DEIM Forum 2014 D8-3

読み出しデータの新鮮度を考慮するキャッシュ機構

高塚　康成† 　長尾　洋也†† 　矢口　　尭†† 　華井　雅俊†† 　首藤　一幸††

† 東京工業大学 理学部 情報科学科
†† 東京工業大学 大学院情報理工学研究科 数理・計算科学専攻
〒 152–8552 東京都目黒区大岡山 2–12–1–W8–43
E-mail: takatsuka.y.aa@m.titech.ac.jp, hiroya.nagao@is.titech.ac.jp, yaguchi.t.ac@m.titech.ac.jp, hanai.aa@m.titech.ac.jp, shudo@is.titech.ac.jp

あらまし　メモリ等を用いたキャッシュはデータベースのアクセス性能向上に有効である．しかし一方で，データベースへの書き込み時にキャッシュ上のデータを無効化または更新しない限りは，キャッシュ上のデータがデータベース上のものと異なる可能性が時間の経過とともに高くなっていく．つまりデータの新鮮度が下がっていくというトレードオフがある．そこで，データベースの負荷やアクセス性能に応じてキャッシュの使用/不使用を切り替えることで，アクセス性能とデータ新鮮度のトレードオフを調整すること，および，その切り替え方を提案する．また，既存のデータベース管理システムを用いて，提案手法に基づき設計，および実装を行ったキャッシュ機構の性能を評価する．

キーワード　データ新鮮度，キャッシュ，データベース

A Caching Mechanism Considering Data Freshness

Yasunari TAKATSUKA†, Hiroya NAGAO††, Takashi YAGUCHI††, Masatoshi HANAI††, and Kazuyuki SHUDO††

† Dept. of Information Science, Tokyo Institute of Technology
†† Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology
2–12–1–W8–43 Ookayama, Meguro, Tokyo, 152–8552 Japan
E-mail: takatsuka.y.aa@m.titech.ac.jp, hiroya.nagao@is.titech.ac.jp, yaguchi.t.ac@m.titech.ac.jp, hanai.aa@m.titech.ac.jp, shudo@is.titech.ac.jp

Key words　data freshness, cache, database

1. はじめに

データベースのアクセス性能を向上させる方法の一つに，クライアントからのアクセスを高速に行える場所にデータを複製するという手法がある．その代表例がキャッシュである．一度アクセスされたデータをクライアントから高速にアクセスできる場所に複製し，次回以降の読み出しアクセスではその複製を利用する．具体的には，クライアントから地理的に近いサーバにデータを複製することで通信遅延を削減する手法や，メモリ上にデータを複製することでディスクアクセスを削減し，読み出し処理の遅延を小さくする手法などが存在する．

しかし，これらの手法には，複製とオリジナルデータとの整合性をどの程度担保するかという問題がある．キャッシュ上のデータを無効化または更新しない限り，時間の経過とともにそのデータは古くなり，オリジナルデータと異なるデータを読み出してしまう可能性が高まる．

我々は，時間経過による，オリジナルデータと複製の整合性欠如，およびその可能性が高まることをデータ新鮮度の低下と位置づけ，メモリを用いたキャッシュのデータ新鮮度に注目する．

サービス運営の継続を優先する場合，キャッシュの使用が必要な時と不要な時が存在する．例えば，過度のアクセスによりデータベースからの読み出し遅延が一時的に増大する時，サービス運営を継続させるためには遅延を一定値以下に維持する必要があり，そのためにキャッシュを使用する必要がある．しかし，それ以外の遅延が小さい時は，キャッシュを使用せずともサービス運営の継続に支障はない．つまり，データ新鮮度を低下させる可能性のあるキャッシュを使用する必要はない．

そこで，本研究はデータベースの負荷やアクセス性能に応じてキャッシュの使用/不使用を切り替えることで，アクセス性能

と新鮮度のトレードオフを調整すること，および，その切り替え方を提案する．また，提案手法はリバースプロキシに実装する．これにより，データベース管理システムには手を入れずに提案手法を機能させることができる．我々は既存のデータベース管理システムを使用し，提案手法の評価を行う．

本論文の構成は以下のとおりである．2 章でデータ新鮮度について述べ，3 章で我々の提案手法を述べる．4 章で提案手法に基づき設計・実装を行ったキャッシュ機構について述べ，5 章でそのキャッシュ機構の性能評価を行う．6 章で関連研究を述べ，7 章で本研究の貢献と今後の課題をまとめる．

2. データ新鮮度

2.1 データ新鮮度とは

データベースに対して書き込みと読み出しが同時に行われる場合，キャッシュ上のデータは，無効化や更新をしない限り，時間とともにデータベース上のオリジナルデータと異なる可能性が高まっていく．本論文では，そのような読み出しデータに関する整合性の欠如，およびその可能性が高まることをデータの新鮮度低下と捉える．キャッシュに存在するデータの新しさ，またはキャッシュから最新のデータが読み出される度合いをデータ新鮮度と呼ぶ．

2.2 データ新鮮度の低下

データ新鮮度が低下するのは，キャッシュ上のデータとオリジナルデータの間に違いが生じた時である．そして，その違いが生じるのはデータベース上のオリジナルデータが更新される時である．オリジナルデータの更新時に，更新される対象データの複製がキャッシュ上に存在し，かつ，そのキャッシュ上の複製データが更新または無効化されない場合に，複製とオリジナルデータの間に違いが生じる．また，オリジナルデータの更新時にキャッシュを更新したとしても，複製されたデータベース間の同期処理など，更新が内部的に起こった場合に古いデータがキャッシュ上に残ってしまう．したがって，データベースを構成するノードが一台の場合など，全ての更新処理時に必ずキャッシュを更新できる場合以外は，キャッシュを使用すると必ずデータ新鮮度の低下が生じる．

