4L-04

時刻変化するワークロードのための NoSQL スキーマのオフライン最適化

涌田 悠佑 †
大阪大学大学院情報科学研究科 †

Michael Mior‡
Rochester Institute of Technology‡

善明 晃由 §
株式会社サイバーエージェント §

佐々木 勇和 ¶
大阪大学大学院情報科学研究科 ¶

鬼塚 真 ‖
大阪大学大学院情報科学研究科 ‖

1 はじめに

NoSQL データベースにおけるスキーマ設計は性能を引き出すために重要な技術分野である．複雑なワークロードに対して最適なスキーマを手動で設計することは困難なため静的なスキーマの最適化技術が提案されている．しかし，時刻変化するワークロードではスキーマ設計も動的に変更しなければデータベースの性能を十分に発揮できない．本稿では，時刻変化するワークロードに対応した NoSQL データベースのスキーマ最適化手法を提案する．提案手法は生成したクエリプラン候補から，使用される見込みの大きいマイグレーションプラン候補を列挙する．そして，クエリプランとマイグレーションプランの依存関係を整数線形計画問題として定式化し，総実行コストを最小化することで全時刻において最適なマイグレーションプランを推薦する．評価実験の結果，時刻変化するシンプルなワークロードにおいて静的な最適化を上回る性能を達成した．

2 既存の課題

NoSQL データベース[*1]は RDB と比べてシンプルなデータ構造を使用しており，大規模なアプリケーションにおいて高い分散処理性能を達成している．また，Extensible Record Store のプロダクトによって詳細な定義は異なるが RDB のテーブル相当のデータ構造として Column Family (以下，CF) を提供している．

時刻変化するワークロードにおける NoSQL データベースのスキーマ設計には主に 3 つの課題がある．1 つ目の課題はスキーマの正規化の度合いにおける Put, Get の性能のトレードオフである．スキーマを正規化すると Get はクライアント側で CF をジョインするため応答時間が増加するが，Put の更新対象の CF は減少する．2 つ目の課題は各時刻のワークロードの実行コストとマイグレーションの実行コストのトレードオフである．ワークロードの実行コストを低減するためにはワークロードの変化に応じてスキーマをマイグレーションすることが有効である．しかし，多くのマイグレーションを実行するとデータベースの他の処理を圧迫する可能性がある．3 つ目の課題は，クエリプランとマイグレーションプランが互いに依存している点である．各時刻において使用可能なマイグレーションプラン候補は各時刻のスキーマとクエリプランによって制約される．同様に，各時刻のクエリプラン候補もその時刻のマイグレーションプランに制約される．

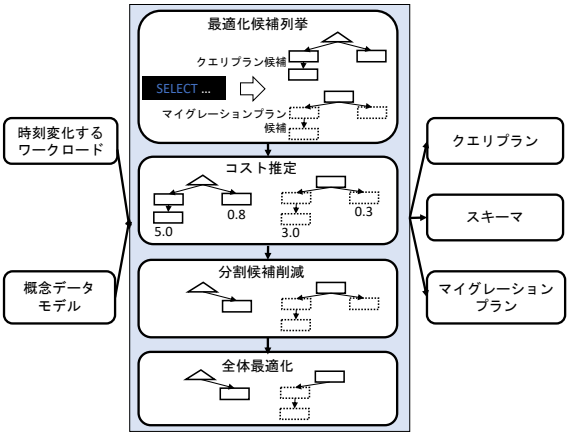

図 1 提案手法の概要図

静的なワークロードに対する NoSQL データベースのスキーマ設計手法として NoSQL Schema Evaluator (NoSE) [4] が提案されている．NoSE は整数線形計画問題 (ILP) を用いてスキーマを最適化するが，時刻変化するワークロードは対象としていない．また，Hillenbrand ら [2] は NoSQL データベースにおいてスキーマが変化した際のデータ移行手法を自動で選択するフレームワークを提案している．ただし，マイグレーションは入力として想定しておりその最適化は対象としていない．

3 提案手法

本稿では時刻変化するワークロードにおける各時刻のスキーマ設計とマイグレーションプランをオフライン最適化するフレームワークを提案する．このフレームワークはワークロードの実行コストとマイグレーションプランの実行コストの総和を ILP を用いて最小化する．これにより，Put と Get のトレードオフ，ワークロードの実行コストとマイグレーションの実行コストのトレードオフ，

Offline NoSQL schema optimization for time-depend workload
† Yusuke Wakuta, Osaka University
‡ Michael Mior, Rochester Institute of Technology
§ Teruyoshi Zenmyo, CyberAgent, Inc.
¶ Yuya Sasaki, Osaka University
‖ Makoto Onizuka, Osaka University
*1 本稿では Extensible Record Store[1] を用いる．

グループ1	SELECT * FROM users WHERE users.id=? SELECT * FROM users WHERE users.firstname=? INSERT INTO users SET id=?, ...
グループ2	SELECT * FROM items WHERE items.id=? SELECT * FROM items WHERE items.name=? INSERT INTO items SET id=?, ...

