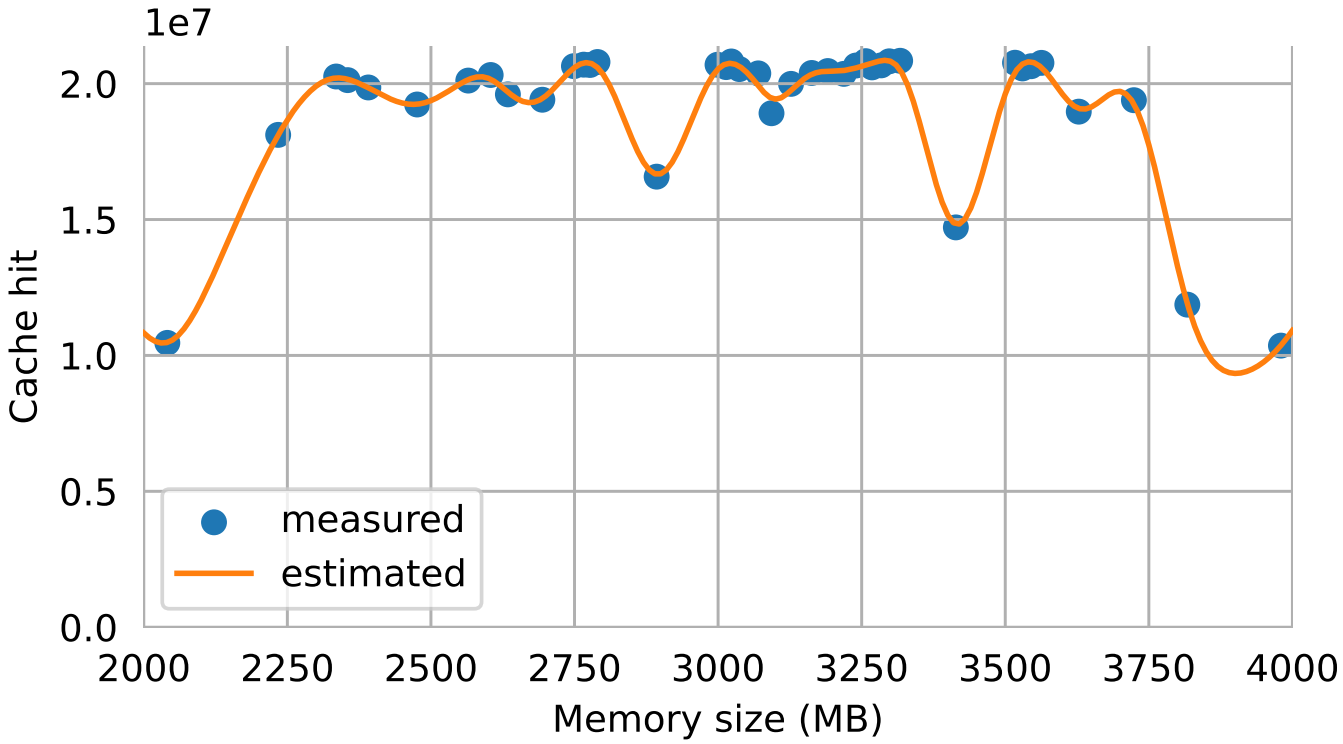

図 13　キャッシュヒット数の実測値と推測値

ヒットの発生が少なくなるためである．実測値だけでなく推測値でも，偏りすぎたメモリ割り当てを行うとメモリ利用効率が低下する傾向が現れている．

メモリ利用効率が低い試行に比べ，メモリ利用効率が高い試行の数が多かった．これは，ベイズ最適化の処理で，獲得関数に Expected Improvement を使用しているためである．Allocator は，メモリ利用効率が高くなると期待されるメモリ割り当て量を算出し，試行を行うことから，メモリ利用効率が高い領域での試行が多くなる．

以上の実験と同様の実験を，負荷プロセスが読み出すファイルのサイズを 3 GB から 4 GB に変更して行った．このときの，各ゲスト計算機へのメモリを割り当て量，各試行でのキャッシュヒット数，キャッシュヒット数の実測値と推測値を，図 14，図 15，図 16 に示す．

ファイルサイズが変更されても同様の傾向が見られるが，メモリ利用効率が高くなるメモリ割り当て量の範囲が狭くなっている．これは，ファイルサイズが大きくなったため，キャッシュヒット数を高くするためには，両方のゲスト計算機でより多くのメモリを利用可能にする必要があるためである．

