Vol.2022-OS-154 No.1
2022/3/14

DBMSの問い合わせ計画の最適化のための プランナコストの見積もり手法

芝 公仁[1]

概要：多くの環境でデータベースが使用されており，データベースの性能がシステムの性能を決める重要な要素となっている．データベースを効率的に動作させるためには，使用するデータベース管理システム（DBMS）を適切に設定する必要がある．しかし，DBMS には多くの設定項目があり，それらを適切に設定することは困難である．本稿では，DBMS の設定項目に対して，適切な設定値を探索する手法について述べる．本手法は，DBMS の問い合わせ計画に関する設定項目であるプランナコスト定数の設定値の最適化を行う．実際に DBMS を動作させ，設定値と性能の関係のモデルを生成し，それをもとに最適な設定値の探索を行う．このとき，関連する複数の設定項目に対して少ない試行回数で最適値を得られるよう，ベイズ最適化を用いている．本手法により，データベースを，それが動作する環境に応じて，効率的に動作させることが可能になる．

1. はじめに

現在，インターネット上で様々なサービスが提供されており，その多くでデータベースが使用されている．また，小規模のアプリケーションでも，そのデータを保存するためにデータベースを使用することがある．このように様々な環境でデータベースが使用されており，データベースがシステム全体の性能を決める要因となることも多い．データベースを適切に運用するためには，データベース管理システム（DBMS）の設定が重要となる．

DBMS を効率良く動作させるには，その設定を動作する環境に応じて適切に行う必要がある．しかし，DBMS の設定項目は非常に多く，また，設定項目間に関連があり，適切に設定するには多くのコストが必要である．さらに，データベース管理者にとって，意味やその効果が分かりづらく，実際には設定することが難しいような設定項目もある．最適な設定を行うことは専門家でも難しく，一定のガイドラインに沿って設定することや，設定をサポートするツールを用いて設定を行うなど，一般的な環境で有用な設定のみを行うことも多い．

本稿では，DBMS の設定項目に対して適切な設定値を探索する手法について述べる．また，本手法に基づいて DBMS の設定を行うシステムの構成とその動作についても述べる．提案システムは，次の 3 点を実現する．

- プランナコスト定数の最適化を行う．
- 設定値と性能のモデルを生成し，環境に応じた最適値を探索する．
- 設定値の探索や DBMS の設定を自動化する．

本システムでは，特に，DBMS のひとつである PostgreSQL[1] のプランナコスト定数の見積もりを行う．プランナコスト定数は，DBMS がクエリをどのように処理するかの問い合わせの実行計画を作成する際に使用されるものである．この設定を適切に行うことによって，クエリの処理に最適な問い合わせ計画が作成されるため，効率的にクエリを処理することが可能になる．

提案手法では，DBMS の実際の動作を観測し，設定値と性能の関係をガウス過程回帰によってモデル化する [2]．また，作成したモデルをもとに最適な設定値の探索を行う．これによって，動作環境，データベースの構成，アプリケーションからのクエリに対応した最適な設定値を探索することができる．

提案システムは，このような最適値探索の処理や DBMS の操作・設定を行う機能を持つ．DBMS の設定の最適化では，データベース管理者が，ガイドラインや経験をもとに様々な設定で試行を繰り返し，設定値の探索を行うことが多い．提案システムを使用することによって，このような作業を自動化し，低コストで DBMS を効率的に動作させる設定を行うことが可能になる．

以下，本稿では，2 章で関連研究，3 章で提案システムの構成，4 章で DBMS 設定の最適化の手順について述べ

[1] 龍谷大学
Ryukoku University

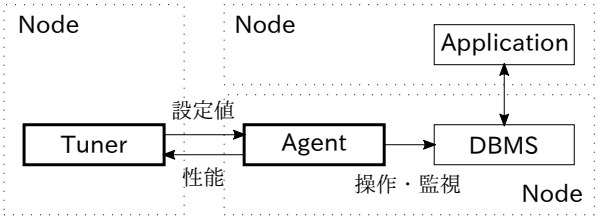

図 1 システムの概要

る．また，5 章で，提案システムを用いた最適化の効果について評価する．

2. 関連研究

多くの場合，データベースの最適化は，データベース管理者によって経験をもとに行われる．この作業は難しく，設定方法のガイドラインなどがあるが，バッファサイズの設定など，一部の設定項目に限られている．これらは，多くのシステムについて共通して設定するとよいとされているものであり，個々の環境固有の設定に関する手法は示されていない．また，設定を支援するツール [3] もあるが，DBMS が動作するハードウェア環境の情報を使用するものであり，データベースやクエリの構成を十分に反映させた設定を行うことはできない．特に，本稿で扱うプランナコスト定数の設定に関しては，指針となる方法やツールは整備されていない．