2.3 データ新鮮度を表す指標

データ新鮮度を表す指標としては，以下の二つが例として挙げられる．

- 最新のデータを読み出す割合

読み出し処理全体に対して，最新のデータを読み出した件数の割合が高いほどデータ新鮮度は高い．例えば，クライアント側に書き込みデータを保存し，読み出しデータとの比較を行うことで，最新のデータを読み出した件数を測定することができる．

- キャッシュ上データの未更新時間

キャッシュ上データの未更新時間が長いほどデータ新鮮度は低い．これを測定する方法の例としては，データをキャッシュに保存する際，保存した時刻の情報をそのデータに付加しておく．キャッシュからデータを読み出した際，そのデータが最新のデータであるか否かを確認する．古いデータを読み出していた場合は，データに付加されている保存時刻情報と読み出した時刻の差により，そのデータの未更新時間を計算し保存する．同じデータの未更新時間に関しては，その値がより大きい方を保存する．これにより，キャッシュがどれほどの時間，更新せずに古いデータを保持し続けるのかを調べることが出来る．

2.4 データ新鮮度低下を抑えるキャッシュ方式

キャッシュからデータの読み出しを行う場合にデータ新鮮度を損なわない単純な方法は，そのキャッシュ上のデータが最新であるかどうかをデータベースに確認することである．しかしこれは，データベースへのアクセスを省くことが出来ないため，キャッシュとしての意味をなさない．そのような手法以外で，データ新鮮度の低下を抑えるようなキャッシュ方式の例としては以下の二つが挙げられる．

- **キャッシュ上のデータに有効期限を設ける**

この手法では，キャッシュ上の有効期限を過ぎた古いデータは無効化され，次にそのデータを読み出す場合はデータベースから直接読み出し，その新しいデータを再びキャッシュに保存する．しかし，有効期限を長く設けた場合，キャッシュ上に古いデータが存在する時間が増え，データ新鮮度が低下する．また，有効期限を短く設けた場合，キャッシュ上のデータがすぐに無効化されデータベースへのアクセスが増加し，キャッシュとしての効果が薄れる．有効期限の長さによっては最新のデータにも関わらず無効化しなければならない場合がある．どれほどの時間でデータが更新されるか不明であり，また，データ毎に更新される頻度も異なるため，適切な有効期限の決め方がこの手法の課題である．

- **ライトスルー・キャッシング**

これは，データベースへデータを書き込む際に，そのデータによりキャッシュを更新する手法である．この手法では，まずデータベースへの書き込みを行い，続けて，キャッシュを更新することで書き込み完了とする．この手法を用いると，キャッシュ上にあるデータが最新である可能性が高まる．しかしながら，依然としてデータ新鮮度を完全に保つことはできない．例えば，複製されたデータベース間の同期処理などにより，更新が内部的に起こった場合，古いデータがキャッシュ上に残ってしまう．

3. 提 案 手 法

通常のデータベースシステムは過度のアクセスにより遅延が予想外に大きくなってしまうことがある．しかし，アプリケーションによっては，サービス運営上，遅延をある一定値以下に維持することが望ましい場合がある．つまり，データベースの負荷が増えた際に，ある上限を超えて遅延が大きくなることを防ぎたい．

そこで，我々は，データ新鮮度を調整することで遅延を維持する手法を提案する．

キャッシュを使用すると 2 章で述べたようなデータ新鮮度の低下が生じるが，読み出し遅延を小さくすることができる．したがって，キャッシュの使用/不使用を切り替えることでデータ新鮮度を調整し，読み出し遅延を維持する．具体的には，デー

タの一部をキャッシュしながら，常にデータベースへの負荷を監視する．そして，データベースへの負荷が，あらかじめ設定した限界を超えた時にのみキャッシュを使用する．例えば，読み出し遅延を監視する．読み出し遅延にしきい値を設け，測定した遅延がそのしきい値を超えた時にのみキャッシュを使用する．そうすることで，読み出し遅延がしきい値より小さいときはデータ新鮮度を保ち，しきい値より大きいときは，データ新鮮度を犠牲にして読み出し遅延を維持する．

読み出し遅延以外に，データベースへの負荷を表す指標としては以下のものが挙げられる．

- 書き込みの遅延
- 単位時間あたりの読み書き処理数
- 同時接続クライアント数
- データベースサーバ側の CPU 使用率

これらの指標を用いてキャッシュの使用/不使用を切り替えることも可能である．

4. 実　　装

提案手法に基づき，読み出しデータの新鮮度を考慮するキャッシュ機構を設計・実装した．本論文では，我々の提案するキャッシュ機構を Freshness–aware Reverse Proxy（以下，FARP と略す）と呼ぶ．

4.1 設　　計

我々は，FARP をリバースプロキシ向けに実装した．つまり，FARP の実装はデータベース管理システムから独立している．この設計には以下の利点がある．

- 様々なデータベース管理システムに対応できる
- データベース管理システムに手を加えないため，開発・改良が容易である
- FARP が故障したとしても，元々のデータベースには影響を与えない

FARP を利用するには，クライアントからの読み出し・書き込み要求の送信先を FARP の稼働するサーバに設定するだけでよい．元々のデータベースからの読み出し遅延の監視，キャッシュへのデータ保存，およびキャッシュの使用/不使用の切り替えは全て FARP が行う．

