Vol.2010-DBS-150 No.18
Vol.2010-IFAT-99 No.18
2010/8/4

レプリケーションのバッファ内容重複排除による応答速度改善手法

山室　健[†1]　須賀啓敏[†1]　小谷尚也[†1]

本論文では，データベースのI/O単位であるブロック情報を利用したロードバランサの振り分け手法を提案する．同期した複数データベースにクエリを振り分ける環境下では，各バッファ間に重複したデータ領域が発生する．しかしSQL構文だけでは，クエリが参照するデータ領域が正確に分からず，バッファ間の重複を起こさないクエリの振り分けが難しい．そこで本手法では，クエリの処理中に参照したデータ領域を記録し，ロードバランサが利用するメタ情報を作成する．そして，このメタ情報を利用することで振り分けるデータベースを決定する．提案手法を実装し，OSDL DBT-1で評価を行った結果，ある条件下でメモリ重複を20%まで抑えることができ，結果としてクエリの応答速度の改善ができることが分かった．本論文の最後では，この条件の妥当性を検証する．

Block-Based Memory Awareness Load Balancing in Replicated Databases

Takeshi Yamamuro,[†1] Yoshiharu Suga[†1] and Naoya Kotani[†1]

We present a block-based memory-aware load balancing technique by means of block (a unit of I/O) reference patterns. There is a well-known problem called "Memory Contention" problems in load balancing for replicated databases. That means that there are duplicated data over buffers in them. In load balancing, however, SQL statements are insufficient to exactly estimate which data queries need, so it's difficult to dispatch queries without the duplication. Our approach builds meta data based on information which it records in query processing, and look up in the meta data which database it should dispatch queries to. We implement a prototype load balancing system, and incorporate OSDL DBT-1. We demonstrate that our technique cut down memory contention to 20% under certain conditions, and improve response time of queries in OSDL DBT-1. Finally, we validate those conditions.

1. 背　景

近年インターネット上にはGoogleやYahoo!に代表される様々サービスが存在し，それらは一般利用者に広く普及している．最近ではAmazonクラウドサービスやEC構築OSSのEC-CUBE[1]など，誰でも容易にインターネット上でサービスを始めることが可能な環境が整ってきている．

このような背景において，サービスに関係するデータを管理するデータベース層における性能問題が指摘されている[2]．このデータベース層は性能向上が難しく，古くから性能向上に関する研究が様々行われてきた．複数のデータベース上にデータを複製するレプリケーション技術は，長年研究されている性能向上を目的とする研究分野の1つで，近年も盛んに研究されている[3)–5)]．

レプリケーションされた複数データベースに対する参照は，問い合わせ結果が同じになる．そのためインターネット向けのシステムでは，入力クエリを特定のルールに従い振り分けることで，負荷を分散させることがある．このルールとは例えば，クエリをある確率で振り分ける方法（等確率で順に振り分けるRound-Robinなど）や，各データベースのCPUやI/O使用率をもとに振り分ける方法である．

これらの従来の振り分け手法では，メモリ重複（Memory Contention）問題が発生する[6),7)]．メモリ重複とは，同じデータが複数のデータベース上のバッファに存在することを示す．例えば，あるクエリAと，そのクエリが参照するデータXの場合を考える．このクエリAを異なるデータベースに振り分けた場合，異なるバッファ上にデータAが存在してしまうケースがメモリ重複である．2台のデータベースを利用し，メモリ重複を測定した結果を**図1**に示す．累計バッファ滞在時間3,600s以上で，かつ両方のデータベース上バッファに存在していたデータ領域をメモリ重複と判断している．図から従来の振り分け手法では，メモリ重複が90%近いことが分かる．メモリ重複は，複数あるデータベース全体のメモリ空間を効率的に利用できていないため，性能低下の原因となる可能性がある．

代表的なメモリ重複解消に関する研究は，クエリ単位で行うもの[6)]と，リレーション単位で行うもの[7)]の2つがある．前者は，以前処理したクエリがロードバランサに再び入力