これまでに，機械学習の技術を用いて DBMS の設定を最適化するいくつかの手法が提案されてきている [4], [5], [6], [7]．これらの手法では，各設定項目の設定値と，その設定での DBMS の性能の関係を数理的に解析する．様々な設定で DBMS を動作させ，それぞれで性能を計測し，ガウス過程回帰や強化学習によって，設定値を入力，性能を出力とするモデルを作成する．また，作成したモデルをもとに最適な設定値の算出を行う．このように DBMS を動作させ性能を計測することによって，動作する環境だけでなく実際のデータベースやクエリの構成を，設定値の最適化に反映させることができる．提案手法も同様のものであり，ベイズ最適化を用いることによって，少ない試行回数で最適な設定値を探索できるようにしている．

3. システムの構成

提案システムの概要を図 1 に示す．提案システムは，DBMS とそれを使用するアプリケーションを対象として動作する．提案システムでは，次の 2 個のコンポーネントが協調して DBMS の設定の最適化を行う．

- Tuner: 設定値の探索を行う．
- Agent: DBMS の操作を行う．

Tuner は，提案のシステムの中核として動作し，DBMS の動作をもとに最適な設定値の探索を行う．Agent は，Tuner からの要求を受け，DBMS の操作や監視を行う．設定を変更するだけではなく，DBMS の起動や終了も行うため，DBMS と同一のノードで動作する．

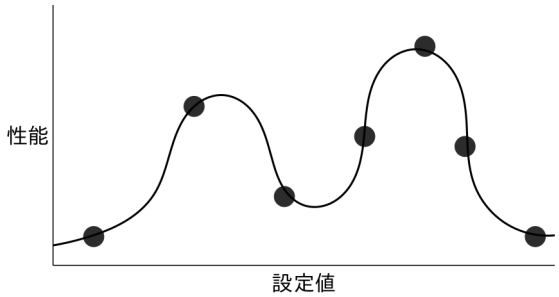

図 2 設定値と性能の関係

Tuner は DBMS の動作を取得しその情報をもとに設定値と性能の関係のモデルを作成する．図 2 は，設定項目が 1 個のときの設定値と性能の関係の例を簡易的に表したものである．グラフの横軸は設定項目の設定値，縦軸は性能である．Tuner は，設定すべき設定値を算出し，その設定で DBMS を動作させ，実際の性能を計測する．このような試行を繰り返し行い，それをもとにガウス過程回帰を用いて設定値と性能の関係を算出する．図 2 では，7 個の点が実測された性能で，曲線が算出された関係から推測される性能である．

このように設定値は連続して変化する量的なデータであり，バッファサイズなどの設定項目がこれに該当する．一方，複数のデータ形式やアルゴリズムから使用するものを 1 個選択するといった設定は該当しない．

Tuner は，試行を繰り返すことによって，設定値と性能の関係のモデルを生成する．このとき，Tuner は，次に試行する設定の設定値をベイズ最適化で算出する．ベイズ最適化を用いることにより，最適値となる可能性の高い設定値を得ることができる．そのため，少ない試行回数で，精度の高いモデルを作成し，最適値を発見することが可能となる．

本稿では，DBMS である PostgreSQL を対象として，その設定項目であるプランナコスト定数の最適化を行う．プランナコスト定数は，問い合わせ計画作成の際に使用されるものである．アプリケーションからの要求があったとき，そのクエリを処理する手順には複数の候補がある．DBMS は，それぞれの手順について，プランナコスト定数をもとに，必要なコストを推測する．推測されたコストをもとに，最適な問い合わせ計画が作成され，それに従い実際の処理が行われる．そのため，アプリケーションからのクエリが同一であってもプランナコスト定数の違いによって，クエリをどのように処理するかが変化する．その結果，アプリケーションからの要求を処理する時間が変化し，それが DBMS の性能として現れる．