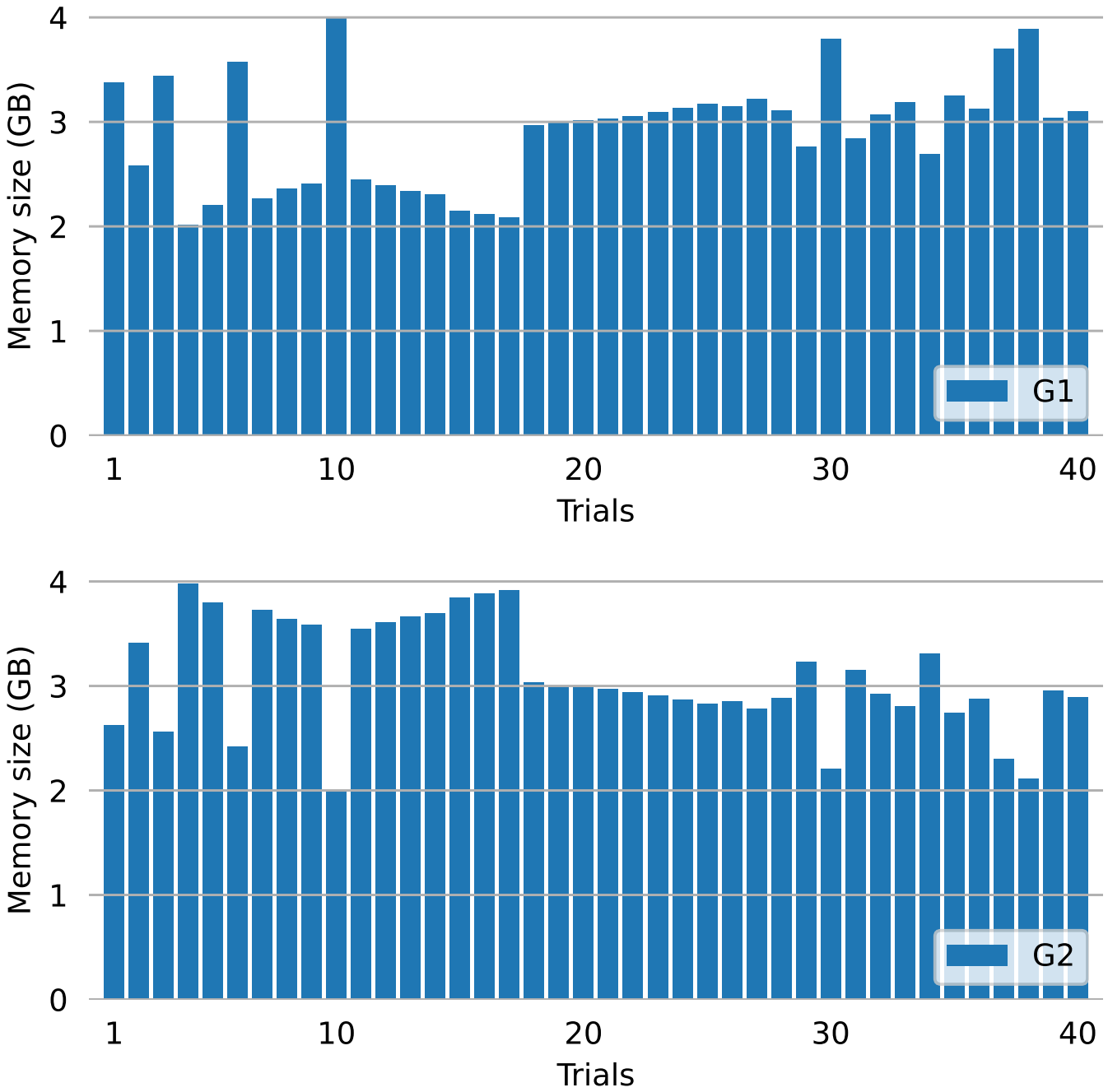

図 14　各ゲスト計算機へのメモリ割り当て量

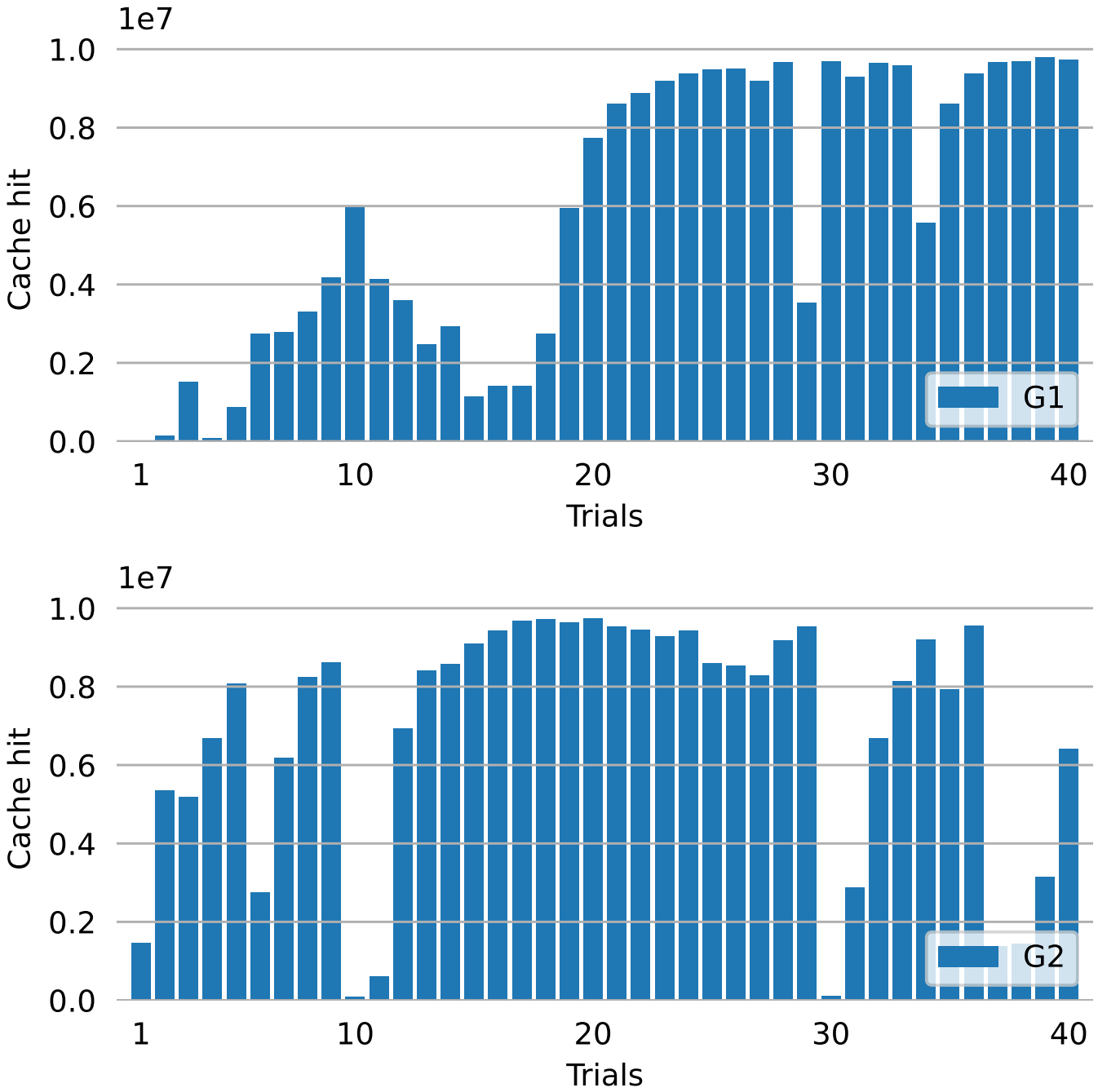

図 15　各試行でのキャッシュヒット数

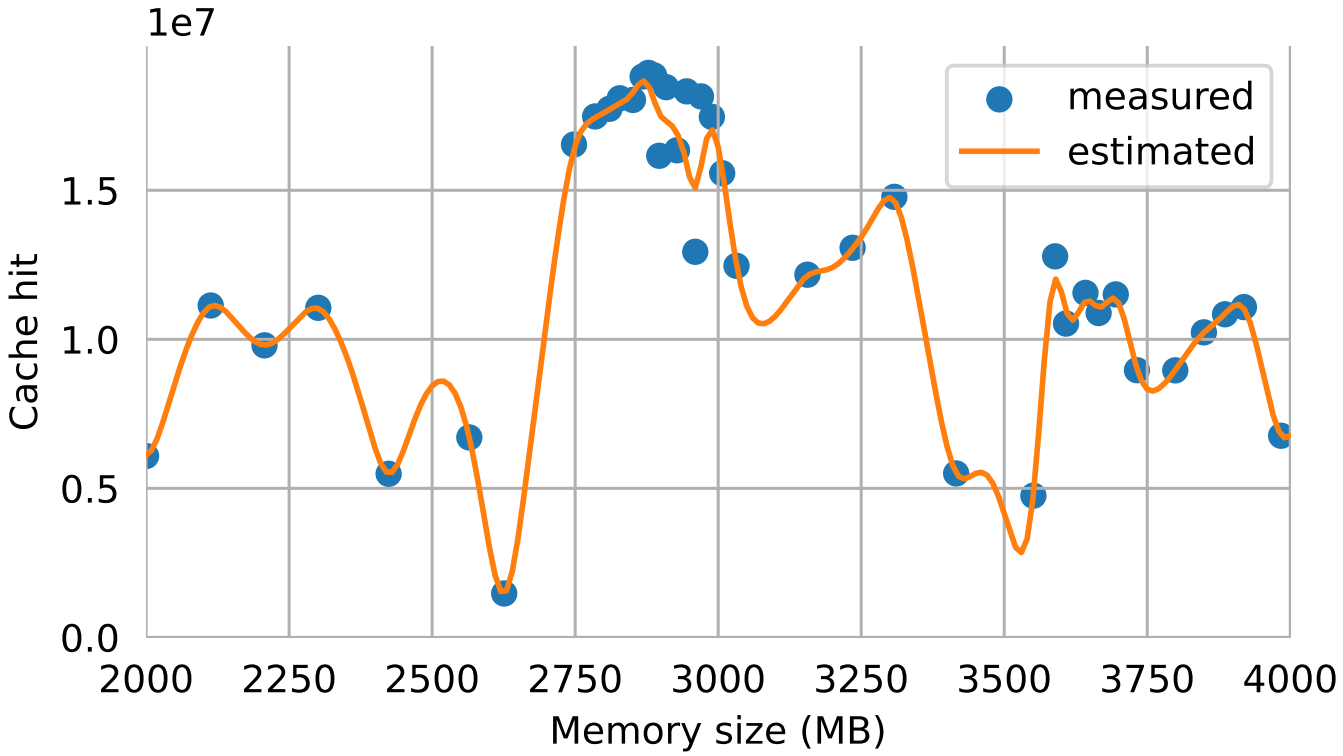

図 16　キャッシュヒット数の実測値と推測値

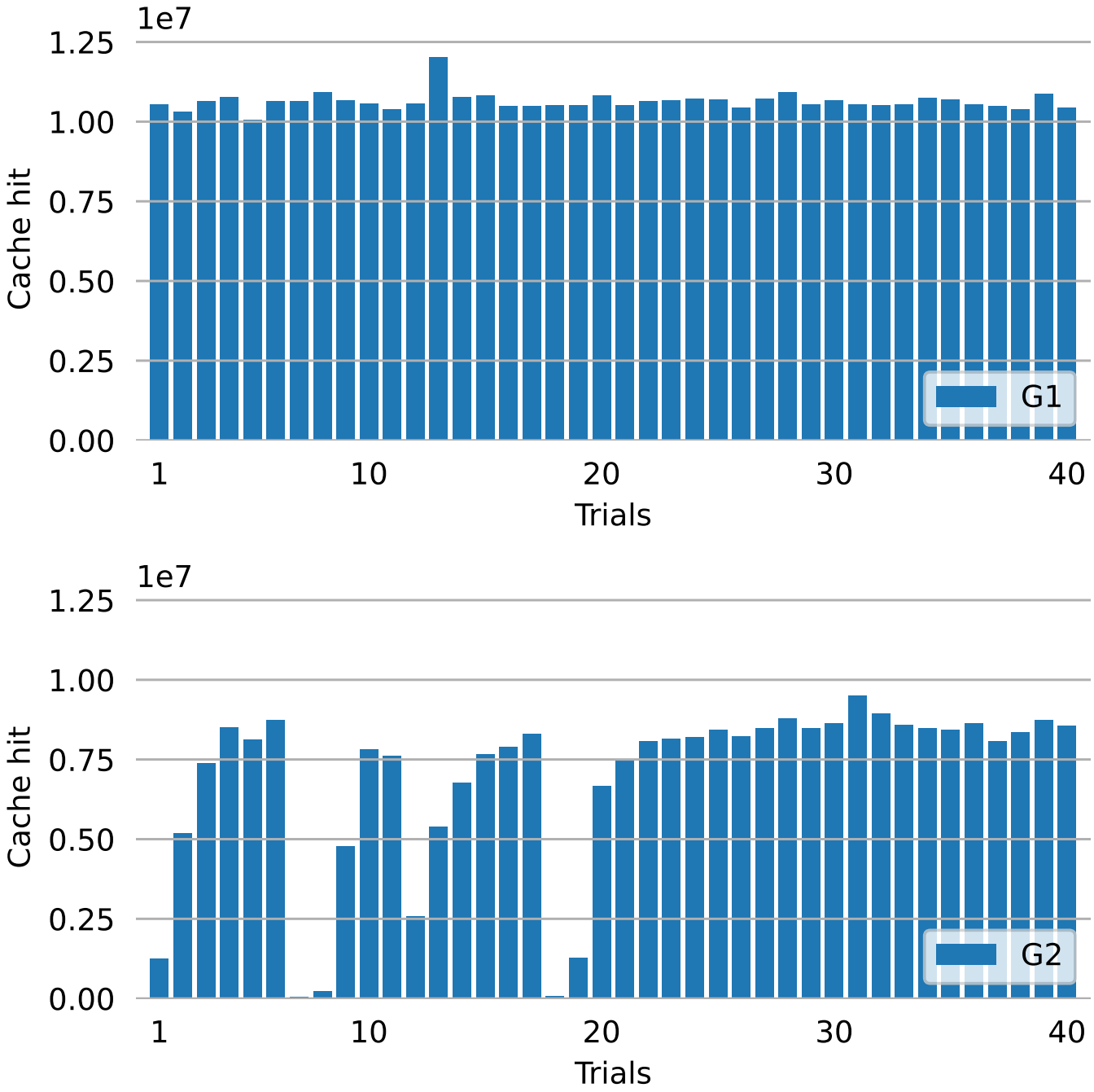

図 17　各試行でのキャッシュヒット数