†1 NTTサイバースペース研究所
NTT Cyber Space Labratries

Vol.2010-DBS-150 No.18
Vol.2010-IFAT-99 No.18
2010/8/4

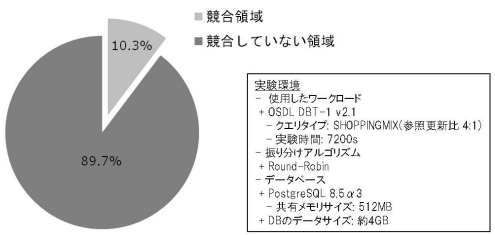


図 1 データベース間のメモリ重複割合

された場合，そのクエリを処理するために必要なデータが，まだバッファ上にあることを期待し，以前と同じデータベースに処理を振り分ける手法である．後者は，データベースとリレーションを固定的に対応付け，クエリが参照するリレーションに対応付けられたデータベースに処理を振り分ける．これにより，異なるデータベース上のバッファに重複するデータが存在しないようにする手法である．前者のアプローチは，クエリが参照するリレーション数やレコード数が多く，これらのデータを処理するために I/O 量が多く発生する場合，性能が悪化する[7]．後者は，ワークロードの大半が単一のリレーションを参照する場合，そのリレーションを割り当てたデータベースにクエリが集中する．例えば，データベース A/B，リレーション X/Y/Z がある場合を考える．データベース A にリレーション X を割り当て，データベース B にリレーション Y/Z を割り当てる．この時，ワークロードの大半がリレーション X を参照する場合に負荷が偏ってしまう．

本研究では，従来手法のクエリ/リレーションと比べ，より詳細なブロック（データベースの I/O 入出力単位）に着目し，上記 2 つの問題が発生しない手法を提案する．本論文の貢献は以下 3 つである．

- クエリが参照するブロック情報を利用した振り分け手法の提案．データベースのエグゼキュータで，ブロック情報を記録し，これを利用することでロードバランサ用のメタ情報を作成する．
- 提案システムのプロトタイプ作成，TPC-W の部分実装である OSDL DBT-1 で実験を行い，特定条件下でメモリ重複を 20%程度までに抑えられることを示す．また，その結果得られる応答速度改善についても示す．
- 上記条件の妥当性考察，過去の研究[8] により，インターネット上サービスの参照確率分布は局所的になるという経験的観測を利用し，条件を再考する．

2. 問題設定

2.1 ロードバランサにおけるクエリ振り分けの問題

ある時間 T 内に入力されるクエリ集合 Q を，レプリケーションで同期したデータベース N 台に分類する問題を考える．時間 T は，1 日，1 週間などワークロードが周期性を持つ範囲を指す．クエリ集合 Q を以下のように定める．

$$Q = \{q_t | t \in T\} \tag{1}$$

この問題は，以下の全写像関数 γ に相当する処理をロードバランサが持つことを示している．

$$\gamma : Q \to 1, 2, 3, \cdots N \tag{2}$$

本研究では，クエリ単位/リレーション単位で発生していた問題を，データベースの I/O 入出力単位であるブロックを利用することで解決する．ブロックとは連続した固定長連続データ領域のことで，リレーションはブロックの集合で構成され，ブロック内には 1 個以上のレコードが格納される．例えば，リレーションが N 個のレコードを持ち，単一のブロックに 2 個のレコードを格納することができる場合，このリレーションは $\lceil \frac{N}{2} \rceil$ 個のブロックから構成されることになる．入力されたクエリは，処理に必要なレコードが置かれているブロックを探し，これをメモリに読み込み，結果を出力する．そのため，上記の γ 関数に相当する処理では，入力されたクエリの参照ブロックを考慮し，全クエリ集合 Q を処理するため必要な I/O 量が最小になるようにクエリを分類し，振り分け先データベースを決定する．

2.2 ブロック参照情報を利用するための課題

ブロックの参照情報を利用するため，以下 2 つの点を考慮する必要がある．

(1) 参照するブロックはクエリがコンパイルされた後に決定される点
(2) ブロック間には参照関係が存在する点

まず課題 (1) に関して，データベースは入力されたクエリをコンパイルすることで，参照するブロックを決定する．一般的なロードバランサでは，コンパイル前のクエリ情報（SQL

Vol.2010-DBS-150 No.18
Vol.2010-IFAT-99 No.18
2010/8/4

構文）しか利用できないため，コンパイル前の情報とコンパイル後の情報を関連付ける必要がある．課題 (2) に関して，クエリには関数従属性，トランザクションブロックや，LIKE 句/BETWEEN 句などによって同時に参照されるブロック集合が存在する．例えば，以下のようなクエリを考える．

Query(X)=SELECT * FROM ITEM WHERE i_subject = 'X'