したがって，DBMS でアプリケーションからの要求を効率的に処理するには，プランナコスト定数を適切に設定す

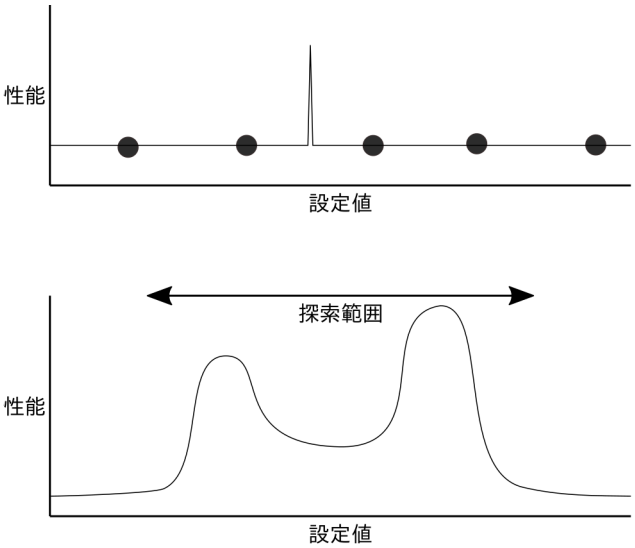

図 3 設定値の探索範囲

ることが重要である．プランナコスト定数は，ハードウェアやオペレーティングシステムなど DBMS が動作する環境に加え，データベースの構成やアプリケーションの動作によって最適値が変わる．提案手法では，DBMS やアプリケーションを動作させ，データベースの性能を実測することによって，実際の動作での設定最適化を実現している．

4. 設定の最適化

提案システムを用いた DBMS の設定の最適化の手順について述べる．本章で最適化の対象とする PostgreSQL には多くの設定項目があるが，ここでは，プランナコスト定数を扱う．プランナコスト定数の設定項目には，次の 7 個がある．

- seq_page_cost
  シーケンシャルなディスクページ取り出しのコスト
- random_page_cost
  非シーケンシャル的なディスクページ取り出しのコスト
- cpu_tuple_cost
  問い合わせ時のそれぞれの行の処理コスト
- cpu_index_tuple_cost
  インデックス走査時のそれぞれのインデックス行の処理コスト
- cpu_operator_cost
  問い合わせ時に実行される各演算子や関数の処理コスト
- parallel_setup_cost
  パラレルワーカープロセスを起動するためのコスト
- parallel_tuple_cost
  あるパラレルワーカープロセスから，1 行を他のプロセスに転送するためのコスト

これら 7 個の設定項目は，それぞれ実数の設定値を持ち，相対的に扱われる．本稿での設定値の最適化では，seq_page_cost を 1.0 に固定し，これ以外の 6 個の設定値を変更する．seq_page_cost と random_page_cost は，ディスクに対するシーケンシャルアクセスとランダムアクセスの性能差を指定するものある．実際の動作では，オペレーティングシステムのキャッシュが処理速度に影響する．したがって，ディスクの性能のみで最適値が決まることはなく，データベースや他のプロセスの構成によって最適値が変化する．そのため，実際に DBMS を動作させて最適値の探索を行う必要がある．同様のことは，seq_page_cost と cpu_tuple_cost のような，ディスクと CPU の処理速度の関係を指定する設定項目についても言える．