図 2　ワークロード

プラン間の依存関係を踏まえたスキーマ設計を行う．本フレームワークは以下の処理手順でスキーマを設計する．

1. 最適化候補の列挙: Put, Get に対して CF, クエリプラン，マイグレーションプランの候補を列挙する．マイグレーションプランの列挙ではクエリプランを再利用する手法と，マイグレーション用に新たに生成した Get のクエリプランを列挙する手法を用いる．
2. コスト推定: 各処理の実行コストを属性のメタデータから推定する．マイグレーションプランのコスト推定では，既存の CF からデータを取得するコスト，新たな CF にデータをロードするコスト，マイグレーション中の CF を更新するコストを評価する．
3. 分割候補削減: 全体最適化で用いる ILP の部分問題を予め解き，全体最適化の候補を削減する．これにより時刻数の多いワークロードを最適化する際の実行時間の増加を低減する．全体最適化で用いる最適化問題を再帰的に部分問題に分割する．分割処理では親問題の最適化結果を子問題に制約として引き継ぐことで大域的な頻度変化を部分問題にも反映する．どの部分問題でも推薦されない CF 候補は全体最適化においても推薦される見込みが低いため全体最適化の候補から排除する．
4. 全体最適化: ワークロード実行コストとマイグレーションコストの総和を最小化する．定式化では，CF 候補に決定変数を割り当てる．さらに，クエリプラン候補とマイグレーションプラン候補において CF 候補を使用する毎に新たに決定変数を割り当てる．そして，最適化によって各 Get にクエリプランが推薦され，実行するマイグレーションに応じてマイグレーションプランが推薦されるように制約条件を設ける．さらに，各時刻のクエリプランとマイグレーションプラン間の依存関係を制約条件として定式化する．

以上の手順により時刻変化するワークロードにおいて総実行コストが最小な各時刻のスキーマとマイグレーションプランを推薦する．

4　評価実験

提案手法の性能を評価するために図 2 に示したシンプルなワークロードにおいて応答時間を評価する．グループ 1 とグループ 2 の頻度割合は時刻 4 までは 1 : 9 であり，時刻 5 以降は 9 : 1 である．さらに，多くの更新処理がある場合を想定して容量制約無しで最適化した場合のストレージサイズから 0.5% 低減した容量制約を設けている．

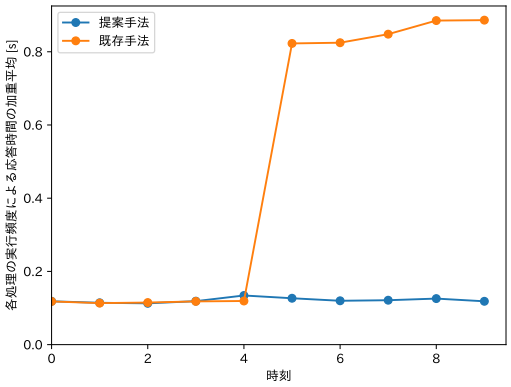

図 3　各処理の実行頻度による応答時間の加重平均

提案手法では，時刻 4 から時刻 5 にかけて 2 つのマイグレーションプランを実行するスキーマを推薦した．また，既存手法として各時刻の頻度の平均値を持つ静的なワークロードに対して NoSE を用いてスキーマを最適化した．Cassandra を用いた評価実験の結果，応答時刻の頻度による加重平均の総和を既存手法に対して 75% 低減することを確認した．図 3 に示した結果から既存手法では頻度変化に追従できず時刻 5 から応答時間の加重平均が増加しているが，提案手法はマイグレーションによって応答時間の加重平均の増加を低減していることを確認した．

5　まとめ

本稿では，時刻変化するワークロードに対して NoSQL データベースのスキーマを設計する推薦フレームワークを提案した．このフレームワークでは，ワークロードの Put, Get の実行コストとマイグレーション処理の実行コストの総和を ILP として最適化する．評価実験を通して，既存手法によるスキーマ設計では性能が低下する場合においても提案手法が高い性能を達成することを確認した．今後の課題として，クエリ頻度の予測手法 [3] を活用することによる頻度変化が未知の場合への対応が考えられる．

6　謝辞

この成果は, 国立研究開発法人新エネルギー・産業技術総合 開発機構（ＮＥＤＯ）の委託業務の結果得られたものです.

参考文献

[1] R. Cattell. Scalable SQL and NoSQL data stores. *SIGMOD Record*, 39:12–27, 2011.

[2] A. Hillenbrand et al. Towards Self-Adapting Data Migration in the Context of Schema Evolution in NoSQL Databases. In *ICDEW*, pages 133–138, 2020.

[3] L. Ma et al. Query-based Workload Forecasting for Self-Driving Database Management Systems. In *SIGMOD*, page 631–645, 2018.

[4] M. Mior et al. NoSE: Schema design for NoSQL applications. *TKDE*, 29:2275–2289, 2017.