この場合に，Query(HELATH) と Query(ARTS) が入力され場合のブロック間参照関係の例を**図 2** に示す．

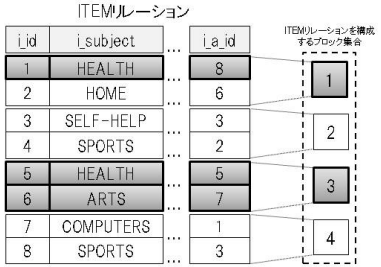


図 2 ブロック間の参照関係

Query(HEALTH) がデータベースに入力された場合，ブロック 1 とブロック 3 を同時に参照する．一方 Query(ARTS) が入力された場合，ブロック 3 を参照する．この時，この 2 つのクエリが入力される場合，ブロック 3 は両方のクエリから参照されているため，このブロックが参照関係を発生させる．このような場合，クエリのパラメータ（この例では，X の値）からはブロックの参照関係が分からないため，コンパイル前のクエリ情報だけではクエリの分類ができない．

2.3 インターネット向けワークロードの特徴

データベースが処理するワークロードは対象とする領域毎に様々であり，特定領域に限った手法として，その領域のワークロードに発生する特徴を利用した研究は非常に多い．本研究も，近年のインターネット上サービスにおけるワークロードの特徴である以下 2 つ着目し，これらを前提とすることで手法の検討を行う．

(1) ワークロード中の大半のクエリが単純にキーを利用したアクセス
(2) データベース上のデータ領域のアクセスの大半がインデックス経由

2000 年後半から memcached，CounchDB や Redis などに代表される KVS（Key-Value Store）が広く利用されるようになった．この背景にはデータベース層の性能問題があり，データベース処理の記述言語として広く認識されている SQL のように記述能力は高いが，データベース上でボトルネックを起こしやすいものに比べ，KVS のように記述が単純で，高性能化しやすいものが近年より好まれる傾向にある．また，インターネット向けのサービスで使用されるクエリの大半が主キーを用いた単純なものであるという報告[9] もある．これらの観点から前提 (1) の条件を制約として，より複雑なクエリを持つワークロードは対象外とする．前提 (2) に関して，一般的にデータベースのデータ領域へのアクセスはシーケンシャルアクセス，インデックスアクセスの 2 つが存在する．これらのアクセス戦略はインデックスの有無，アクセスコストを考慮して決定される．しかし，近年インターネットサービスの品質を決める重要な指標として，サービス反応速度が挙げられており[10]，データベースの処理時間は限りなく短いことが要求される背景から，サービスのシステム設計時にデータベースに入力されるクエリはインデックスを利用するように調整され，比較的時間のかかるシーケンシャルスキャンは避けられる傾向にある．そのため本研究では，インデックス経由のデータ領域のアクセスだけを対象とする．

3. 提 案 手 法

3.1 提案手法概要

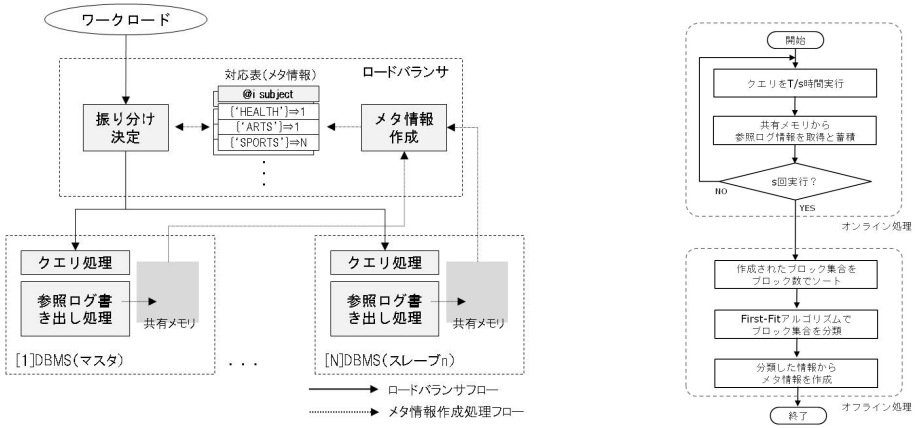
本研究では，I/O 量が多くなった場合に発生する性能が悪化する問題と，ワークロードが参照するリレーションが偏った場合に発生する問題を，より粒度の細かいデータベース I/O 単位であるブロックの参照情報を利用することで解決する手法を提案する．クエリとブロックの対応関係がクエリの実行後に決定される点と，ブロック間に参照関係がある点を考慮し，クエリ処理の実行ログを利用し，ロードバランサが利用するメタ情報を作成する．