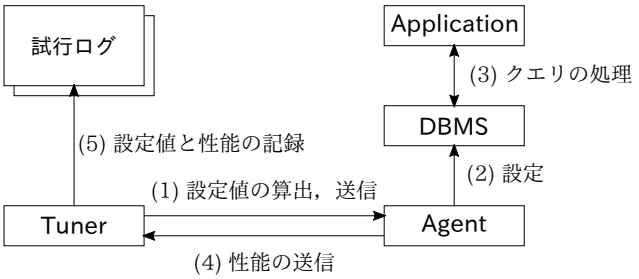

図 4 設定値の最適化

プランナコスト定数の各設定項目には，0 から 1.79769e+308 までの実数値を設定可能である．これは非常に範囲が広いものであるが，図 3 の上のグラフが示すように，多くの場合，設定可能な範囲の中で性能に変化があるのは限られた一部である．設定可能な範囲全てを探索範囲とすると，最適化の試行において，性能に変化がある最適値の周辺が選択される可能性が低い．そのため，設定値と性能の関係のモデルを適切に構築することができず，探索を有効に行うことができない．

探索の範囲は，図 3 の下のグラフが示すように，次の 2 点を満たす必要がある．

(a) 最適値を含む範囲
(b) 設定値の変化で性能が変化する範囲

(a) は範囲を広く，(b) は範囲を狭くすることに繋がるが，探索を有効に行うためには，両方を満たす必要がある．設定項目の意味を考慮し実際に実験を行って適切な範囲を決める必要があるが，通常は，既定の設定値を基準にして決めることができる．

提案システムは，指定された探索範囲内で設定値を変化させ，そのときの DBMS の性能を調べることで，最適な設定値の探索を行う．探索は，図 4 に示すように，以下の処理を繰り返すことで実現される．

スループットやレイテンシ

(1) Tuner が設定値を算出し，それを Agent に送る．
(2) Agent が受け取った設定値を，DBMS に設定する．
(3) Agent による設定で DBMS が動作し，アプリケーションからの要求を処理する．

(4) Agent が DBMS の性能を取得し，それを DBMS に送る．
(5) Tuner が受け取った DBMS の性能をログに保存する．

最適化を行う際，最初に Tuner は，ソボル列で設定値を生成し，その設定での性能としてスループットやレイテンシを調べる．これを何回か繰り返し，性能のログを作成する．このログにより，設定値の探索空間全体の概要のモデルを作成できるようになる．その後は，ベイズ最適化により次に試行すべき設定値を算出する．また，その設定値での試行で性能を調べ，結果をログに追加する．これにより，より精度の高いモデルを作成できるようになる．

5. 評価

本章では，提案システムによる DBMS の最適化を評価するために行った実験について述べる．実験の環境を表 1 に示す．実験では 1 台の物理計算機と当該計算機上で動作する仮想計算機 1 台を使用して行った．DBMS と Agent を仮想計算機で動作させ，Tuner と DBMS を使用するアプリケーションを物理計算機上で動作させた．

データベースのデータのファイルの実態は物理計算機のハードディスク内にあり，物理計算機ではこれをキャッシュしない．本章で行う各実験では，データベースのファイルのサイズは，約 2 GB である．仮想計算機の物理メモリのサイズが 2 GB であるため，このファイルすべてがキャッシュされることはない．

Tuner の実装では，ベイズ最適化に Ax[8], [9] を用いている．DBMS を使用するアプリケーションとしてベンチマークフレームワークである BenchBase[10], [11] を用いた．また，DBMS の性能としてスループットを用いた．

5.1 設定値の変化

本システムによって，DBMS の設定値がどのように変化するかを調べる実験を行った．本実験では，DBMS を使用するアプリケーションとして BenchBase の SEATS を使用した．図 5 は，最適化の各試行においてどのような設定値が用いられたかを示している．グラフの横軸は試行回数であり，縦軸は各設定項目の設定値である．なお，設定値の探索範囲は，グラフの縦軸の範囲と同一である．

この実験の最適化においては，初めの試行の 12 回はソボル列で生成された設定値を用いて行われている．実際に，12 回目までの試行では，すべての設定項目について，設定値が広い範囲で変化している．これらの試行によって，探索範囲全体の性能の概略が取得される．

13 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．13 回目以降の試行において，random_page_cost や parallel_setup_cost では，設定値は変化が少なく偏った値となっている．これは，12 回目までの試行で，性能が向上する設定値の範囲を絞り込めたためであると考えられる．13 回目以降も，設定値が大きく変化する設定項目は，最適値の探索が広い範囲で続いていると考えられる．35 回目以降の試行では，どの設定項目も設定値の変化が減少している．このように，6 個の設定項目に対しても，少ない試行回数で性能が向上する設定値を見つけることができている．