6.4　異なる動作をするゲスト計算機へのメモリ割り当て

各ゲスト計算機が異なる動作をする場合の，提案システムによるメモリ割り当てについて調べる実験を行った．本実験では，6.3 節と同様の実験を行うが，負荷プロセスが読み出すファイルのサイズをゲスト計算機ごとに異なるものとする．

図 17 に，ゲスト計算機 G1 と G2 の負荷プロセスがそれぞれ 2 GB と 4 GB のファイルを読み出したときのキャッシュヒット数を示す．アクセスするファイルのサイズが小さいゲスト計算機 G1 の方がキャッシュヒット数が多くなった．また，ゲスト計算機 G2 でも，試行を

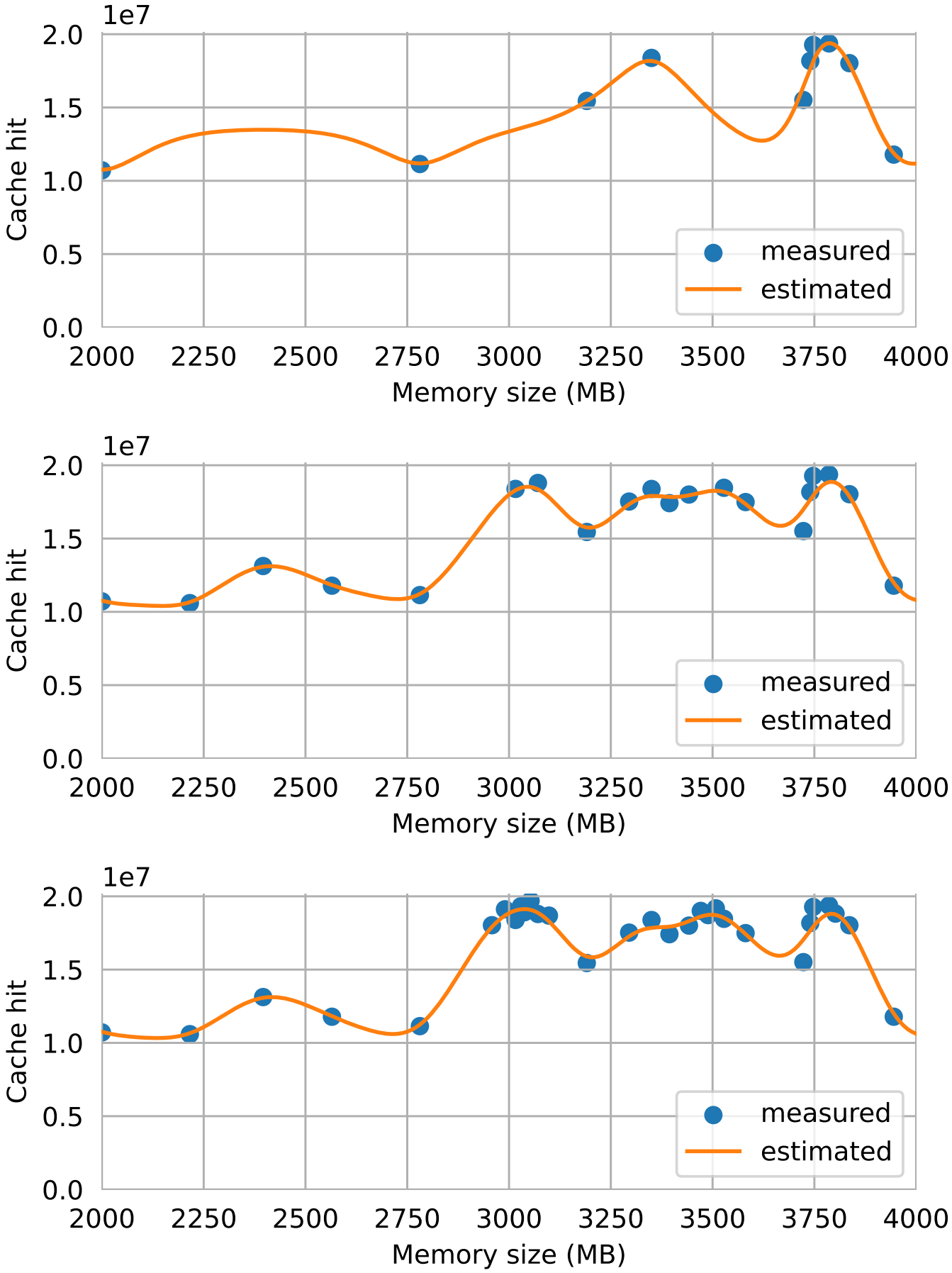

図 18　キャッシュヒット数の実測値と推測値

繰り返すことで，キャッシュヒット数が安定して多くなった．

メモリ割り当て量に対するキャッシュヒット数の実測値と推測値を図 18 に示す．グラフの横軸はゲスト計算機 G2 へのメモリ割り当て量，縦軸はキャッシュヒット数である．図 13 と同様に，グラフ内の点は実測値，曲線は推測値である．3 個のグラフは，上から，10 回，20 回，30 回の試行を終えた時点のものである．