クライアントからは FARP と元々のデータベースを合わせたものが一つのデータベースとして認識される．つまり，FARP はデータベースの機能拡張と位置づけられる．以下では区別のために，オリジナルデータを保存する元々のデータベースをオリジナルデータベース，FARP とオリジナルデータベースを合わせたものをデータベースと呼ぶ．

4.2 Freshness–aware Reverse Proxy

4.2.1 キャッシュアーキテクチャ

FARP のキャッシュサイズは，FARP の起動時に，キャッシュに保存するレコード数の上限値により設定する．

キャッシュサイズは指定されているため，キャッシュが満杯になった際は，キャッシュ上のデータを破棄することで新しくデータを保存するための空きを作らなければならない．そのデータ置換ポリシーは，キャッシュの使用/不使用を切り替える FARP のコアなアルゴリズムに依存しないため，任意のものを用いることが可能である．キャッシュのデータ置換ポリシーの例としては以下のものが挙げられる．

- Least Recently Used（LRU）

未使用の時間が最も長いデータを破棄する．

- Most Recently Used（MRU）

最も最近使われたデータを破棄する．

- Least Frequently Used（LFU）

使用頻度が一番低いデータを破棄する．

また，本論文では 2.4 節で紹介したライトスルー・キャッシングは未実装である．したがって，書き込み時に FARP が行う操作はクライアントから受け取ったデータをデータベースへ受け渡すことのみである．

4.2.2 キャッシュ使用/不使用の切り替え

読み出し遅延を測定し，その値が設定されたしきい値を超えた場合にのみキャッシュからデータを読み出すようにキャッシュ使用/不使用の切り替えを行う．

キャッシュ使用/不使用切り替えの判断には，FARP とオリジナルデータベース間で測定された読み出し遅延を用いる．そのため，切り替え判断に FARP とクライアント間の通信遅延は考慮されない．しかし，オリジナルデータベースのディスクアクセスによる遅延はネットワーク通信による遅延に比べ大きいため，FARP とオリジナルデータベース間の読み出し遅延のみを測定することは，クライアントとデータベース間のおおよその読み出し遅延を知るのに十分である．

また，読み出したいデータがオリジナルデータベースのメモリに存在する場合と存在しない場合など，オリジナルデータベースの状況により個々の読み出し遅延は多少変動し，突発的に読み出し遅延が大きくなることがある．そのため，読み出し要求が発行されるたびにキャッシュ使用/不使用の判断を行うと遅延制御の精度が落ちてしまう可能性がある．そこで，キャッシュ使用/不使用の判断には読み出しの平均遅延を用いる．平均遅延を計算するまでの読み出し回数を指定することにより，キャッシュ使用/不使用の判断を行う間隔を設定する．

FARP は以下のようなアルゴリズムでキャッシュ使用/不使用の切り替えを行う．キャッシュの使用/不使用を示すマーカを cacheSwitch と呼ぶ．cacheSwitch の値が true の場合キャッシュを使用し．cacheSwitch の値が false の場合キャッシュを使用しない．また，平均遅延を計算するまでの読み出し回数を readCount と呼ぶ．

FARP はクライアントから読み出し処理の要求を受け取ると，まず cacheSwitch の値を確認し，その値により異なる処理を行う．

- cacheSwitch が true の場合

（1） キャッシュに目的のデータが保存されているか調べる．

– 保存されている場合はその値を返す．

– 保存されていない場合はオリジナルデータベースから目的のデータを読み出し，そのデータによりキャッシュを更新する．その際に読み出し遅延を測定し，readCount を 1 増やす．

（2） readCount が設定値に達した場合はそれまでの読み出

し遅延の平均を計算する．その値がしきい値以下であった場合は cacheSwitch を false とする．また，readCount を 0 に戻す．

- cacheSwitch が false の場合

（1） オリジナルデータベースから目的のデータを読み出し，読み出したデータによりキャッシュを更新する．その際に読み出しの遅延を測定し，readCount を 1 増やす．

（2） readCount が設定値に達した場合はそれまでの読み出し遅延の平均を計算する．その値がしきい値以上であった場合は cacheSwitch を true とする．また，readCount を 0 に戻す．

4.2.3 チューニングのポイント

FARP には，読み出し遅延のしきい値以外に，性能に影響するパラメータが二つ存在する．どちらも FARP の起動時に設定できる．

- **キャッシュ容量**

FARP のキャッシュ容量が多いほど，キャッシュ使用時にオリジナルデータベースへのアクセスを削減できるため，より大きな負荷に対応できる．

- **キャッシュ使用/不使用の切り替え判断の間隔**

これは読み出し遅延制御の精度に影響する．4.2.2 節で述べた通り，我々の実装ではキャッシュ使用/不使用の切り替えを判断する間隔は平均遅延を計算する間隔と等価である．そして，平均遅延を計算する間隔は，平均遅延を計算するまでの読み出し回数（readCount）を指定することにより設定する．

readCount が大きいほど平均遅延の計算に用いる標本数が多くなり，より正確な平均遅延を測定できる．しかし一方で，readCount が大きすぎるとキャッシュ使用/不使用の切り替え判断までの間隔が大きくなり，切り替えを行いたい時にうまく切り替えを行えない．また逆に，readCount が小さい場合はキャッシュ使用/不使用の切り替えの判断を細かく行うことができるが，平均遅延の計算に用いる標本数が少なくなり正確な平均遅延を計算することが出来ないため，本来切り替える必要のない場面で誤って切り替えを行ってしまう可能性が高まる．

そこで，測定した平均遅延により readCount を動的に変更するように実装した．平均遅延を計算するたびに，FARP が維持したい遅延のしきい値と平均遅延の差を計算し，その差が大きいほど readCount を大きく変更する．また，readCount に上限値を設けることで，readCount が過度に大きくなることを防ぐ．

