統計情報を用いた半構造化データに対する SQL クエリ処理実行の効率化

加藤 千裕† 中村 実† 原田 リリアン†

†株式会社富士通研究所 〒211-8588 神奈川県川崎市中原区上小田中 4-1-1

E-mail: †{kato-chihiro, nminoru, harada.lilian}@fujitsu.jp

あらまし RDBMS による従来の基幹システムのデータの解析に加え，近年，NoSQL データベースに格納された XML や JSON などの半構造化データを SQL により解析する必要性が認識されてきている．しかし，NoSQL の半構造化データにスキーマを定義し固定的なテーブル列にマッピングする従来の方法では, NoSQL の複数の形式のデータを許すという特徴が失われてしまう．この特徴を維持する別の方法として，スキーマを定義せず，半構造化データを要素に分解しないで一つのテーブル列として定義する方法があるが，これは RDBMS のオプティマイザが要素ごとの統計情報を使用することが出来ない．そこで本論文では，NoSQL の半構造化データからあらかじめ統計情報を収集し，スキーマを定義しない SQL クエリが投入されたときに統計情報を反映する新たな手法を提案した．そしてこの手法を評価し，RDBMS のオプティマイザに適切な実行計画を作成させ，効率的なクエリ実行が可能となることを確認した．

キーワード RDBMS，NoSQL，JSON，Foreign Data Wrapper

1. はじめに

データベースシステムはデータ利活用のコア技術として多様な発展を遂げてきた．リレーショナルデータベースシステム（RDBMS）は，スキーマ（データ構造）を一意に定義することで，SQL 言語を用いた複雑な分析処理を実現し，今日に至るまで多くの場面で使われている．更に近年では，センサ・ログデータを，JSON や XML といったスキーマが 1 種類でない半構造化データとして格納する NoSQL データベースも広く使われるようになっている．これは柔軟なスキーマを持つデータを格納することから，スキーマレスなデータベースとも表現される．しかし NoSQL データベースは多くの場合独自に言語を持ち，集計・分析機能もそれぞれ異なるため，分析者の習得コストが大きいという問題がある．そのため，汎用的な一律のインターフェース, 例えば SQL により接続し, RDBMS と NoSQL データベースを統合することは，近年の研究課題の一つとなっている [1] [2].

これを可能とする機構の一つとして，Foreign Data Wrapper（FDW）がある．これは既存の RDBMS から外部データソースに接続し，あたかも自らの内部テーブルのように操作する機構である．RDBMS の高度な分析機能をそのまま利用できることから，我々はこの機構に注目した．

FDW は基本的に，XML の要素や JSON フィールドに接続する場合は，各要素・フィールドをそれぞれ列として指定した外部テーブルを事前に定義する．EnterpriseDB 社が公開している PostgreSQL と MongoDB を接続するための FDW [3]もこの方法を採用している．しかしこの方法は，RDBMS の統計情報取得機構やオプティマイザを十分に使った効率的な SQL 実行が可能となる一方で，固定されたテーブル定義を通じて JSON や XML にアクセスするため, 半構造化データのスキーマレスな特徴が失われてしまう．

```
CREATE FOREIGN TABLE mongo_table ( doc JSON)
SERVER mongo_server ... ;                          ... (a)

SELECT (doc->>'id')::integer FROM mongo_table;     ... (b)
```

図 1 スキーマレスな問合せ

スキーマレスな特徴を保った FDW による接続としては，図 1 のように一つの JSON や XML データ全体を単一のテキスト列として定義した外部テーブルを作成する，という方法が考えられる．この時，XML の要素や JSON フィールドは, RDBMS 内でオペレータや関数を用い, 動的に JSON や XML を解析することで取得する. しかし, この方法は XML 全体を読み込むため，NoSQL データベースから RDBMS への大きな転送コストを必要とする上, データをリアルタイムにパースし，処理する必要があり，大きなクエリ処理性能の低下を引き起こす原因となっていた．

これを解決すべく，我々の以前の研究 [4]では，JSON や XML データを単一のテキストとして定義した場合においても，問合せ中のオペレータや関数から操作対象となる要素を絞り込むことで転送データをフィルタリングし，処理時間を削減する効率的な読み込み機構を提案した．しかし 2 章冒頭で説明するように，

この方法は多段のテーブル結合を含む問合せの場合に改良の余地が残されていた.