メタ情報の作成手順は，以下 2 つである．

(1) データベースのクエリ処理中に，クエリの BEGIN から COMMIT 区間で参照したブロックの集合を，クエリのパラメータと共に記録する．
(2) クエリ処理中に取得した実行ログのエントリに対して，First-Fit アルゴリズム[11] を利用することで，データベース台数 N に分類し，メタ情報を作成する．

提案手法の全体概要を**図 3(a)** に，作成手順のフローチャートを図 3(a) に示す．図 3(a) の「参照ログ書き出し処理」で処理 (1) をオンラインで，時間 T 実行する．そして「メタ情報作成」で処理 (2) をオフラインで行い，作成されたメタ情報を利用し，クエリを振り分

Vol.2010-DBS-150 No.18
Vol.2010-IFAT-99 No.18
2010/8/4



(a) 概要図　　(b) フローチャート図

図 3　提案手法

ける．

3.2　実行ログの記録

クエリ処理は，データベースのエグゼキュータで実行されるため，このモジュールでブロックの参照情報とクエリのパラメータを，(HEALTH → 1，8，9)，(SELF-HELP → 7，13，28，32）の様に 1 つのエントリとして記録する．これらのエントリは共有メモリ上に書き出され後，オンライン処理中に s 回に分けてロードバランサの「メタ情報作成」に転送される．

3.3　実行ログエントリの分類

取得した実行ログを利用し，以下の手順でメタ情報を作成する．

(1)　実行ログのエントリをブロック数が多い順に並び替える．

(2)　データベース台数 N 個のビンを想定し，荷物の容量をブロック数と考え，First-Fit アルゴリズムに従い，各エントリを N 個に分類する．

(3)　各ビンに分類されたエントリをもとにメタ情報を作成する．

First-Fit アルゴリズムとは，容量を持つ荷物を，複数あるビンに均等に詰める問題に対する近似的なアプローチであり，既に詰められている荷物の総容量の少ないビンに荷物を詰めていく手法である．本手法では荷物を各エントリと考え，それらの容量はそのエントリに含まれるブロック数とすることで，各データベースに対応するビンに割り当てられるブロック数を平均化することを目的とする．例えば，ビン A とビン B にエントリ 1（SPORTS → 1，2，3，4，5），エントリ 2（SELF-HELP → 6，7，8，9），エントリ 3（HEALTH → 10，11，12），の順序で詰めることを考える．まず，ビン A にエントリ 1 を詰める．その時，総容量の少ないビンは B（総容量 0）となり，次のエントリ 2 は B に詰められる．同じように，最後のエントリ 3 は，総容量の少ないビン B（総容量 4）に詰められる．結果，ビン A は総量 5 で，対応付けられたエントリは 1，一方ビン B は総量 7 で，対応付けられたエントリは 2 と 3 となる．しかし，実際は，エントリ間のブロック集合が重複している場合がある．例えば，続いてエントリ 5（COMPUTERS → 11，12，13）を詰める場合を考える．First-Fit アルゴリズムに従うならば総容量 5 のビン A に詰められるが，ブロック 11 と 12 は既にビン B に詰められているため，ビン A に詰めた場合，メモリ重複の原因となる．そこで本手法では例外として，最小のビンの総容量と，既に部分的に詰められているビンに詰めた場合の総容量の差が，最小のビンの総容量の K%を超えない場合に，これを許容するものとする．先ほどの例で $K = 5$ とした場合，総容量の差が 60%（$\frac{8-5}{5}$）になるため，通常通りビン A に詰める．

3.4　振り分け処理

振り分け処理は，クエリのパラメータを利用して，メタ情報を参照し，その中に記述されたデータベースに処理を転送する．クエリ内にパラメータが複数個ある場合には，最も参照ブロック数が多いパラメータを優先する．それ以外の，つまりクエリのパラメータがメタ情報に無い場合には，Round-Robin に従いクエリを転送する．

4. 実　　験

4.1　実 験 環 境

提案手法を OSDL DBT-1 に組み込んだもので実験を行う．OSDL DBT-1 は，E コマース向けシステムの標準ベンチマークとして策定された TPC-W を部分的に実装したベンチマークツールである．クエリパタンとして ORDERING_MIX，SHOPPING_MIX，BROWSING_MIX の 3 種類が用意されている．本実験では，標準設定値である SHOPPING_MIX（参照更新比 4:1）を選択し，評価を行った．OSDL DBT-1 の設定値は全ての実験において，item を 1,000,000，customers を 2,880,000，eu/min を 400 に，平均 think_time を 7.2 に設定している．