変更は以下の式により行う．readCount の初期値と上限値はキャッシュ使用時と不使用時で別々に設定することができる．キャッシュ使用時における初期値を cacheLoopCount，不使用時における初期値を direct_readLoopCount とする．

－ キャッシュ使用時

readCount = cacheLoopCount × (平均遅延 / しきい値)

－ キャッシュ不使用時

readCount = direct_readLoopCount /(平均遅延 / しきい値)

また，キャッシュ使用時はキャッシュ不使用時より，キャッシュからの読み出し回数分だけ平均遅延を計算するまでの間隔が長くなる．そのため，キャッシュ使用時の readCount やその上限値はキャッシュ不使用時より小さく設定することを推奨する．

5. 実　　験

本章では，4.2 節で述べた各種パラメータを調整し，提案手法を次の二つの観点から評価する．まず，単位時間あたりの読み書き処理の量を増やすことによりデータベースへの負荷を高めることで，FARP を使用した場合に読み出し遅延をどれほど制御・維持できるかを評価する．次に，読み出しの負荷を高めた場合の新鮮度の変化を評価する．データベース管理システムには Apache Cassandra [1] [5] を使用する．

性能評価には，Yahoo! Cloud Serving Benchmark（YCSB）[2] を用いる．YCSB は NoSQL 向けのベンチマークツールであり，読み出しおよび更新処理の比率や，データベースへのアクセス分布，また 1 秒あたりに発行する処理要求数の目標値を指定できる．ただし，YCSB には遅延の集計を行う機能はあるものの，データ新鮮度を測定する機能はない．そこで，データ新鮮度を測定できるよう YCSB に改良を施した．

5.1 データ新鮮度の測定方法

本実験では，2.3 節で述べた，最新のデータを読み出した割合をデータ新鮮度として測定した．以下の式により計算する．

$$\text{データ新鮮度} = \frac{\text{最新のデータを読み出した件数}}{\text{読み出し処理の総数}}$$

全体の読み出し件数は YCSB により指定する事ができる．したがって，上式のデータ新鮮度を計算するためには最新のデータを読み出した件数を測定する必要がある．そこで，その値を測定できるよう YCSB に改良を施した．最新のデータを読み出した件数を測定する手法としては以下の二つが考えられる．

- 全ての書き込みと読み出し内容のログを書き出し，後にそのログを調査する事で正しく最新のデータを読み出している箇所を確認する．
- 書き込み内容を YCSB 側で保存し，読み出したデータと保存されているデータに不整合が無いか逐次確認する．

一つ目の手法は全ての書き込みと読み出しを網羅的に調査する事ができるため，誤差のない新鮮度測定が可能である．しかし，ログのディスクへの書き出しには時間がかかるため，実験結果に影響を与えないようにこの手法を実装することは難しい．そのため，本論文では二つ目の手法を採用した．

我々が採用した新鮮度測定の手法には考慮すべき点が二つ存在する．一つ目は，データ保存や整合性確認のための余分な負荷が YCSB 側にかかるという点である．その負荷を軽減するために，書き込みデータの保存と整合性の確認を高速に行うことができるよう，データは YCSB を稼働させるマシンのメモリに保存する．また，本研究では，全ての実験に改良した YCSB を用いているため，全ての実験結果に整合性確認のオーバヘッドが含まれている．したがって，改良した YCSB を用いて得た実験結果により，提案手法を比較・評価する事に問題はない．

二つ目は，上述した理由で書き込みデータ全てをメモリに保存した場合，メモリ容量が不足する可能性があるという点である．全ての書き込みデータを YCSB 側で保存しておく場合，データベースのディスク容量に比べてメモリの容量はかなり小さいため，大量の書き込みを行った際は容易にメモリの容量不

足に陥る．それに対処する方法としては，書き込み対象のキーを整数値に変換し，その値をあらかじめ指定した整数値 n で割り，余りが 0 になるデータのみを保存するという手法が考えられる．結果として得られた，古いデータを読み出した回数を n 倍すれば本来の値に近い値が得られるはずである．しかし，この手法は保存する書き込みデータ件数の増加速度を n 分の 1 に抑える効果しかないため，書き込み件数が推定できない場合に有効な手法であるとは言えない．本研究では書き込むデータ件数があらかじめ判明しているため，この手法を採用できるのである．

5.2 ベンチマーク環境

表 1 に利用したマシンの構成を示す．ストレージデバイスには SSD を使用する．

1 ノードに保存するデータ件数は 1000 万件で実験を行う．今回，一つのレコードは 10 個のカラムを持ち，一カラムの容量を 100 バイトとしたため，一レコードの容量は 1KB となる．つまり，データ全体のサイズは 10GB となり，これにさらにスキーマの情報が加わるため，8 GB のメモリに全データが乗る事はない．

表 1　ベンチマーク環境

OS	Ubuntu 12.04.3 LTS
CPU	2.40 GHz Xeon(R) E5620 × 2
メモリ	8 GB RAM
Java	Java SE 7 Update 4

5.3 ワークロード

書き込み比率が高い Update—Heavy と，読み出し比率が高い Read–Heavy の二つのワークロードで実験を行う．表 2 にワークロード毎の書き込み処理と読み出し処理の割合を示す．データベースへのアクセス分布としては Zipf 分布を用いる．Zipf 分布は，データの新しさとは無関係に各データへのアクセス頻度が決まるようなアプリケーションのアクセス分布を模す．

