超個体型データセンターを目指した分散協調クエリキャッシュ構想

坪内 佑樹[†1,a)]　松本 亮介[†1]

概要：インターネットが普及し，Web サービスが広く利用されるようになるにつれて，利用者からのコンテンツ配信の速度への要求が高まっている．クラウドコンピューティングを利用する場合，利用者に対して高速に応答するには，全体のレイテンシの主要な部分を占めるネットワークレイテンシを低減することが重要となる．そこで，コンピューティングの利用者の近傍で要求を処理することにより，レイテンシを最小化する CDN(Content Delivery Network) が利用されており，近年では，IoT，スマートシティやクラウドゲーミングのような低レイテンシを要求するアプリケーションに向けて，クラウド上の処理を利用者近傍のエッジへと移動するエッジコンピューティングが研究されている．このような背景により，我々は，クラウドを中心に小規模データセンター，あるいはデータセンターと呼べないまでも小型のラック群があらゆる場所に分散していく超個体型データセンターの時代になっていくと考えている．超個体型データセンターを目指す上で，各データセンターを抽象化して利用するために，各データセンターの場所や規模を意識せずに，透過的かつ高速にデータを読み書きできる必要がある．本研究では，Web アプリケーションのコンテンツ配信の性能を向上させるために，データベースへのクエリ結果のキャッシュを分散したデータセンター間で一貫性を保ちつつも共有し，データセンター間のレイテンシを意識せずに，書き込み時と読み込み時の応答性能のトレードオフを解決するための基盤の設計構想を提案する．

A Concept of Distributed Coordinated Query Caching Toward Super-Organic Data Center

Yuuki Tsubouchi[†1,a)]　Ryosuke Matsumoto[†1]

1. はじめに

インターネットが普及し，Web サービスが広く利用されるようになり，一般の人々が文章や画像，動画といったコンテンツを生み出し，家族や友人，その他の人々と共有することが当たり前の時代となってきている．さらには，スマートフォンの普及により，インターネットはより身近なものになってきている．Web サービスが日常的に利用されるにつれて，利用者数は増大し，利用者からの Web サービスの応答性能への要求も高まっている．文献 [1] によると，40%以上の Web サービスの利用者はページの読み込みを 3 秒以上待てないという調査結果が報告されている．そこで，Web サービスを開発し，運営する Web サービス事業者および多くの Web サイトを管理する Web ホスティングサービス事業者にとって，利用者数を増大させるために，Web サービスの応答性能を向上することが重要となっている．

Web サービスの提供のためにクラウドコンピューティングを利用する場合，中央集権のデータセンター上でコンピューティングを実行するために，応答するまでのレイテンシは次のように分解できる [2]．まず，利用者の端末上の処理のためのレイテンシ，次に，利用者からデータセンター上のサーバまでの往復のネットワークレイテンシ，最後に，データセンター上のサーバの処理のためのレイテンシがある．文献 [2] の研究では，全体のレイテンシが 100ms

[†1] さくらインターネット株式会社 さくらインターネット研究所
SAKURA Research Center, SAKURA Internet Inc., Ofukatyo, Kitaku, Osaka 530-0011 Japan
[a)] y-tsubouchi@sakura.ad.jp

のうち，利用者とデータセンター間のネットワークレイテンシが 80ms を占めるとされている．したがって，利用者に対して高速に応答するには，全体のレイテンシの主要な部分を占めるネットワークレイテンシを低減することが重要となる．