表 1 実験環境

物理計算機	
CPU	Intel Core i7–4770 3.40 GHz
メモリ	8 GB
OS	Debian–11.0

仮想計算機	
CPU	1 コア
メモリ	2 GB
OS	Debian–11.0
DBMS	PostgreSQL–13.4

5.2 性能の実測値と推測値

前節で述べた実験での 40 回目の試行を終えたときの，性能の全実測値と，生成された設定値と性能のモデルから得られる性能の推測値を図 6 に示す．グラフの横軸は各設定項目の設定値，縦軸はスループットである．グラフ内でプロットされている点は各試行において実測された値である．6 個のグラフそれぞれで 6 個の設定値とスループットの関係を示しているが，同一の試行は同一の色でプロットされている．グラフの曲線はモデルから推測されたスループットである．このとき，推測値は，当該設定項目以外の設定項目の設定値を，推測される最適値としたものである．例えば，random_page_cost のグラフでは，random_page_cost 以外の設定項目の設定値は最適化によって求められたそれぞれの最適値を用いている．

すべての設定項目について，設定値を変更して試行すると性能が変化している．また，それをもとに生成されたモデルによる推測値も，設定値によって性能が変化することを示している．プランナコスト定数については，既定値から安易に変更することは避けるよう推奨されているが，環境に応じて適切に設定を行うことは有効であることがわかる．

各試行のために算出された設定値は，性能が良い場合の設定値に近いものが多い．ベイズ最適化の処理では，獲得関数に Expected Improvement を使用している．これにより，性能が向上すると期待される設定値が算出され，性能が高い領域を重点的に試行することができる．そのため，少ない試行回数で，高い性能となる設定値を見つけることができている．