本実験ではゲスト計算機 G2 の方が大きなファイルを扱うため，G2 に多くのメモリを割り当てることが有効である．ただし，ゲスト計算機 G2 に極端に偏ってメモリを割り当てると，ゲスト計算機 G1 のキャッシュ容量が不足し，キャッシュヒット数が低下する．Allocator は，試行を繰り返すことによって，作成したモデルからこの特徴を推測することができるようになっている．特に，20 回の試行で十分にこの特徴を推測できている．

さらに，ゲスト計算機ごとにファイルへのアクセス頻度が異なる場合の動作について調べる実験を行った．本実験では，6.3 節と同様の実験を行うが，ゲスト計算機 G1 の負荷プロセスは，2 GB/s で読み出しを行うようにした．

図 19 は，各ゲスト計算機での，各試行におけるキャッシュヒット数を示している．グラフの横軸は試行回数であり，縦軸はキャッシュヒット数である．ゲスト計算機 G1 では，ファイルアクセス頻度が低いため，キャッシュヒット数も少なくなった．ゲスト計算機 G2 では，多くの試行において，キャッシュヒット数が高くなった．

本実験では，ゲスト計算機 G2 へのメモリ割り当て量が，2 回目の試行で 2231 MB，10 回目の試行で 2000 MB と少ない場合があった．これらの試行ではキャッシュヒット数が少なくなったが，結果，ゲスト計算機 G2 のキャッシュ容量が不足したと考えられる．特に，10 回目の試行で多くのキャッシュが解放されたため，11 回目の試行でもキャッシュヒット数が少なくなった．

メモリ割り当て量に対するキャッシュヒット数の実測値と推測値を図 20 に示す．グラフの横軸はゲスト計算機 G2 へのメモリ割り当て量，縦軸はキャッシュヒット数である．図 13 と同様に，グラフ内の点は実測値，曲線は推測値である．3 個のグラフは，上から，10 回，20 回，30 回の試行を終えた時点のものである．ゲスト計算機 G2 へのメモリ割り当てが少ない場合にキャッシュヒット数が少なくなることを，2 回目や 10 回目の試行によって推測できるようになっている．