そこで本論文では，以前の手法を改良し，XML や JSON のデータソースからスキーマレスなデータの統計情報を事前収集し，適用する機構を提案する．これにより，RDBMS のオプティマイザにより適した実行計画を作成させる．ただしこの機構は現在まだ実装中であり，本論文では提案手法がデータの柔軟さと性能のトレードオフなしに統合が出来ることを予備実験により明らかにする.

2. 提案手法

我々は，XML または JSON データソースと，広く使われている OSS の RDBMS である PostgreSQL とのデータ統合技術について，以前から研究を進めてきた．そしてその過程で MongoDB 内の半構造な JSON データに対して PostgreSQL から接続する効率的な手法について提案，発表した [4]．図 1(a)はこの手法において，JSON データを，どのようにして一つの文字列として PostgreSQL の外部テーブルに定義するかを示している.

本論文では，この以前の手法をベースとし，NoSQL データの統計情報を収集することで最適な実行計画を生成する新たな機構を提案する.

2.1. スキーマレスなデータアクセス

従来，PostgreSQL からスキーマレスに MongoDB 内のドキュメントに問合せするためには，まず各 JSON ドキュメント全体を MongoDB から読み込み，それから PostgreSQL の JSON オペレータを使って処理をしていた．この方法は PostgreSQL が JSON ドキュメント全体を読み込むため，必要ない JSON フィールドも転送データに含まれており，非効率であった．例えば図 1(b)では，“id”列のみが選ばれているにも関わらず，JSON ドキュメント全体が MongoDB から PostgreSQL へと転送されていた.

以前の我々の手法は，MongoDB から PostgreSQL へのデータ転送を最小化するため,まず問合せを解析し，問合せ内でオペレータや関数によって参照されている JSON フィールドを抽出する．そして，これを元に MongoDB から JSON データをフェッチする際に，必要なフィールドだけを転送するようにプッシュダウンを行い,取得するフィールドを制限する.それに加えて，我々の手法は BSON [5]と呼ばれる MongoDB 用のバイナリ形式への最適化も行っている.MongoDB に格納される JSON はパースされ，独自の BSON 形式に変換される．BSON 形式は，フィールドが数値の場合には文字列から整数型や浮動小数点型にエンコードして記録

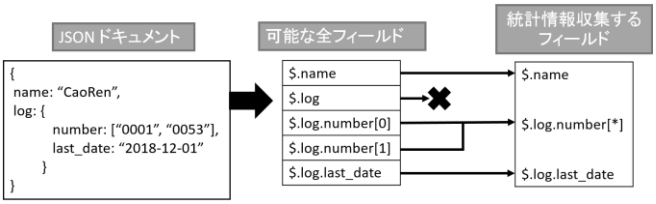

図 2　統計情報収集フィールドの選択

している．我々の手法は JSON オペレータがフィールドの値を変換する型を解析しておき，直接必要な値を BSON データから展開し，処理する．以上により，必要な JSON フィールドの効率的な PostgreSQL への転送・処理を実現した.

しかし,この以前の手法において PostgreSQL のオプティマイザは,MongoDB の持つ JSON データの各フィールドをテーブル列として認識することができない．そして，PostgreSQL は外部テーブルのテーブル列を定義しない限り，統計情報（最頻値やデータのヒストグラム等）を設定できない．この統計情報の不足は間違ったコスト計算と非効率な実行計画の選択を引き起こし，処理能力を低下させる．SQL の問合せが複雑になればなるほど，実行計画の非効率性は増加する.