データベースにかかる負荷が変わっても読み出し遅延を制御できることを確認するために，データベースにかかる負荷の大きさを変える必要がある．今回は，一秒あたりに発行する処理要求数の目標値（以下，スループットと呼ぶ）を増加させることでデータベースにかかる負荷を増やす．一回のワークロードにおける処理命令数は 100 万件とし，それぞれのスループットにおける読み出し遅延の平均，およびデータ新鮮度を測定する．

また，今回の実験では，書き込みデータの総量がマシン搭載メモリに比べて小さい．そのため，新鮮度測定のために，YCSB 側で書き込みデータの全てを保存する．

表 2　実験に用いるワークロード

Workload	書き込み割合	読み出し割合	アクセス分布
Update–Heavy	50%	50%	Zipf 分布
Read–Heavy	5%	95%	Zipf 分布

5.4 Freshness–aware Reverse Proxy の設定

データベースノード一台につき一台の FARP を稼働させる．FARP のキャッシュアルゴリズムには LRU を使用し，4.2.3 節で述べた二種のパラメータは以下のような設定で実験を行う．

- **キャッシュ容量**

100 万レコードを保存する．

- **キャッシュ使用/不使用の切り替え判断の間隔**

4.2.3 節で説明した，キャッシュ使用時/不使用時の readCount の初期値とその上限値は表 3 のように設定した．

表 3　readCount 各種設定

readCount	初期値	上限値
キャッシュ使用時	10	1000
キャッシュ不使用時	100	10000

5.5 実験結果

目標スループットを上げていき，実スループットが追いつかなくなった時点で終了とする．

データベースノードは 1 ノードと 3 ノードで実験を行った．FARP が制御する遅延のしきい値は，1 ノードでの実験では 5 ms, 10 ms, 15 ms の三つを設定し，3 ノードでの実験では 10 ms を設定した．また，比較のため，FARP を使用せずに直接アクセスする場合と遅延に関係なくキャッシュを常に使用した場合の実験結果も示す．

5.5.1 データベースノード一台による実験

図 1 は Update–Heavy ワークロードを課し，スループットを変更することでデータベースへの負荷を変えた時の読み出し遅延を示したものである．その際のデータ新鮮度を図 2 に示す．しきい値ごとに遅延が維持されていることが確認できる．また，図 3，4 は Read–Heavy ワークロードを課した際の結果である

Update–Heavy ワークロードと Read–Heavy ワークロードともに，しきい値が低いほどデータ新鮮度の低下は大きいが，実スループットが目標スループットに追いつかなくなった時のデータ新鮮度はどのしきい値も同じ値であった．

5.5.2 データベースノード三台による実験

図 5 は Update–Heavy ワークロードを課し，スループットを変更することでデータベースへの負荷を変えた時の読み出し遅延を示したものである．その際のデータ新鮮度を図 6 に示す．設定したしきい値に従い遅延が維持されていることが確認できる．また，図 7，8 は Read–Heavy ワークロードを課した際の結果である．

.

5.6 考　　察

図 1 から図 8 より，直接アクセスの遅延がしきい値を超えたあたりを境としてデータ新鮮度は低下し始め，負荷をより大きくした場合は，さらにデータ新鮮度を犠牲にして遅延を維持している．したがって，遅延を維持できるスループットには限界があるものの，FARP により読み出し性能とデータ新鮮度のトレードオフが調整されていることがわかる．