Vol.2010-DBS-150 No.18
Vol.2010-IFAT-99 No.18
2010/8/4

サーバ構成に関しては，OSDL DBT-1 にサーバ 1 台使用し，各データベースも個別のサーバ上に配置した．全てのサーバは同等の性能と仕様のものを用い，サーバ間は 1000BASE-T の L2 スイッチで接続され，同一のラックに収納されている．サーバは linux kernel-2.6.18 の CentOS 5.2 を用い，CPU は Intel Xeon X5260 の 3.30GHz，物理メモリは 16GB である．実験では，カーネルの起動パラメータを利用することで物理メモリサイズを変更し，評価を行った．使用したデータベースは OSS の PostgreSQL 8.5 α 3 で，パラメータの shared_buffer を 512MB に変更した以外は，全て初期値で実験を行っている．

実行ログエントリの分類の際の，First-Fit アルゴリズムの例外を許す最小ビン容量の差の割合に関しては，各データベースに対応するビンに割り当てられるブロック数を平均化する目的から，全ての実験において $K = 5$ とした．

4.2 実験目的

実験において，以下を確認すること目的とする．

- メモリ重複に影響のあるパラメータは何か？そのパラメータを変化させた場合，メモリ重複の割合はどのように変化するか？
- 物理メモリサイズを変化させた場合のクエリ応答速度の変化

4.3 実験結果

データベースには様々なパラメータが存在するが，その中でもメモリ重複の割合に影響があったパラメータは FILLFACTOR である．これはブロック内に含まれるレコード数を制御する PostgreSQL のパラメータである．単一ブロックに含まれるレコード数が少ない場合，データベースサイズは増大する．データベースを 2 台使用し，FILLFACTOR を変えた場合の提案手法のメモリ重複変化に関する実験結果を**図 4** に示す．実験で利用した FILLFACTOR の値は 30%，50%，80%，それぞれのデータベースサイズは，14GB，7GB，4GB である．実験結果から分かる通り，ブロック内のレコード数が下がるにつれて，データベース間のメモリ重複割合が下がり，FILLFACTOR が 30%の時に最大でメモリ重複割合が 20%程度まで抑えられている．逆に FILLFACTOR が 80%を超えるとメモリ重複が 100%になり，提案手法の効果が無くなっていることが分かる．

FILLFACTOR の値を 30%，50%，80%の条件で，ロードバランサの一般的な戦略として用いられる Round-Robin と，提案手法（Block-Based）の応答速度の変化を評価した結果を**図 5**，**図 6**，**図 7** に示す．OSDL DBT-1 に含まれる 12 個のトランザクションの中で，図では HOME トランザクションだけを示しているが，HOME トランザクションを含む 8 個の参照トランザクションは同等の傾向が得られている．提案手法が最もメモリ重複を抑えることのできた 30%の条件下（図 5）では，Round-Robin における物理メモリ 10GB の応答速度と，提案手法における物理メモリ 6GB の応答速度がほぼ同じである．ここから，提案手法はより少ない物理メモリで Round-Robin と同等の応答速度を維持できていることが分かる．これは，従来手法ではメモリに載ることの少なかったデータ領域を、提案手法では長期間メモリに滞在させることができているためである．そのため，メモリ重複の割合が高くなってしまう 50%（図 6）と 80%（図 7）の条件下では，この効果が少なくなり，最終的に Round-Robin と応答速度の変化が同等になっていることが分かる．