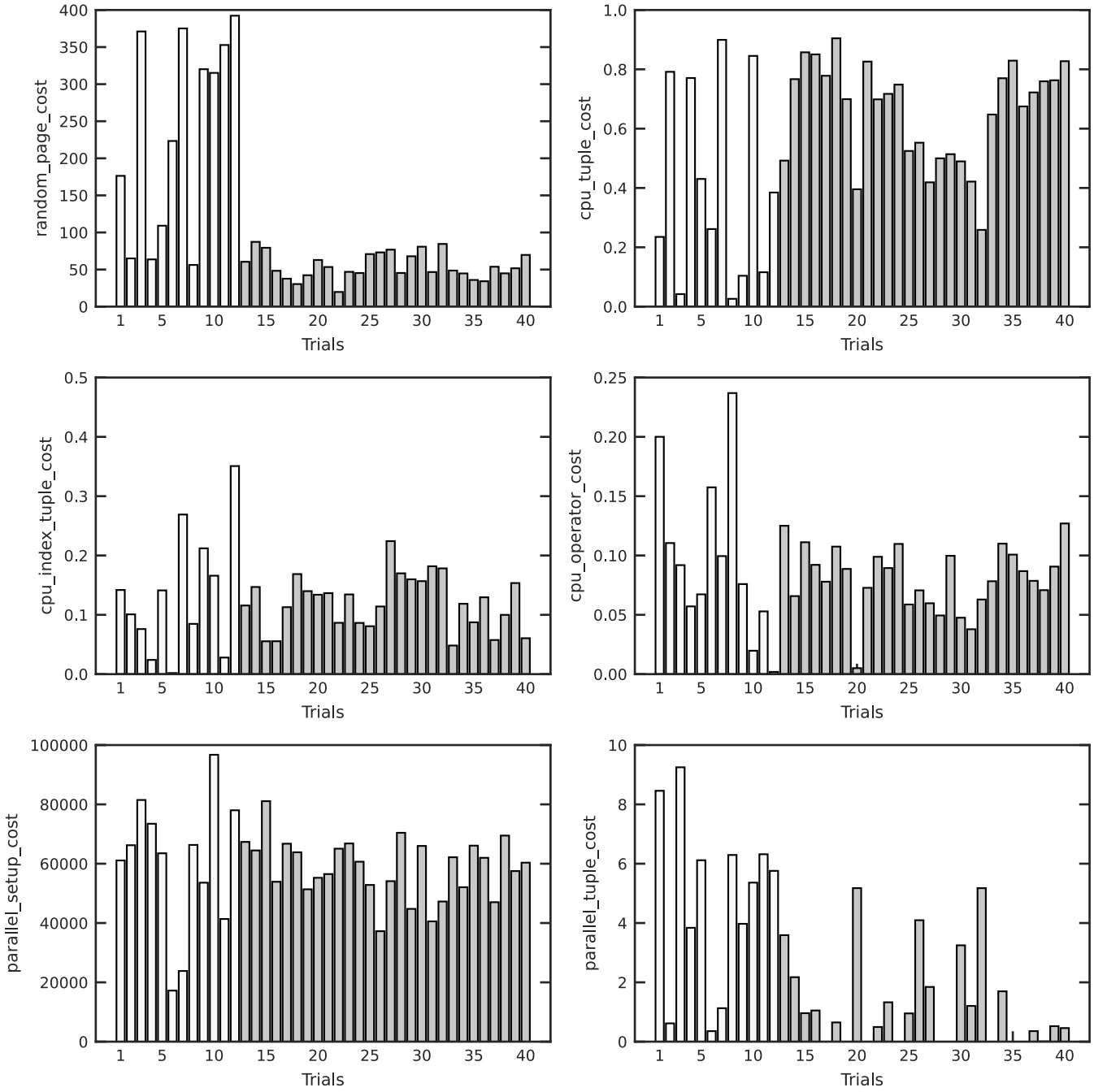

図 5 設定値の変化

5.3 性能の変化

前節で述べた実験において，各施行で DBMS の性能がどのように変化したかを図 7 に示す．グラフの横軸は試行回数，縦軸はスループットである．1 回目から 12 回目までの試行はソボル列で生成された設定値で行われているため，それぞれのスループットは大きく異なるものとなっている．13 回目以降の試行では，最適値に近い設定値が設定され，スループットは安定して高いものとなっている．また，20 回程度の試行で十分に良い設定値が見つかっていることがわかる．このように，設定値と性能のモデルを作成することにより，DBMS が高い性能で動作する設定値を少ない試行回数で見つけることが可能となる．

1 回目から 12 回目までの試行のスループットの平均値は 69.03 である．また，全試行の中で最も性能が高かった試行のスループットは 128.46 である．128.46 / 69.03 = 1.86 倍の性能差であり，プランナコスト定数のみの最適化でも，DBMS の性能向上が可能であることがわかる．