以下では各種実験データについての考察を行う．

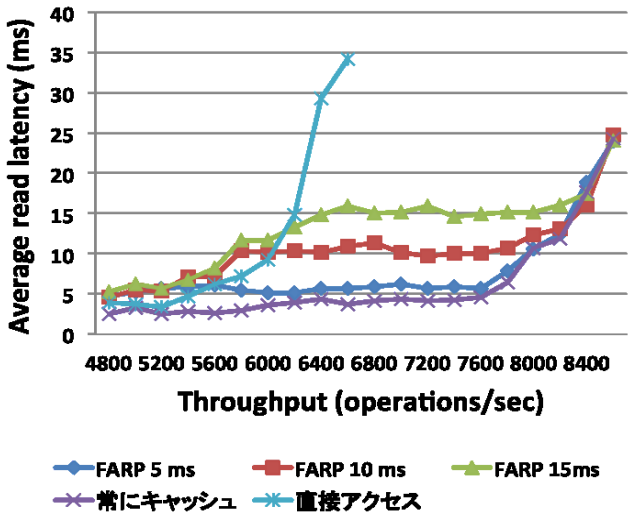

図 1　Update–Heavy ワークロードでの読み出しアクセスの遅延比較

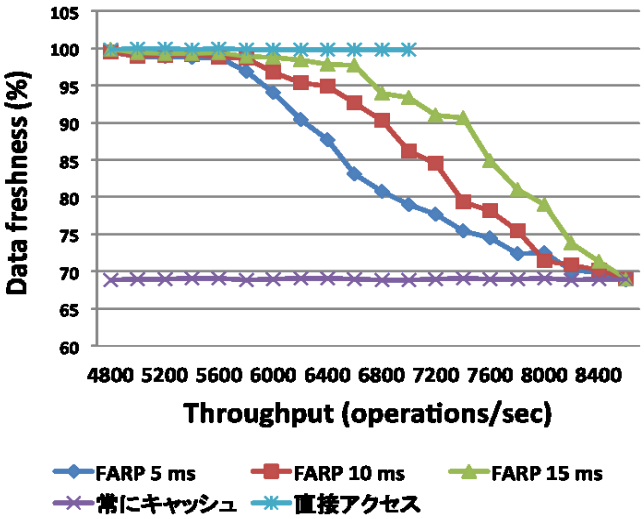

図 2　Update–Heavy ワークロードでのデータ新鮮度

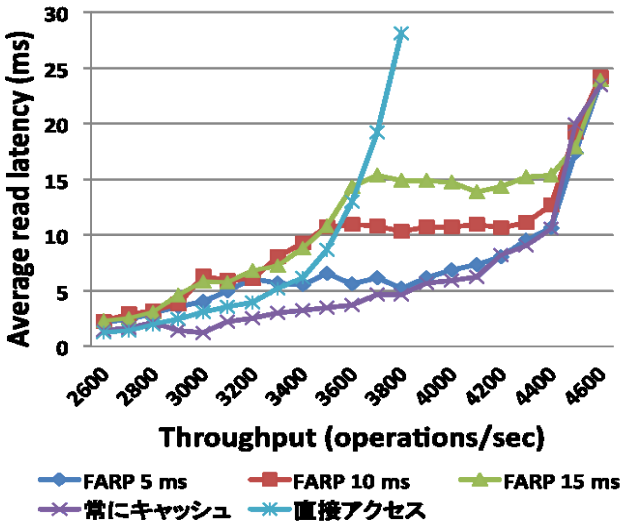

図 3　Read–Heavy ワークロードでの読み出しアクセスの遅延

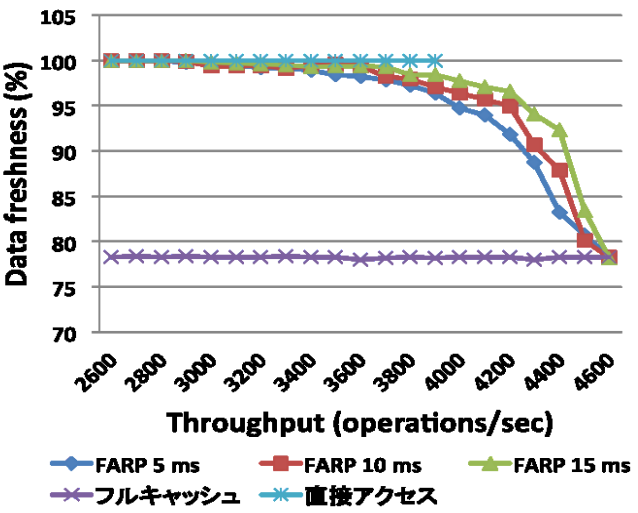

図 4　Read–Heavy ワークロードでのデータ新鮮度

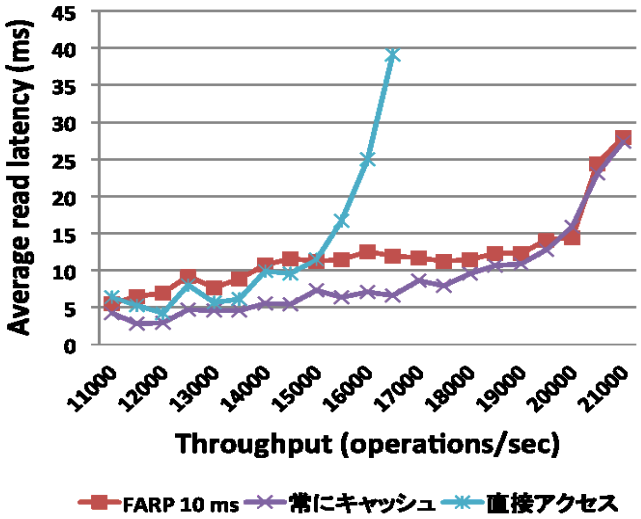

図 5　3 ノードクラスタにおける読み出しアクセスの遅延（Update–Heavy）

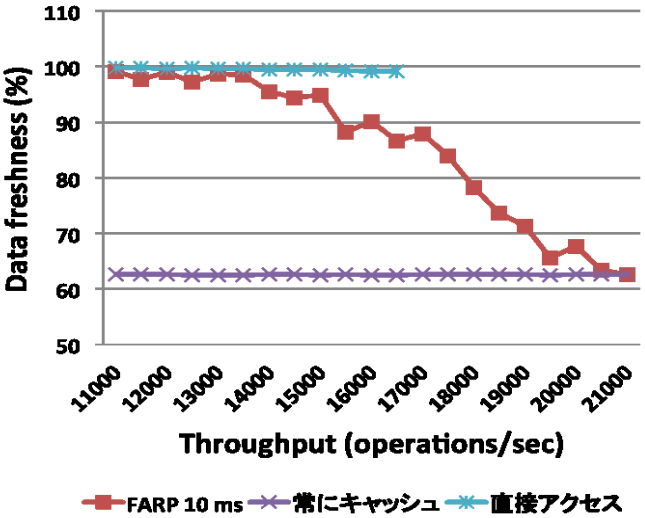

図 6　3 ノードクラスタにおけるデータ新鮮度（Update–Heavy）

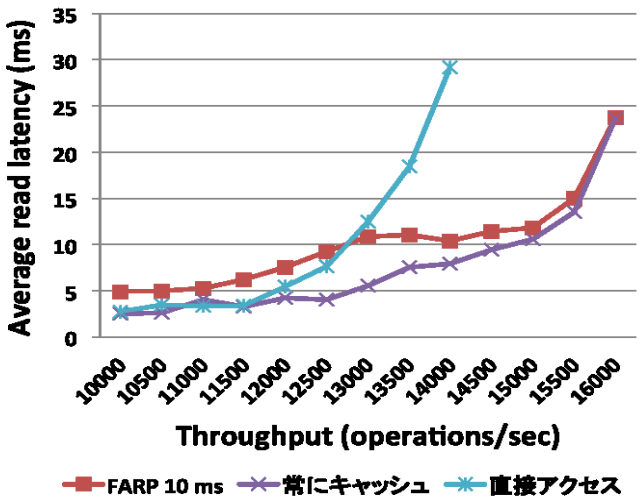

図 7　3 ノードクラスタにおける読み出しアクセスの遅延（Read–Heavy）

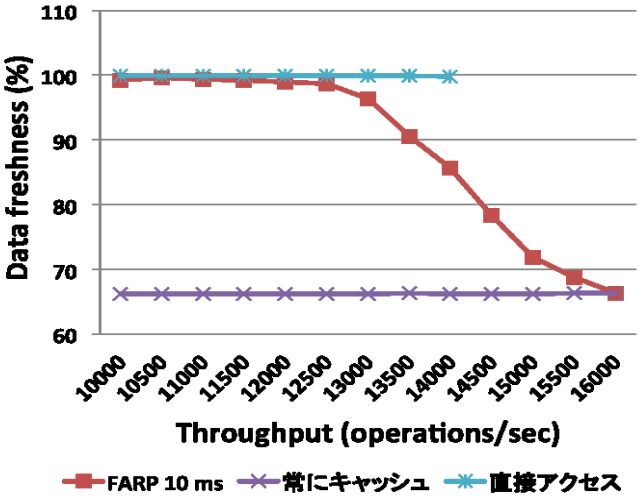

図 8　3 ノードクラスタにおけるデータ新鮮度（Read–Heavy）

5.6.1　読み出しアクセスの遅延

直接アクセスの遅延がしきい値を超えるまでは，Read–Heavy ワークロードと Update–Heavy ワークロードのどちらも，FARP を使用した遅延が直接アクセスの遅延を上回っている．これは，読み出し処理全てが FARP を経由して行われていることで，サーバ間の通信遅延が余分に加わっているためと考えられる．

スループットが一定の値を超えるまで，FARP はしきい値に従い遅延を維持している．維持された遅延がしきい値を少しだけ超えているのは，FARP がデータベースからの読み出し遅延のみを監視し，キャッシュ使用/不使用の切り替え判断にクライアントと FARP 間の通信遅延を考慮していないためと考えられる．

Update–Heavy ワークロードと Read–Heavy ワークロードともに，遅延を維持できるスループットには限界がある．FARP はデータの保存にメモリを用いているため，FARP が保持できるデータ量はデータベースに保存されているデータ量に比べて小さい．また，FARP はキャッシュにヒットしなかった場合，クライアントへ返すデータを取得するためにオリジナルデータベースへアクセスする．そのため，スループットを増加させると単位時間あたりのキャッシュヒットミス率が高くなり，単位時間あたりのオリジナルデータベースへのアクセスも増加する．つまり，負荷が高まりメモリによるキャッシュを常に使用している状態でも，オリジナルデータベースへの読み出しアクセスの削減には限界がある．FARP により限界まで削減した，オリジナルデータベースへの読み出しアクセス量がその処理能力限界に達した時，遅延を維持できなくなり，遅延が大きく増加している．