2.2. 問合せ最適化のための統計情報収集

以前の手法における各 JSON フィールドの統計情報不足の問題を解決するため，提案手法では，NoSQL に蓄積されたデータを外部テーブルの定義がない状態で解析し，属性やフィールドの統計情報を収集し“統計情報格納テーブル”へと保存する.そして,PostgreSQL に MongoDB に対する図 1(b)のような SQL の問合せが投入されるたびに，問合せ内の情報から仮想的な列を決定し“問合せ固有スキーマ”を作成する．そして問合せ固有スキーマの各仮想列に沿って，統計情報格納テーブルから対応する統計情報を取りこむことで，PostgreSQL のオプティマイザを有効化する.

しかし，この提案手法では，MongoDB のドキュメントが入れ子構造や配列によって非常に多くのフィールドを含んでいるため，統計情報格納テーブルが不必要に巨大化する危険性がある.これを回避するため,我々は以下に示す統計情報収集基準を用いて統計情報格納テーブルをコンパクト化する.

(1) 入れ子構造は葉にあたる要素だけを用いる

(2) 配列要素は同じパスと同じ型を持つ場合，統計情報を統合する

(1)は入れ子構造の JSON 内で重要な値は葉の位置にあるという仮定に基づいている．そして(2)は，同じ

```
{
  “ipAddress”: “xxx.xxx.xxx.xxx”,
  “dataSize”: “11111”,
   …
  “userId”; “zzzz”,
   “data”: [
       {“id”: “xx”, “value”: …}, {“id”: “xx”, “value”: …}, … {“id”: “xx”, “value”: …}
   ]
}
```

図 3　収集基準の削減検証用 JSON データ

表 1　統計情報取得基準によるフィールド削減

統計情報取得基準	取得フィールド数
無し	61,829
有り	116

パスと型を持つ配列要素は同じドメインに属しているとの仮定に基づく．図 2 は我々の統計情報収集基準の例を表している．例に示す JSON は 5 個のフィールド“$.name”，“$.log”，“$.log.number[0]”，“$.log.number[1]”，“$.log.last_date”を持つ．まず基準(1)により，葉の要素ではない”$.log”は統計情報の収集対象から外される．そして基準(2)により，“$.log.number[0]”と“$.log.number[1]”は“$.log.number[*]”として纏められ，一つの列として統計情報の収集対象となる．

この基準による統計情報収集フィールドの削減率について，実際に社内で保持している JSON データを用いて検証を行った．これは空間の状況を察知するセンサーデータであり，図 3 に示されるように，基本的な情報を表すフィールドと実際のデータを表す data フィールドの配列から構成される．

このデータに統計情報収集基準を適用したところ，表 1 に示す結果が得られた．基準を用いないと列数は 61,829 列となった．一方，基準を用いた場合は 116 列と，1/500 程度に圧縮できた．これは，データ中の data 配列に 1,000 以上の要素を持つものがあり，この中に入った配列要素を一つ一つ数えたことにより，数が非常に増加したためである．以上のことから，実際の JSON データに対して選定基準は統計情報の肥大化を防ぐのに有効であると言える．以降，提案手法はこの基準を用いて統計情報を取得し，オプティマイザに適用する．

2.3. 手法の構成図

図 4 (a)は 2.1 節で説明した，以前の手法（PREV）の機構を表したものである．これはユーザから問合せを受け取る PostgreSQL の Master ノードが外部の MongoDB 内の JSON データに対して直接接続する．対して，2.2 節で提案した改良手法（ENH）は図 4(b)に