5.4 動作環境への適応

データベースやクエリの構成の違いで，設定の最適値に違いがあるかを確認するため，ベンチマークの種類を変更して実験を行った．使用したベンチマークは，前節で述べ

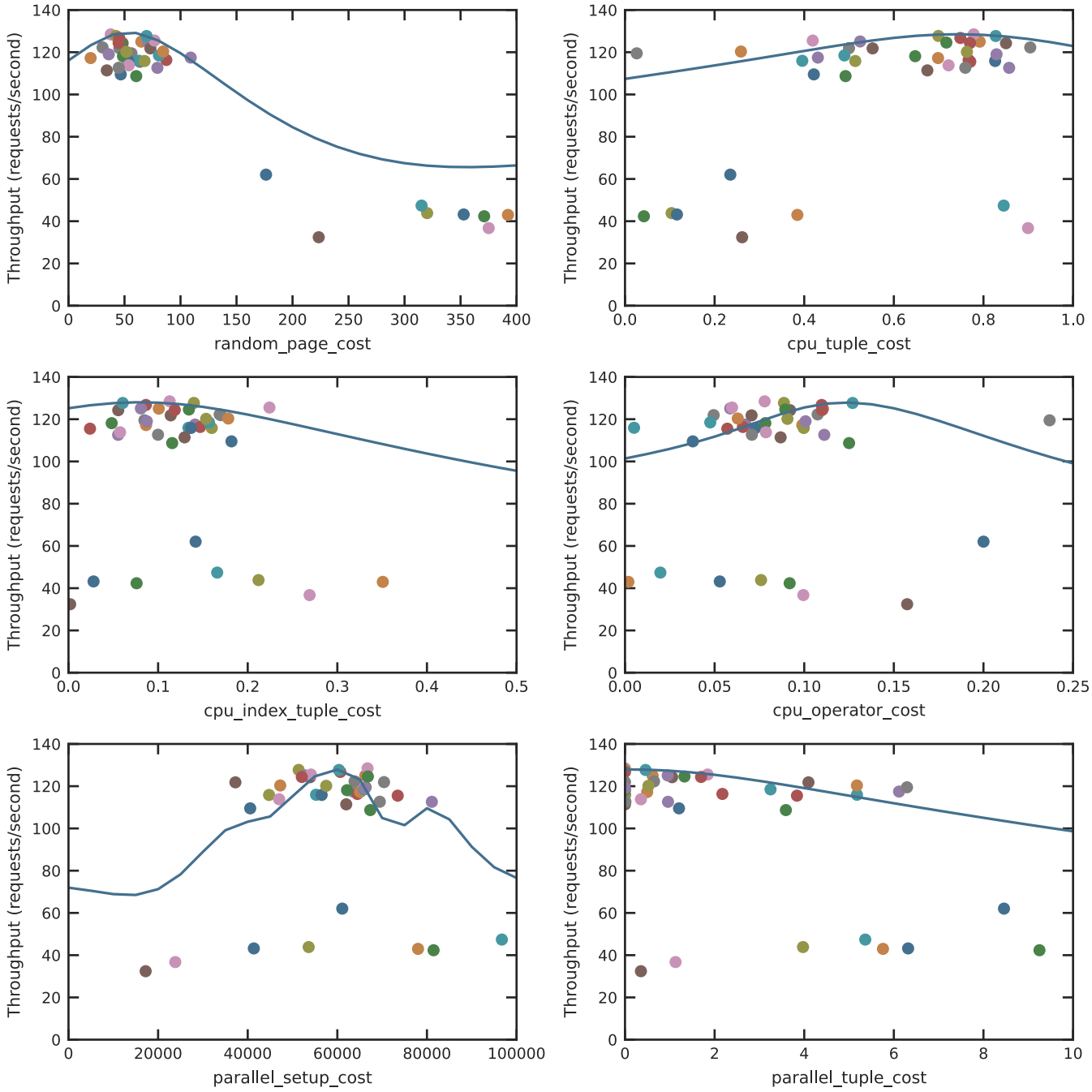

図 6 性能の実測値と推測値

た SEATS に TPC–C，Twitter，Epinions，YCSB を加えた 5 個である．各ベンチマークを用いた場合の，40 回の試行から作成したモデルで得られた最適とされる設定値を表 2 に示す．表の (a) から (f) は，random_page_cost から parallel_tuple_cost までの設定項目である．また，(g) は，1 回目から 12 回目までの試行のスループットの平均値，(h) は，全試行の中で最も性能が高かった試行のスループットである．

ベンチマーク毎に，探索された最適値が大きく異なるものとなっている．DBMS を効率的に動作させるためには，データベースやクエリの構成に応じてプランナコスト定数を設定することが重要であることがわかる．

特に，Epinions では，(a) random_page_cost の最適値が 0 になっている．これは，ディスクへのランダムアクセスのコストが，シーケンシャルアクセスのコストである seq_page_cost の設定値 1 より小さいと見積もっていることになる．一般的には，ランダムアクセスのコストは，シーケンシャルアクセスのものより大きくなる．しかし，この実験での DBMS のクエリ処理においては，オペレーティングシステムが行うキャッシュにより，コストが逆転していた可能性がある．