より高い負荷に対応するための手法

さらに高い負荷に対して低遅延を維持するためには，オリジナルデータベースへのアクセスをさらに削減する必要があり，そのためには FARP のキャッシュ容量を増やす必要がある．その単純な方法は，FARP が使用できるメモリ容量を増やすことである．例えば，複数サーバのメモリを使用することが考えられる．それを実現する設計の具体案としては，Memcached [6] [9] など，既存の分散型メモリキャッシュシステムを複数のサーバで稼働させ，FARP が自身のメモリに加え，Memcached にもデータを保存する設計が考えられる．

しかし，メモリの容量を増やすことでキャッシュヒット率を上げることは出来るが，データベースへのアクセスを完全に削減することはできない．したがって，メモリ容量を増やすだけでは，やはり FARP による遅延維持には限界がある．そこで，キャッシュにディスクを使用することを考える．

データベース管理システムは書き込み性能か読み出し性能のどちらかに特化した設計がなされている．我々の実験で使用している Apache Cassandra は書き込み性能に特化しているが，一方，MySQL [7] などは読み出し性能に特化している．オリジナルデータベースのデータベース管理システムとして書き込み性能に特化したものを使用している場合，FARP で読み出し性能に特化したデータベース管理システムを使用することで，ディスクを使用したキャッシュを行うことが出来る．例えば，オリジナルデータベースのデータを，MySQL を使用して FARP のディスクに複製する．メモリ上のデータ同様，FARP はディスク上のデータが古いデータであることを許容し，負荷に余裕があるときのみ更新を行う．オリジナルデータベースへの負荷が増加し，メモリによるキャッシュのみでは遅延を維持できなくなった時，FARP のディスクからデータを読み出す．すると，古いデータを読み出す可能性は高まるが，オリジナルデータベースにアクセスするより早く読み出しを行うことができ，また，オリジナルデータベースへのアクセスを完全に削減することができる．したがって，維持できる遅延の下限値は FARP で使用するデータベース管理システムの性能によるものの，FARP による遅延維持の限界を無くすことが出来ると予想される．

5.6.2 データ新鮮度

Update–Heavy ワークロードと Read–Heavy ワークロードのどちらも，スループットを大きくするほどデータ新鮮度が低下している．これは，データベースへの負荷が増えた際に，遅延を維持するためにキャッシュをより頻繁に使用する必要があるためである．また，遅延をより低く維持するためには，より頻繁にキャッシュを使用する必要があるため，しきい値が低いほどデータ新鮮度の低下が大きい．