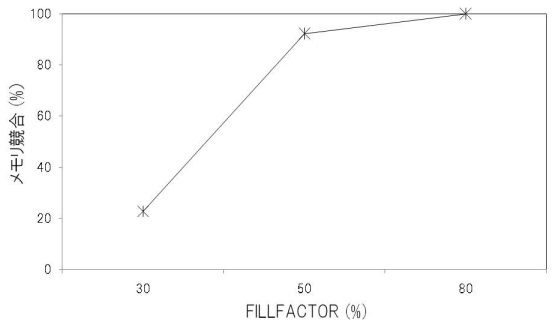


図 4 メモリ重複評価

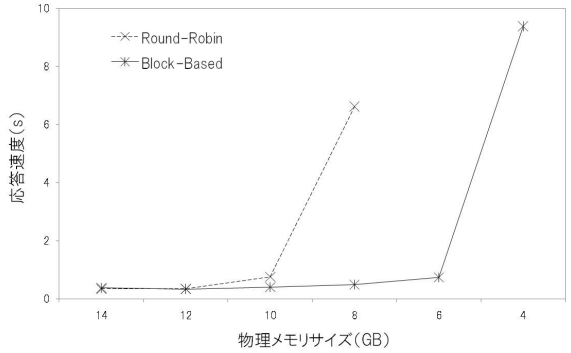


図 5 FILLFACTOR=30 における応答速度評価

Vol.2010-DBS-150 No.18
Vol.2010-IFAT-99 No.18
2010/8/4

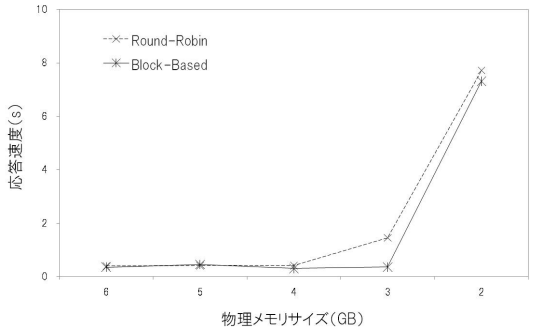


図 6 FILLFACTOR=50 における応答速度評価

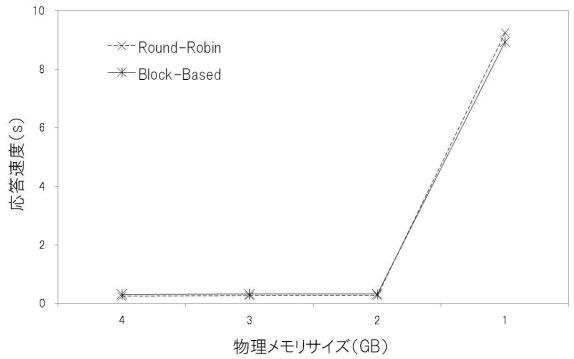


図 7 FILLFACTOR=80 における応答速度評価

5. 考　　察

5.1 考察観点

実験結果の考察観点は以下 2 つである．

- ブロック内に含まれるレコード数（FILLFACTOR）が低い場合に，メモリ重複割合が低くなるのはなぜか？
- 提案手法の効果が高くなる条件は妥当か？

5.2 メモリ重複のモデル化

メモリ重複モデルの概要を**図 8** に示す．クエリ集合 Q は式 (1) に従い，これを均等に 2 分類できることを前提とする．ここで，あるブロックでメモリ重複が発生する確率を P_{mc} と置く．メモリ重複が発生するブロック数 $|N_{loss}|$ が，P_{mc} の 2 項分布に従うとした場合，$P(|N_{loss}|)$ は以下のように表わすことができる．

$$P(|N_{loss}| = k) = \binom{N}{k} P_{mc}^{k}(1 - P_{mc})^{N-k} \tag{3}$$

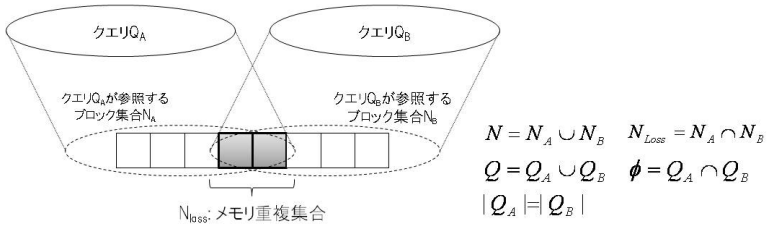


図 8 メモリ重複モデル概要図

次に，あるブロックがメモリ重複を起こす確率 P_{mc} を考える．まず，クエリ処理のモデル化を行う．ある単一クエリは，N 個のブロックからなるデータベースのうち WS 個参照する．ブロックの参照確率は一様で，独立であるとした場合，あるブロックが，そのクエリから参照される確率 P_{ref} は以下の式で表わされる．