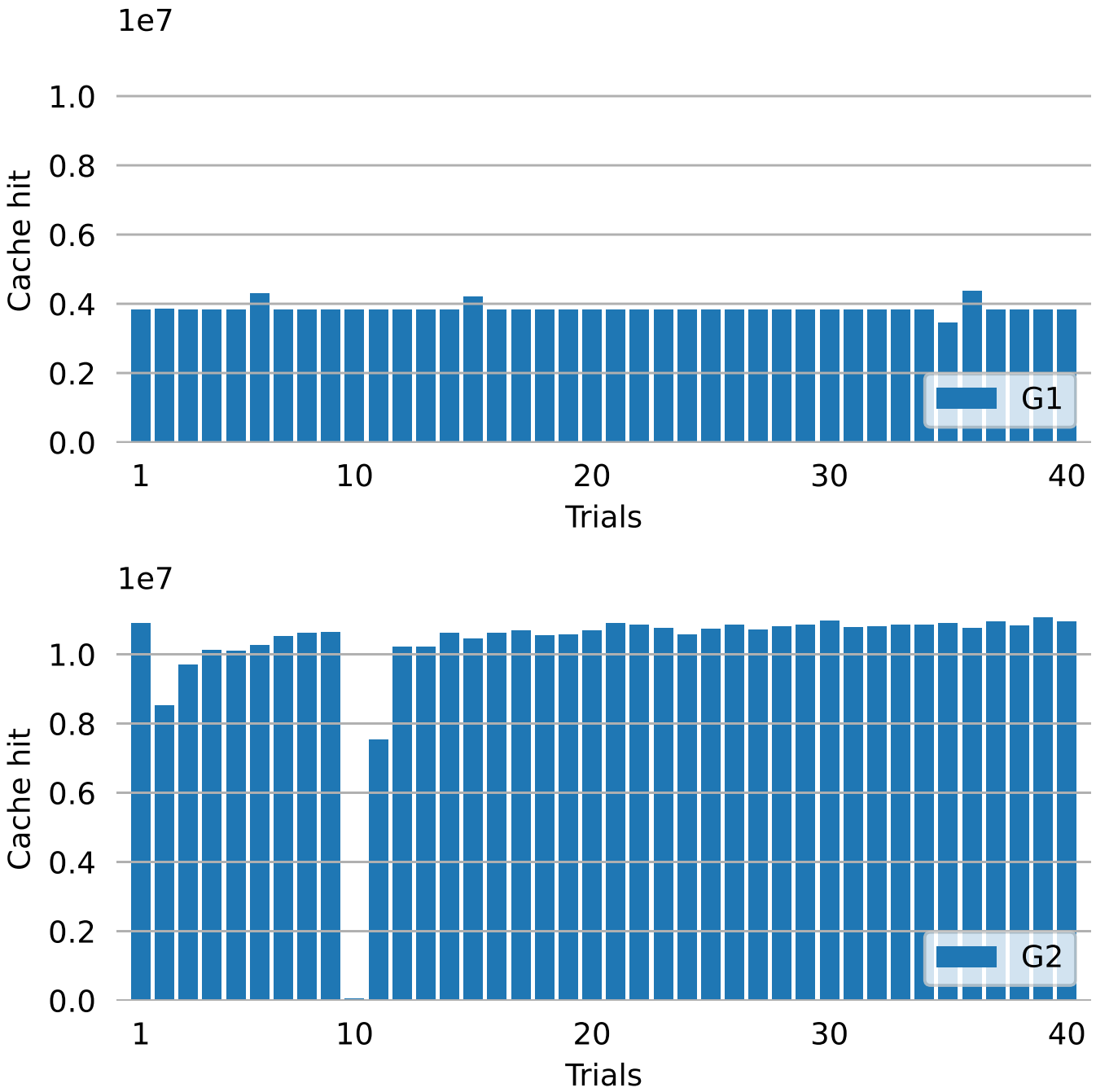

図 19　各試行でのキャッシュヒット数

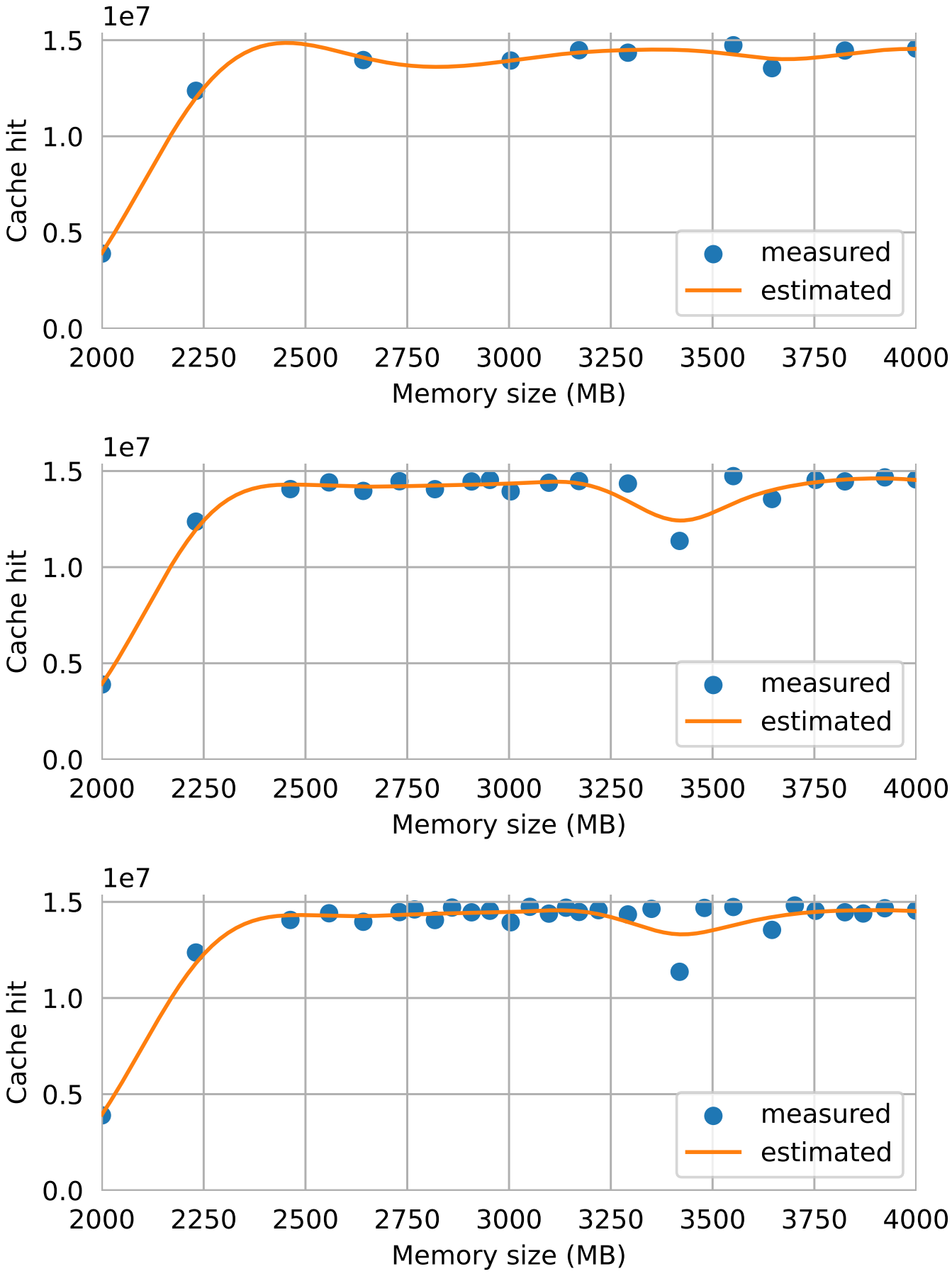

図 20　キャッシュヒット数の実測値と推測値

6.5　3 台のゲスト計算機へのメモリ割り当て

提案システムによる 3 台のゲスト計算機へのメモリ割り当てについて調べる実験を行った．本実験では，6.3 節と同様の負荷プロセスが動作するゲスト計算機 G1 ，G2，G3 に対して，メモリ割り当て量の設定を行った．負荷プロセスは，それぞれ，2 GB，3 GB，4 GB のファイルにランダムアクセスを行う．Allocator は，9 GB のメモリをこの 3 台のゲスト計算機に分配する．ただし，各ゲスト計算機には少なくとも 2 GB のメモリが割り当てられる．すなわち，各ゲスト計算機のメモリ量は，2 GB から 5 GB までの間で変化することになる．

図 21 は，各試行における，各ゲスト計算機でのキャッシュヒット数を示している．グラフの横軸は試行回数であり，縦軸はキャッシュヒット数である．Allocator は，各試行において様々なメモリ配分を行うため，それぞれの試行でキャッシュヒット数が異なるものとなる．25 回目以降の試行では，キャッシュヒット数を多くできるメモリ割り当て量を推測できており，すべてのゲスト計算機においてキャッシュヒット数が高くなっている．