直接アクセスの遅延がしきい値を超えた時を境として，データ新鮮度の低下が始まっている．特に Update–Heavy ワークロードは，その境におけるデータ新鮮度の低下が顕著に現れている．Update–Heavy ワークロードはオリジナルデータの更新頻度が高いため，キャッシュ上のデータが古くなる可能性が高く，そのため，キャッシュの使用が少量でもデータ新鮮度が急激に低下していると考えられる．逆に，Read–Heavy ワークロードはオリジナルデータの更新頻度が低く，キャッシュ上のデータが最新である可能性が高いため，キャッシュ使用が少ないうちはデータ新鮮度の低下が小さい．

スループットを限界まで大きくしたときに，異なるしきい値におけるデータ新鮮度が同じ値に収束しているのは，しきい値に関係無くキャッシュをほぼ常に使用しているためである．

5.6.3 複数ノード

データベースの構成ノードを三台にし，FARP を三台稼働した場合も，1 ノードの場合と同様の結果が得られた．したがって，データベースノードが増えた数だけ FARP を稼働する台数を増やすことで，Update–Heavy ワークロードなら約 30 %，Read–Heavy ワークロードなら約 20 % 程度までは，スループットが上昇しても遅延を目標値に維持できると予想される．

今回，データベースノード一台につき一台の FARP を稼働させたのは，メモリ容量に制限され，FARP がキャッシュできるデータ件数がオリジナルデータベースの扱うデータ件数に比べて少ないからである．データベースノードが増え，オリジナルデータベースが扱うデータ件数が数倍に増えたにも関わらずキャッシュ容量が変わらない場合，キャッシュヒットミス率の増加に伴いオリジナルデータベースへのアクセスも増加し，FARP により遅延を維持できるスループットの限界値が減少すると考えられる．5.6.1 節で述べた手法により，FARP のキャッシュ容量を大幅に増やすことが出来れば，データベースノードが増えた場合も FARP を稼働させる台数を増やすことなく，高スループットに対応できると予想される．

6. 関 連 研 究

Bolt–on causal consistency [3] は我々と同じ設計方針で，データベースが担保する整合性のレベルを，最も弱い eventual consistency から一段階上の causal consisntency へ向上させるシステムの研究である．我々と同じ設計方針であるため，コアなアルゴリズム部分はデータベース管理システムから独立している．そのため，そのアルゴリズムを我々の提案手法に取り入れる事も可能である．

データベースの負荷やアクセス性能を監視し，それに従いデータベースの保証するサービスレベルを変更するシステムは研究されている．Terry らの手法 [4] はその一例である．データベースが担保する整合性にレベルを設け，データベースからの読み出し遅延に応じて担保する整合性レベルを変更する．データ新鮮度の向上は整合性の向上と捉えることができるため，整合性と遅延のトレードオフを考慮する点は本研究と共通している．しかし，データベース管理システムの機能としてシステムが組み込まれており，strong consisntency を担保するために単一障害点が存在するなど，本研究とはその設計方針が異なる．

また，整合性のベンチマーク方法も研究されており，Rahman らの手法 [8] はその一つである．この研究も，ベースとなるベンチマークツールとして YCSB を用いている．

7. まとめと今後の課題

キャッシュの使用量を調整することでデータ新鮮度とデータベースのアクセス性能のトレードオフを調整する手法を提案し，提案手法に基づき設計・実装したキャッシュ機構を評価した．

実験により，我々の設計したキャッシュ機構を用いることで，対応できる負荷の大きさに限界はあるものの，読み出し性能とデータ新鮮度のトレードオフを調整できることを確認した．

5.6.1 節で述べた手法を実装することで FARP のキャッシュ容量を増やし，その性能評価を行いたい．

また，今回の実装では，読み出し遅延の平均を維持することは出来るが，個々の読み出し遅延をすべてしきい値以下に維持することは出来ない．したがって，キャッシュ使用の調整により 99 パーセンタイルの遅延を維持することは課題である．

謝辞

本研究は JSPS 科研費 25700008，24650025 の助成を受けたものである．

文　　献

[1] The Apache Software Foundation: Apache Cassandra, http://cassandra.apache.org/.

[2] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears.: Benchmarking Cloud Serving Systems with YCSB, Proc. SOCC '10, pp. 143–154, 2010.

[3] Peter Bailis, Ali Ghodsi, Joseph M. Hellerstein, and Ion Stoica.: Bolt-on Causal Consistency, Proc. SIGMOD '13, pp. 761–772, 2013.

[4] Douglas Terry, Vijayan Prabhakaran, Rama Kotla, Mahesh Balakrishnan, Marcos K. Aguilera, and Hussam Abu-Libdeh.: Consistency-based Service Level Agreements for Cloud Strage, Proc. SOSP '13, pp. 309–324, 2013.

[5] Avinash Lakshman, and Prashant Malik.: Cassandra - A Decentralized Structured Storage System, Proc. LADIS '09, 2009.

[6] Brad Fitzpatrick.: Distributed Caching with Memcached, Linux Journal, vol. 2004, no. 124, p. 5, 2004.

[7] Oracle Corporation: MySQL, http://www.mysql.com/.

[8] Muntasir R. Rahman, Wojciech Golab, Alvin AuYoung, Kimberly Keeton, Jay J. Wylie.: Toward a Principled Framework for Benchmarking Consistency, Proc. Workshop on Hot Topics in System Dependability, 2012.

[9] Jure Petrovic.: Using Memcached for Data Distribution in Industrial Environment, Proc. IEEE Computer Society, pp. 368–372, 2008.