そこで，利用者の近傍であるネットワーク端（以降，エッジとする）にて予めキャッシュとしてコンテンツを配置しておき，利用者はそのコンテンツにアクセスすることで高速にコンテンツを取得可能な CDN(Content Delivery Network)[3] を活用する．CDN により，クラウド上のデータセンターまで到達することなく，コンテンツを配信できるため，ネットワークレイテンシが低減される．近年では，一度キャッシュされたデータを破棄するまでの処理時間が高速化したことにより，画像や動画といった静的コンテンツ以外に，利用者の行動に応じて，中身が変化する動的コンテンツのキャッシュにも CDN を利用することがある [4]．さらには，単なるコンテンツを配信するだけでなく，CDN のエッジサーバ上で要求に対して任意の処理を実行できるように，エッジサーバに任意のアプリケーションロジックを配置することが可能となる CDN Edge Worker[5], [6], [7] が登場している．このように CDN のエッジサーバの活用の幅が広がっている背景に加えて，IoT(Internet of Things)[8]，スマートシティ，クラウドゲーミング [2] などのリアルタイム性を要求される領域において，非力なエンドデバイス，利用者近傍のホームゲートウェイやネットワークルーター，小規模データセンターといったようなエッジのコンピューティング資源を活用するエッジコンピューティング [9], [10], [11] が研究されている．さらに，松本らの研究 [12] では，クラウドに集中したコンピューティング資源はそのままに，クラウドを中心にエッジとしての小規模データセンター，あるいはデータセンターと呼べないまでも小型のラック群があらゆる場所に分散していく時代になっていくという構想が報告されている．さらに，このような時代のデータセンターのあり方を分散型データセンターと定義し，分散されたデータセンターを抽象化する OS の層を分散型データセンター OS と定義している．分散型データセンターにおける各データセンターは単に分散するだけでなく，協調し，全体としては一つとして見えるように動作することから，名称をより適切なものとするために，本稿では，分散型データセンターの命名を改め，超個体型データセンターとする．超個体型データセンターを目指す上で，各データセンターを抽象化して利用するために，各データセンターの場所や規模を意識せずに，透過的かつ高速にデータを読み書きできる必要がある．

分散したデータセンター上のデータを透過的に扱うためには，いつ誰がどこでクラウドを利用しても，データがどのデータセンターに配置されているかを意識せずに利用できる必要がある．データの配置を意識せずに利用するためには，各データセンター上のデータが一貫している必要がある．しかし，クラウド内におけるサーバの分散とは異なり，データセンターが分散した結果，データセンター間のレイテンシを考慮する必要がある．データの一貫性を維持するために，データセンター間で同じデータを共有しようとすれば，書き込み時のデータの同期時間が大きくなり，逆に特定のデータセンターのみにあるデータを読み出そうとすれば，読み出し時のデータの転送時間が大きくなる．したがって，データセンター間のデータの一貫性を維持しつつ，書き込み時と読み込み時のそれぞれの応答性能はトレードオフとなる．エッジを利用した Web コンテンツ配信を検討している先行研究の Ferdinand[13] では，動的コンテンツの配信を高速化するために，オーバーレイネットワークを利用し，データベースへのクエリの結果キャッシュを複数のサーバが共有する手法が提案されており，サーバ間のレイテンシの大きな環境で実験したときに，読み込み時のデータの転送時間が性能ボトルネックとなることが課題となっている．さらに，Ferdinand では，一時的なエッジの故障によりレイテンシが増大したときに，当該エッジがもつキャッシュエントリにアクセスする場合に応答性能が低下する可能性がある．

本研究では，Web アプリケーションのコンテンツ配信の応答性能を向上させるために，データベースへのクエリ結果のキャッシュを分散したデータセンター間で一貫性を保ちつつも共有し，書き込み時と読み込み時の応答性能のトレードオフを解決するための設計構想を提案する．具体的には，クォーラムシステムを基に，分散したすべてのデータセンターにキャッシュを同期せずに，クラウドの近傍にあるデータセンターにのみ同期し，読み込み時はデータが同期されている近傍のデータセンターから転送する．さらに，一時的なエッジの故障に対して，クラウド上のデータベースサーバが故障を検知し，当該エッジをクラスタから除外することにより，故障の影響を最小限に留める手法を提案する．本手法により，すべてのエッジで同一のキャッシュを一貫性を維持しながら共有する場合と比較し，読み込み性能が低下することはあるものの，書き込み性能に対するデータセンター間レイテンシの影響を小さくできる．

本稿の構成を述べる．2 章では，Web アプリケーションをエッジへ展開するときの課題について議論する．3 章では，分散協調キャッシングのための関連技術について整理する．4 章では，本研究での提案を述べる．最後に，5 章でまとめとする．