図 22 は，メモリ割り当て量に対するキャッシュヒット数の実測値と推測値を示している．グラフの横軸はゲスト計算機 G3 へのメモリ割り当て量，縦軸はキャッシュヒット数である．グラフ内の点は，キャッシュヒット数の実測値である．また，曲線は，ゲスト計算機 G2 のメモリ割り当て量が 2000 MB，2500 MB，3000 MB，3500 MB，4000 MB，4500 MB のときの，ゲスト計算機 G3 へのメモリ割り当て量に対するキャッシュヒット数の推測値である．

ゲスト計算機 G2 へのメモリ割り当て量が 2000 MB，5000 MB の場合は，キャッシュヒット数が少なくなると推測している．これは，ゲスト計算機 G1 と G3 に偏ってメモリを割り当てたり，ゲスト計算機 G2 のみに偏ってメモリを割り当てた場合である．逆に，ゲスト計算機 G2 へのメモリ割り当て量が 3000 MB，3500 MB，4000 MB の場合には，キャッシュヒット数は多くなると推測している．このように，Allocator は，試行を繰り返すことでキャッシュヒット数を高くできる適切なメモリ割り当て量を探索することができている．

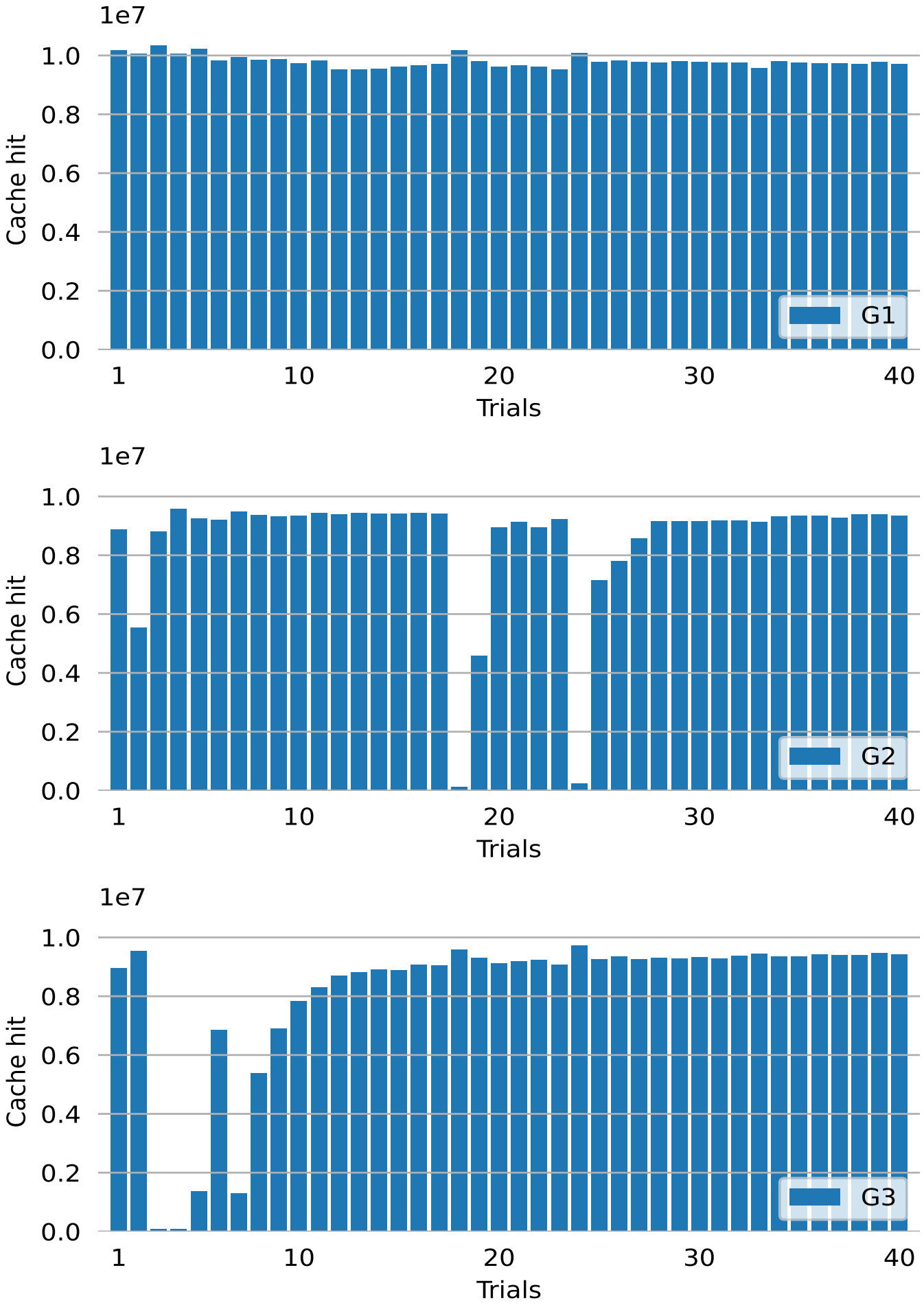

図 21　各試行でのキャッシュヒット数

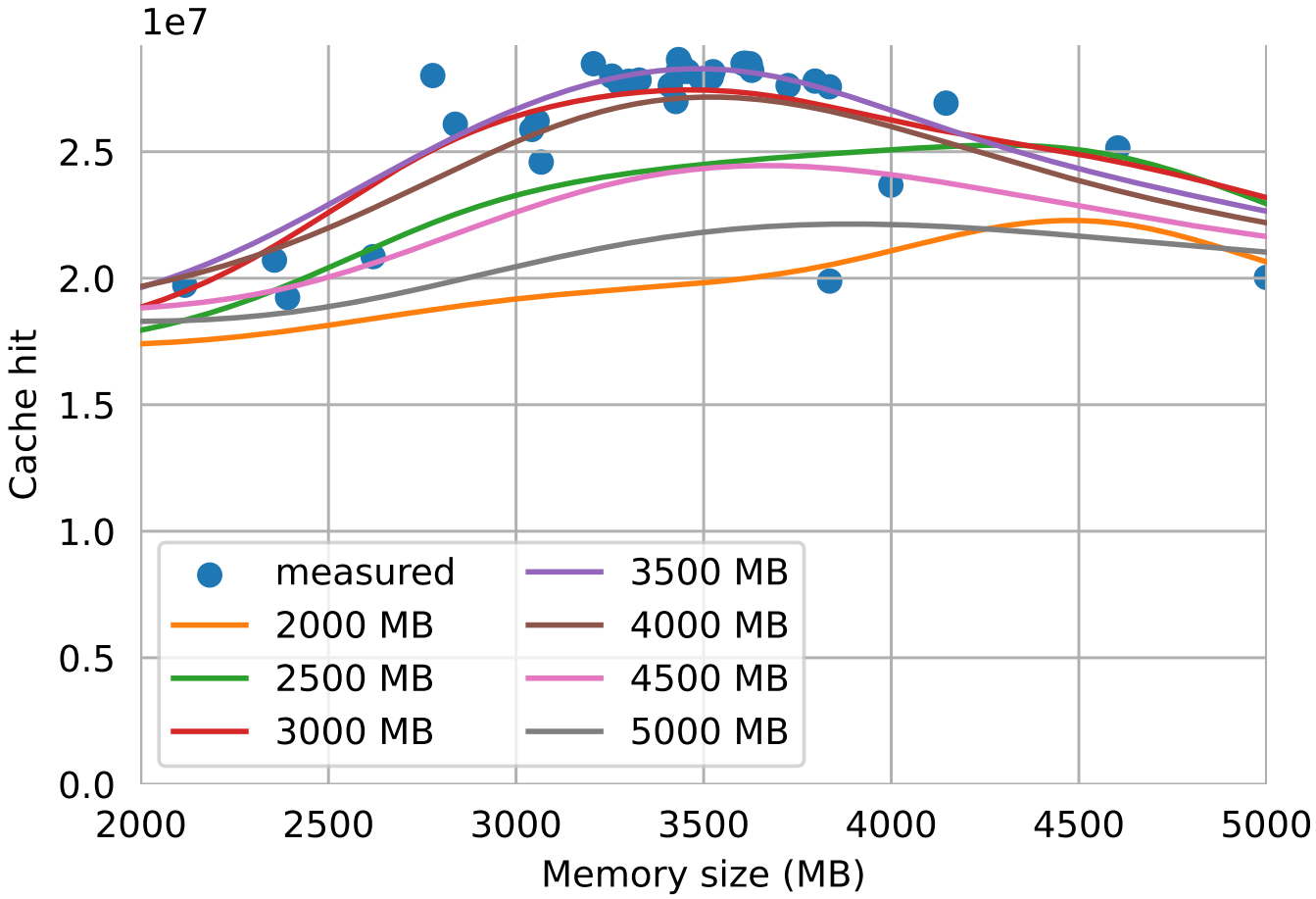

図 22　キャッシュヒット数の実測値と推測値

7　おわりに

本稿では，ディスクキャッシュが有効に機能するよう，仮想計算機へのメモリ割り当て量を設定する手法について述べた．本手法では，ゲスト計算機の動作を監視し，メモリ割り当て量とキャッシュヒット数の関係のモデルを作成する．また，このモデルを用いてキャッシュが有効に機能するメモリ割り当て量を算出し，各ゲスト計算機にそのメモリ割り当て量を適用する．本手法によって，各ゲスト計算機のキャッシュが有効に機能するようなメモリの割り当てが実現される．これにより，限られた物理メモリ資源を効率的に活用し，システム全体の性能を向上させることが可能になる．

謝辞

本研究を遂行するにあたり，終始適切な助言を承り，また幾度なく導いてくださった三好力先生，芝公仁先生に深く感謝します.

参考文献

[1] Yamaguchi, S. and Fujishima, E.: Optimized VM Memory Allocation Based on Monitored Cache Hit Ratio, *Proceedings of the 4th Workshop on Distributed Cloud Computing*, DCC '16, New York, NY, USA, Association for Computing Machinery (2016).

[2] Liu, H., Jin, H., Liao, X., Deng, W., He, B. and Xu, C.-z.: Hotplug or Ballooning: A Comparative Study on Dynamic Memory Management Techniques for Virtual Machines, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 26, No. 5, pp. 1350–1363 (2015).