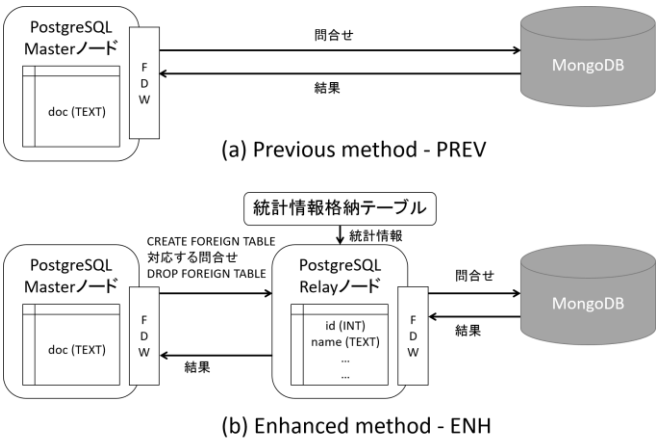

図 4　以前の手法と提案手法の概要

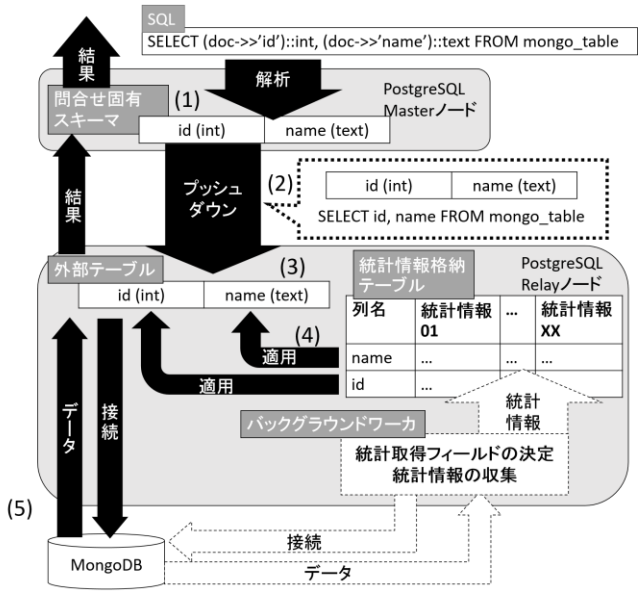

図 5　提案手法の動作

表される．我々は PostgreSQL の Master ノードと外部の MongoDB の間に， Relay ノードを挿入する．このような中間ノードの導入はノード間の通信オーバーヘッドを生むが，この機構は複雑なデータ分析の問合せを想定しているため，転送されるデータは大半が集約結果であり，オーバーヘッドは非常に小さいと考えられる．この仮説は 3 章の実験において確認する．

ENH の実際の動作を図 5 に示す．事前の統計情報取得は，Relay ノード内部のワーカによってバックグラウンドで行われる．ワーカは外部の MongoDB からサンプルデータを取得し，2.2 節の基準により統計取得フィールドを決定する．そして統計情報の収集を行い，統計情報格納テーブルへと格納する．

ユーザが Master ノードに問合せを入力すると，問い合わせは解析され，対応する問合せ固有スキーマが動的に定義される(1)．そしてそのスキーマと対応する

表 2　実験環境

プロセッサ	Xeon E5-2680 2.70GHz×2
メモリ	384GB
OS	CentOS Linux release 7.5 (64bit)
DB	PostgreSQL 10.4 (buffer 16GB) MongoDB 4.0
ワークロード	TPC-H SF10

SQL 問合せが Relay ノードへと送信される(2). Relay ノードは送られてきた問合せ固有スキーマから MongoDB の外部テーブルを作成する(3). この外部テーブルの各列に対し，対応するデータを統計情報格納テーブルから検索し，発見した場合はこのデータを外部テーブルの統計情報として適用する(4). PostgreSQL のオプティマイザはこの外部テーブル定義と統計情報を用いて最適な実行計画を作成し，それに基づいて MongoDB へ接続，JSON データを転送・処理する．最終的に Relay ノードで実行された問合せ結果は Master ノードに送られる．全ての処理が終了次第，Relay ノードは問合せ固有スキーマにより生成した外部テーブルを削除する．

3. 性能評価

我々の目的は ENH と PREV を比較し，性能向上を評価することである．しかし，現在 Master ノードと Relay ノードの接続機構はまだ実装中である．そのため，ENH の性能評価のための予備実験として，我々は Relay ノードからクエリを実行した際の実行時間を計測する．これに加え，ENH のオーバーヘッドを見積もり，足し合わせることで推定の ENH 実行時間を算出する．オーバーヘッドとして考慮する必要があるのは，Master ノードでの問合せ解析，Master ノードから Relay ノードへの問合せと問合せ固有スキーマの転送，そして Relay ノードから Master ノードへの最終結果転送である．しかし最終結果転送時間以外のオーバーヘッドは実行時間と比較して非常に小さいと考えられるため，今回の予備実験では無視するものとする．