$$P_{ref} = \frac{WS}{N} \tag{4}$$

上記で定義したクエリが $|Q|$ 個連続で入力され，それぞれが独立であるとした場合，あるブロックを参照するクエリ数の確率分布は以下の式で表わすことができる．

$$P(X = l) = \binom{N}{l} P_{ref}^{l}(1 - P_{ref})^{N-l} \tag{5}$$

Vol.2010-DBS-150 No.18
Vol.2010-IFAT-99 No.18
2010/8/4

ここで X は，参照するクエリ数を表す確率変数とする．この結果，P_{mc} は以下のように表わされる．

$$P_{mc} = \sum_{m=2}^{|Q|} P(X = m) p'_m \tag{6}$$

ここで p'_m はあるブロックが m 回参照される場合，クエリ集合 Q_A と Q_B 両方から参照される条件付き確率とする．p'_m はクエリ集合 Q の分類方法によって変化するため，単純にモデル化できない．しかし $WS = 1$ の時，クエリが参照するブロックに重複が発生しないため，p'_m は常に 0 になる．一方，WS が $1, 2, 3, \cdots$ と大きくなるにつれて，p'_m は直に 1 に漸近することが予想される．この観察から，ここでは p'_m は WS の値によってのみ決定される $Q(WS)$ と考え，(5) を以下のように置きかえる．

$$P_{mc} = \sum_{m=2}^{|Q|} P(X = m) Q(WS) = P(X > 1) Q(WS) \tag{7}$$

この結果得られた 3 つのパラメータ $|Q|$，N，WS を持つモデルをもとに，図 8 の実験で得られた値を利用し，算出した推定値を図 8 に示す．上記の観察を参考に，$Q(WS)$ は以下の式を利用した．

$$Q(WS) = 1 - \frac{1}{WS} \tag{8}$$

FILLFACTOR が 50%時の誤差は 15%以上あるが，メモリ重複割合変化の特徴を表すことはできていることが分かる．

モデルでは $P(X > 1)$ が高いほど，メモリ重複の発生するブロック数 $|N_{loss}|$ が大きくなる．$P(X > 1)$ はあるブロックが 2 回以上参照される確率を表しているため，ブロック内のレコード数が多くなるほど高くなる値と考えることができる．このモデルにメモリ重複割合が従うと考えるならば，ブロック内に含まれるレコード数を少なくした場合，$P(X > 1)$ が小さくなり，結果としてメモリ重複する割合が低くなったと考えることができる．

5.3 局所的なアクセスが発生するワークロード適用の考察

前節の考察から，提案手法の効果が高くなる条件は $P(X > 1)$ が小さいことであることを示した．これは，あるブロックを参照するクエリ数が 1 以下である確率が低いことを意味している．しかし，ブロック内のレコード数を少なくする場合，それに伴いデータベースサイズが大きくなるため現実的ではない．そこで，より現実的なワークロードを分析することで，$P(X > 1)$ が小さいことを満たす別条件を検討する．

インターネット上のサービスに対するワークロードは，アクセスが局所的になることが報告[12] されている．そのため，データベース上のレコードの参照もこの特徴を持つことが予測される．参照確率が高いレコードと，そのレコードのブロック集合内の位置に相関が少なく，散らばって配置されている場合，ある単一ブロックは参照確率の高い一部の少ないレコードと，参照確率が低いその他のレコードから構成されることが予測できる．この場合に，このブロックは $P(X > 1)$ が小さくなると推測される．これを評価するため，OSDL DBT-1 のレコード参照確率をパラメータ λ の指数分布に従うように変更し，FILLFACTOR を 80%，物理メモリサイズ 2GB で実験を行った結果を**図 9** に示す．図では応答速度改善効果が見られた BEST_SELLER と NEW_PRODUCT トランザクションのみを示す．図からわかる通り，λ が高くなると Round-Robin に比べ，提案手法の応答速度改善割合が高く，上記で推測した効果が表れていることが分かる．

この結果，局所的なアクセスが発生するワークロードで，かつ参照確率の高いレコードがブロック集合内に散らばって配置されるような条件下では本提案手法が有効であると判断できる．