スループットである (g) と (h) を比較すると，Twitter，

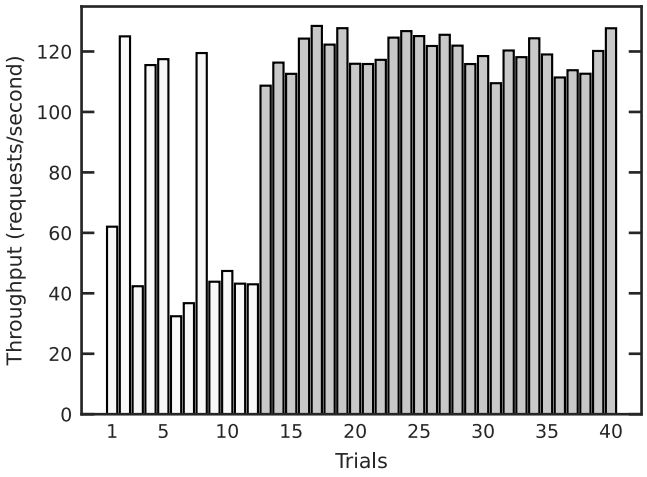

図 7 性能の変化

表 2 探索された最適値とスループット

	SEATS	TPC–C	Twitter	Epinions	YCSB
(a)	69.72	86.51	151.92	0.00	17.33
(b)	0.82747	0.37816	0.68491	0.18486	0.09278
(c)	0.06052	0.41059	0.3308	0.36702	0.15018
(d)	0.12699	0.23849	0.2206	0.15106	0.05909
(e)	60351	31640	6789	80974	2708
(f)	0.4587	4.2525	6.0425	4.3786	3.4249
(g)	69.03	44.34	10.53	8.66	22.93
(h)	128.46	47.96	48.52	19.91	126.73

Epinions，YCSB では，SEATS より最適化の性能向上が大きいことがわかる．一方，TPC–C では，47.96 / 44.34 = 1.08 倍と最適化による性能向上が他のものに比べ小さい．プランナコスト定数が変化しても，処理時間に差が現れるような問い合わせ計画の変化があまり発生しなかったためであると考えられる．

6. おわりに

本稿では，DBMS の設定項目に対して適切な設定値を探索する手法と，本手法に基づき DBMS の最適化を行うシステムについて述べた．提案システムは，プランナコスト定数に関する設定値の最適化を行う．実際に DBMS を動作させ，性能を実測し，その結果をもとに設定値と性能の関係のモデルを生成する．このモデルを用いて最適値の探索を行うことによって，少ない試行回数で性能が向上する設定値を見つけることができる．提案システムを使用することによって，低コストで DBMS を効率的に動作させる設定を行うことが可能になる．

参考文献

[1] PostgreSQL: https://www.postgresql.org/about/.
[2] Williams, C. K. and Rasmussen, C. E.: *Gaussian processes for machine learning*, Vol. 2, No. 3, MIT press Cambridge, MA (2006).
[3] PGTune: https://pgtune.leopard.in.ua/.
[4] Duan, S., Thummala, V. and Babu, S.: Tuning Database Configuration Parameters with iTuned, *PVLDB*, Vol. 2, pp. 1246–1257 (2009).
[5] Zhang, B., Van Aken, D., Wang, J., Dai, T., Jiang, S., Lao, J., Sheng, S., Pavlo, A. and Gordon, G. J.: A Demonstration of the Ottertune Automatic Database Management System Tuning Service, *Proc. VLDB Endow.*, Vol. 11, No. 12, pp. 1910–1913 (2018).
[6] Zhang, J., Liu, Y., Zhou, K., Li, G., Xiao, Z., Cheng, B., Xing, J., Wang, Y., Cheng, T., Liu, L., Ran, M. and Li, Z.: An End-to-End Automatic Cloud Database Tuning System Using Deep Reinforcement Learning, *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, New York, NY, USA, ACM, pp. 415–432 (2019).
[7] Ishihara, Y. and Shiba, M.: Dynamic Configuration Tuning of Working Database Management Systems, *2020 IEEE 2nd Global Conference on Life Sciences and Technologies (LifeTech)*, pp. 393–397 (online), DOI: 10.1109/LifeTech48969.2020.1570619232 (2020).
[8] Ax: https://ax.dev/.
[9] 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*, (online), available from ⟨http://arxiv.org/abs/1910.06403⟩ (2020).
[10] Difallah, D. E., Pavlo, A., Curino, C. and Cudré-Mauroux, P.: OLTP-Bench: An Extensible Testbed for benchmarking Relational Databases, *PVLDB*, Vol. 7, No. 4, pp. 277–288 (2013).
[11] BenchBease: https://github.com/cmu-db/benchbase.