3.1. 実験環境

表 2 は PostgreSQL の Master ノード，Relay ノードと MongoDB を動作させる実験環境である．

性能評価には，複雑な解析問合せに対する実行計画の変化と性能向上を検証するため，意思決定用の問合せとして一般的な TPC-H ベンチマークを用いる．但しこのベンチマークのテーブルは RDBMS における実験用に正規化されており，MongoDB に格納する JSON データとしては一般的なデータ構造ではない．そのため，他の研究 [6] [7]と同様に，我々はオリジナルの TPC-H データを非正規化した．具体的には，“nation”と“region”

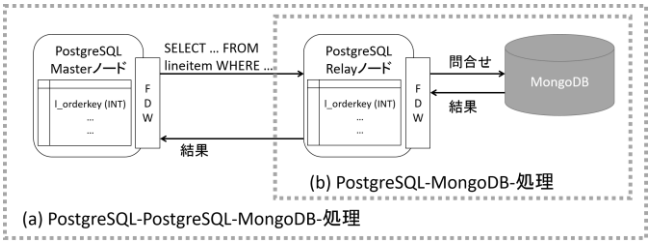

図 6　オーバーヘッド測定方法

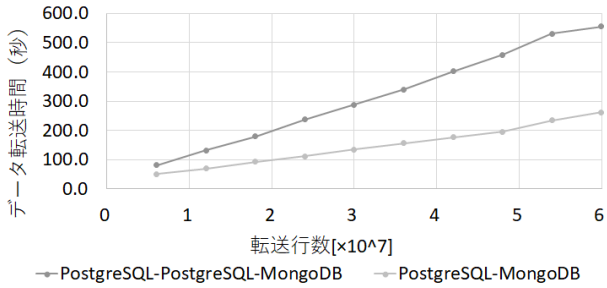

図 7　データ転送時間測定結果

データを“customers”と“supplier”と外部キーで紐づけ，入れ子として包含した．この実験では全ての非正規化された TPC-H データを MongoDB に格納した．更にこの非正規化された入れ子データに対応するよう，TPC-H の問合せも改変した．

3.2. オーバーヘッド見積り

まず ENH の Relay ノードから Master ノードへの結果を転送する際のオーバーヘッドを見積もるため，図 6 に示されるように，Master ノード，Relay ノードどちらも JSON データの各フィールドをテーブル列として定義した状態で実験を行った．図 6 (a)は，PostgreSQL サーバから別の PostgreSQL サーバに問合せをプッシュダウンし，プッシュダウン先から MongoDB へと接続する．対して図 6 (b)は，一つの PostgreSQL サーバから直接 MongoDB へと接続する．この二つの環境で同じ問合せを実行した場合，その差はおおよそ，図 6 に示される PostgreSQL 間の結果転送時間であると考えられる．これを Master ノードから Relay ノードへの結果転送オーバーヘッドの見積もりとして使用する．

図 7 は TPC-H の Q1 の転送行数を WHERE 句の変更によって変え，転送時間を測ったグラフである．転送行数に比例して転送時間が増加していることが分かる．我々の提案手法は問合せの処理を Relay ノードに委譲するため，Master ノードへ転送するのは結果のみである．また，今回は意思決定のための解析クエリを対象としており，元データの行数が多くても中間で集計処理され，結果は数行のデータとなる．以上より Master ノードへ転送するデータ量は一般的には十分に小さい

表 3　TPC-H の実行時間とオーバーヘッド

クエリ	PREV(s)	ENH(s)	ENH'(s)	OVH(s)
1	281.52	259.95	259.74	0.21
3	221.65	211.16	210.64	0.52
4	2,385.21	237.89	237.63	0.26
5	3,863.45	185.08	184.82	0.26
6	74.63	76.17	76.11	0.05
7	Error	223.21	223.00	0.21
8	247.23	199.53	199.43	0.10
9	385.57	262.29	253.23	9.06
10	118.40	132.34	131.82	0.52