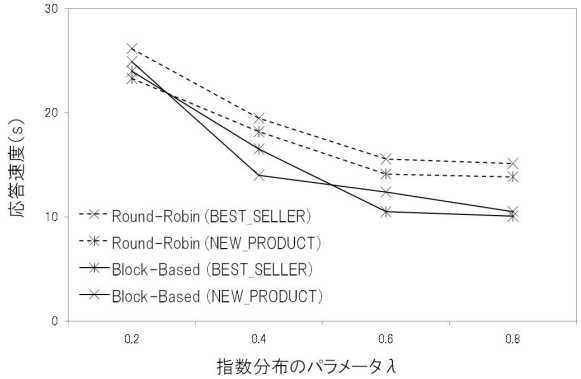


図 9 指数分布の参照確率における評価

Vol.2010-DBS-150 No.18
Vol.2010-IFAT-99 No.18
2010/8/4

6. 関連研究

レプリケーションにおけるメモリ重複解消に関する研究は，以下 2 つに分類できる．

- 複数データベース間のバッファ機構を協調的に動作させるもの
- ロードバランサがバッファを意識した振り分け処理を実施するもの

前者に関しては，代表的な商用製品として Oracle の RAC（Real Application Clusters）や IBM の DB2 pureScale8 があり，研究[8),13)] も古くから行われている．複数のデータベース間でバッファ情報を共有する必要があるため，データのどの部分がバッファ上に存在するかを示すルックアップ表の管理が必要になる．そのためこのアプローチは，データベースの I/O 単位でメモリ重複の制御が行える利点がある半面，データベース間の通信やロック処理必要になるため，その部分がボトルネックになる欠点がある．一方後者に関しては，背景で紹介したクエリ単位で行う手法[6)] とリレーション単位で行う手法[7)] がある．このアプローチはロック処理が不要ではあるが，データベースの I/O 単位のメモリ重複を直接制御できないため非効率的な部分がある．本研究では，データベースの I/O 単位であるブロックの参照情報を利用し，ロードバランサが利用可能なメタ情報を作成することで，従来研究の相反する問題を解決している．

7. おわりに

本論文では，ブロックの参照情報から作成したメタ情報利用するロードバランサの振り分け手法を提案した．本手法では，データベースのエグゼキュータにおいて，単一クエリ内のインデックス経由のアクセスを記録し，これを利用することでメタ情報の作成を行った．またこの手法の実装を行い，OSDL DBT-1 で評価を行った結果，ブロック内レコード数が少なくなる場合に，メモリ重複を 20%まで抑えることが可能であることを示した．考察では，この条件の妥当性を検討し，実際のワークロードへの適用を考察した．

今後は以下に挙げる点を検討していく予定である．

- CPU と I/O の使用率を考慮したロードバランサとの連携
- メタ情報作成時のオンライン処理の軽量化

参考文献

1) EC-CUBE, http://www.ec-cube.net/
2) Jayashree Ravi et al.: A survey on dynamic Web content generation and delivery techniques, Journal of Network and Computer Applications, Vol.32, No.5, p943-960 (2009).
3) K. Daudjee and K. Salem.: Lazy Database Replication with Snapshot Isolation, In Proceedings of 32nd International Conference on Very Large Data Bases (2006).
4) M. Patino-Martinez, et al.: Consistent Database Replication at the Middleware Level, ACM Transactions on Computer Systems (TOCS), Volume 23, No. 4 (2005).
5) T. Mishima, et al.: Pangea: An Eager Database Replication Middleware guaranteeing Snapshot Isolation without Modification of Database Servers, In Proceedings of 35nd International Conference on Very Large Data Bases (2009).
6) V. S. Pai et al.: Locality-Aware Request Dsitribution in Cluster-based Network Servers, In Proceedings of the 8th ACM Conference on Architectural Support for Programming Languages and Operatinig Sysmtes (1998).
7) Sameh Elnikety et al.: Tashkent+: Memory-Aware Load Balancing and Update Filtering in Replicated Database, SIGOPS Oper. Syst. Rev., Vol.41, No.3, p399-412 (2007).
8) M. J. Feeley et al.: Implementing global memory management in a workstation cluster, In Proceedings of the Fifteenth ACM symposium on Operating System Principles (1995).
9) Sivasubramanian S. et al.: Autonomic data placement strategies for update-intensive Web applications, In Proceedings of the international workshop on advanced architectures and algorithms for Internet delivery and applications (2005).
10) The Aberdeen Group, http://www.sbigroup.co.jp/news/pr/2009/0416_2389.html
11) M. R. Garey et al.: Resource constrained scheduling as generalized bin packing, J. Combinatrial Teory, Ser. A, p257-298 (1976).
12) Breslau L. et al.: Web caching and zipf-like distributions: evidence and implications, In Proceedings of IEEE conference on computer communications (INFOCOM'99), p126-134 (1999).
13) H. Levy et al.: Implementing Cooperative Prefetching and Caching in a Globally-Managed Memory System, In Proceedings of the ACM SIGMETRICS'98 Conference (1998).