[3] Schopp, J. H., Fraser, K. and Silbermann, M. J.: Resizing memory with balloons and hotplug, *Proceedings of the Linux Symposium*, Vol. 2, pp. 313–319 (2006).

[4] Lian, Z., Li, Y., Chen, Z., Shan, S., Han, B. and Su, Y.: eBPF-based Working Set Size Estimation in Memory Management, *2022 International Conference on Service Science (ICSS)*, pp. 188–195 (2022).

[5] Jones, S. T., Arpaci-Dusseau, A. C. and Arpaci-Dusseau, R. H.: Geiger: Monitoring the Buffer Cache in a Virtual Machine Environment, *SIGOPS Oper. Syst. Rev.*, Vol. 40, No. 5, p. 14–24 (2006).

[6] Chiang, J.-H., Li, H.-L. and cker Chiueh, T.: Working Set-based Physical Memory Ballooning, *International Conference on Automation and Computing* (2013).

[7] Harby, A. A., Fahmy, S. F. and Amin, A. F.: More Accurate Estimation of Working Set Size in Virtual Machines, *IEEE Access*, Vol. 7, No. 2019, pp. 94039–94047 (2019).

[8] Rasmussen, C. E. and Williams, C. K. I.: *Gaussian processes for machine learning.*, Adaptive computation and machine learning, MIT Press (2006).

[9] Ax: https://ax.dev/.

[10] Balandat, M., Karrer, B., Jiang, D. R., Daulton, S., Letham, B., Wilson, A. G. and Bakshy, E.: BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization, *Advances in Neural Information Processing Systems 33* (2020).