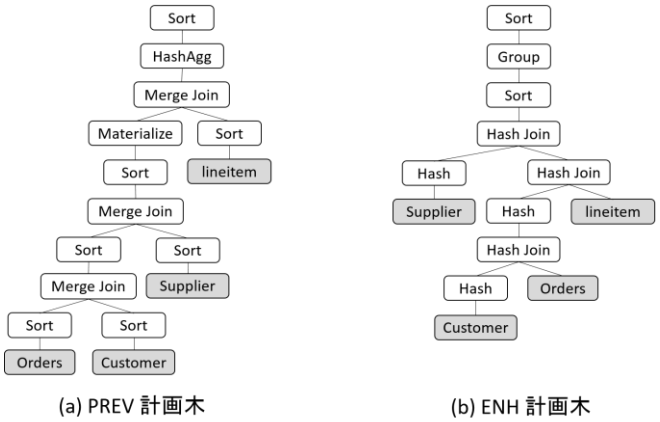

図 8　Q5 の実行計画木

と考えられる.

表 3 の OVH 列は図 7 の結果を元に推測した TPC-H クエリの結果転送オーバーヘッドである. これらは ENH'列に示される問合せ処理時間と比べて非常に小さいといえる.

3.3. 性能測定結果

次に, 実際に TPC-H の Q1 から Q10 までの処理を行った. PREV と ENH の測定結果を表 3 に示す. ただし ENH は ENH'（Relay ノードから実行した処理時間）と OVH（3.2 節で見積もった Relay ノードから Master ノードへの結果転送オーバーヘッド）を合計した推測値である. PREV は Q4, 5, 7 について特に多くの時間が掛かっている. なお, Q2 はどちらの手法も非常に時間がかかり測定できなかったため比較を行っていない. Q7 の PREV は一時間以上止まらず, 最終的にメモリエラーで停止した. 一方で, ENH は Relay ノードと Master ノードのオーバーヘッドがあるが, 明らかに時間を要した問合せはなかった. これは列の統計情報が実行計画の最適化に有効に使われていることを示している.

特に差が大きい問合せの一つである Q5 について, その PREV と ENH それぞれの実行計画木を図 8 に示す. この二つの実行計画木は結合方式が明らかに異なっており, PREV はマージ結合を選択している一方で, ENH はハッシュ結合を選択している. そして PREV の実行計画は, Orders（15,000,000 行）, Customer（1,500,000 行）, Supplier（1,000,000 行）の三テーブルを結合する過程で, 中間結果が 1,809,554,466 行にまで膨れ上がっており, マージ結合のためのソートに非常に多くの時間を消費していた.

これは PREV にはソートキーとなる列の統計情報が不足しており, オプティマイザがコスト推定にデフォルト値を用いたことが原因である. このため選択率などが正しく推定されず, ハッシュがメモリに乗り切らないという誤った判断をしたことで, ハッシュ結合を回避してしまった. これにより, 膨大な中間結果を生むマージ結合を繰り返す非最適な実行計画が生成された.

一方で ENH は正しくハッシュのメモリ量を推定し, ハッシュ結合を選択してソートを回避したため, PREV の 20.9 倍の速度で処理を終了している. このように, 実行計画木を深く解析することで, ENH が統計情報を用いたことで最適と思われる実行計画を選択しているという結果を得た.

以上の測定から, 我々は非最適な実行計画の生成が問合せの実行時間に大きな影響を与えることを確認した. この処理性能の低下は Master ノードと Relay ノード間のデータ転送オーバーヘッドに比べて非常に巨大であり, これらの結果は我々の手法が RDBMS と NoSQL データベース間の効率的なスキーマレス接続を可能にすることを示している.

4. 関連研究

半構造化データをデータベースで解析する研究は様々なものが存在する.

Zhen Hua Liu らは Oracle データベース内で JSON を格納・処理する方法として, 動的スキーマの JSON DataGuide と JSON のバイナリエンコード形式である OSON を提案している [8]. Zhiyi Wang らは木構造データのための解析データベースシステムを発表している [9]. これは実際に使われている JSON にはデータの生成元の特性によってデータ構造に傾きが存在するので, その特徴を生かした圧縮方法や処理方法を提案している. これらの研究は JSON の解析速度を改良している点で提案手法と似ているが, エンジン内部に JSON を保存することを想定しており, 外部データソースから JSON を読む我々の提案手法とは適用する場面が異なる.

外部の NoSQL データベースに SQL を通じてアクセスすることに注目した製品や研究も多く存在する. Apache Drill [10]はデータの局所性を考慮し外部データソースに処理をプッシュダウンすることで実行性能を上げることを特長としており, 複雑なクエリに対す

る実行計画の生成に着目した本研究とは着眼点が異なる．一方，Esper [11]は半構造化データに対するスキーマレスな SQL 問合せを可能にするパーサ兼オプティマイザである．Esper は最初のデータを受け取るまで問合せの実行計画の最適化を遅延することで，実行性能を高めている．ストリーム処理用のエンジンであるため，データの内容に関する統計情報は利用していない．

5. 結論

本論文において我々は，RDBMS から SQL を用いて NoSQL データベースから半構造化データを取り出し，処理をする際に，RDBMS の高性能な実行計画最適化機構を十分に使う効率的な手法を提案した．我々の手法はバックグラウンドにおいて NoSQL データのスキーマを推定し，統計情報を解析・収集する．そして問合せ処理の際には，問合せ固有スキーマを生成し，これに対して事前取得した統計情報を適用する．

TPC-H を変形したベンチマークを MongoDB から読みだす予備実験において，ベストケースでは提案手法は以前の手法に比べて 20.9 倍に処理性能が向上するという結果を得た．

本論文に示した実装では，データ転送オーバーヘッドが非常に小さいと推定し，中間ノードを用いた．今後は実装中の部分を完成させ，中間ノードのオーバーヘッド測定を行っていく．また，より現実に即した JSON データで実験を行うことも検討している．

参 考 文 献

[1] John Roijackers and George HL Fletcher. “On bridging relational and document-centric data stores". *In British National Conference on Databases*, pp. 135-148. Springer, 2013.

[2] Ramon Lawrence. “Integration and virtualization of relational SQL and NoSQL systems including MySQL and MongoDB". In Computational Science and Computational Intelligence (CSCI), *2014 International Conference on,* Vol. 1, pp. 285-290. IEEE, 2014.

[3] “PostgreSQL foreign data wrapper for MongoDB”. https://github.com/EnterpriseDB/mongo_fdw.

[4] 加藤千裕, 中村実, 原田リリアン. NoSQL データソースに対する SQL クエリ処理実行の効率化．日本データベース学会和文論文誌, Vol.16-J, 2018.

[5] “BSON – Binary JSON”. http://bsonspec.org.

[6] Patrick O'Neil, Elizabeth O'Neil, Xuedong Chen, and Stephen Revilak. The Star Schema Benchmark and Augmented Fact Table Indexing. In *Technology Conference on Performance Evaluation and Benchmarking*, pp. 237-252. Springer, 2009.

[7] Max Chevalier, Mohammed El Malki, Arlind Kopliku, Olivier Teste, and Ronan Tournier. Benchmark for OLAP on NoSQL Technologies. In *9th IEEE International Conference on Research Challenges in Information Science (IEEE RCIS 2015)*, pp. 480-485, 2015.

[8] Zhen Hua Liu, Beda Hammerschmidt, Doug McMahon, Ying Lu, and Hui Joe Chang. Closing the functional and Performance Gap between SQL and NoSQL. *In Proceedings of the 2016 International Conference on Management of Data*, pp. 227-238. ACM, 2016.

[9] Zhiyi Wang and Shimin Chen. Exploiting Common Patterns for Tree-Structured Data. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pp. 883-896. ACM, 2017.

[10] “Apache Drill – Schema-free SQL Query Engine for Hadoop, NoSQL and Cloud Storage". https://drill.apache.org.

[11] “Esper - EsperTech”. http://www.espertech.com/esper.