2. Web アプリケーションのエッジへの展開

2.1 データセンターを分散したときのデータ抽象

分散したデータセンター上のデータを抽象化し，透過的に扱うために，データセンター間のレイテンシをいかに隠

蔽し，応答性能とデータ一貫性を両立するかが重要となる．強いデータ一貫性を維持する前提で，データセンター間のレイテンシが応答性能に影響するのは，次の 2 通りのネットワーク転送である．まず，あるデータを特定のデータセンターのみに配置し，読み出し時にデータセンターを特定しネットワーク転送する（以降，オンデマンド転送とする）場合である．2 つ目に，全てまたは主要な部分の複数のデータセンターにデータの複製を配置し，読み出し時に自身のデータセンターまたは近傍のデータセンターからネットワーク転送する（以降，レプリケーション転送とする）場合である．いずれの手法においても，データの読み出し時またはデータの複製の配置時において，データのネットワーク転送時間を要する．オンデマンド転送の場合は，利用者近傍のデータセンターから遠いデータセンターから転送しなければならない場合，応答性能が低下する．一方で，レプリケーション転送の場合は，各データセンター上のデータの一貫性を保持するのであれば，書き込み時のデータ同期時間が長くなり，同期待ちのために応答性能が低下するという課題がある．データの一貫性を緩めるのであれば，非同期でレプリケーションすることにより，応答性能の低下を防げるかわりに，ある瞬間において，エッジ上のアプリケーションが古いデータを読み込む可能性があることを許容する必要がある．このように，楽観的な一貫性モデルを採用する場合，アプリケーション開発者が，古いデータを読み込むことを考慮した上で，アプリケーションの処理を記述しなければならないという課題がある．

2.2 Web アプリケーションの構成パターン

伝統的な Web アプリケーションは，Web サーバ，アプリケーションサーバ，データベースサーバの 3 階層 [14] により構成される．この 3 階層の各構成要素をエッジとクラウド環境に展開すると，組み合わせパターンは次の 3 つとなる．

(a) Web サーバ，アプリケーションサーバ，データベースサーバをすべてエッジに配置．
(b) Web サーバとアプリケーションサーバをエッジに配置し，データベースサーバをクラウドに配置．
(c) Web サーバをエッジに配置し，アプリケーションサーバとデータベースサーバをクラウドに配置．

図 1 および図 2 に示すパターン (a) については，エッジ環境にて各要素をすべて配置することにより，エッジ内で処理が完結するため，利用者への応答時間を短縮できる．しかし，データベースサーバを分散することにより，データベース間の書き込み競合が発生しやすくなり，各データベースサーバの一貫性の制御が複雑となる．書き込み競合を避けるために，図 2 に示すように，各エッジ上のアプリケーションサーバがクラウド上のデータベースサーバのみに書き込み，ローカルのレプリカデータベースから読み出すように構成し，読み込みクエリのみを高速化する手法もある．つまり，クラウド上のデータベースサーバを単一のリーダー（マスタとも呼ぶ）として，エッジ上のデータベースサーバをフォロワー（スレーブとも呼ぶ）とし，フォロワーには読み込みクエリのみを向ける．しかし，一貫性を強める場合はクラウドとエッジ間のレイテンシの分だけ同

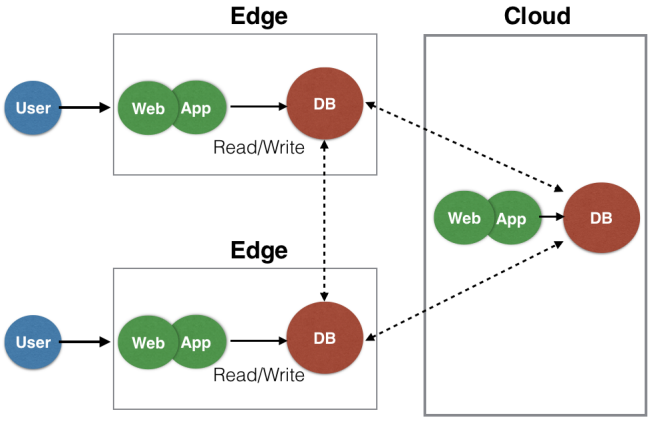

図 1: パターン (a): リーダーレス

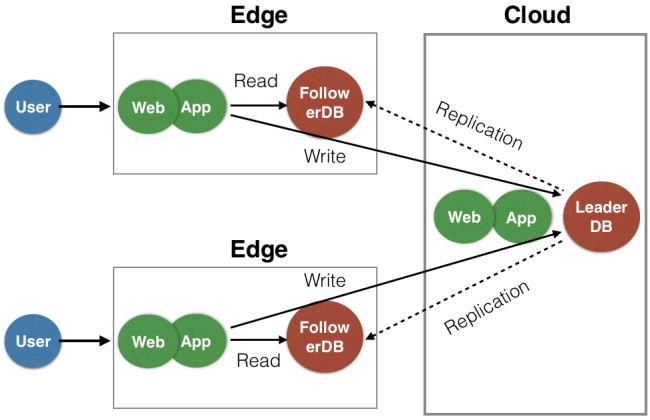

図 2: パターン (a): シングルリーダー

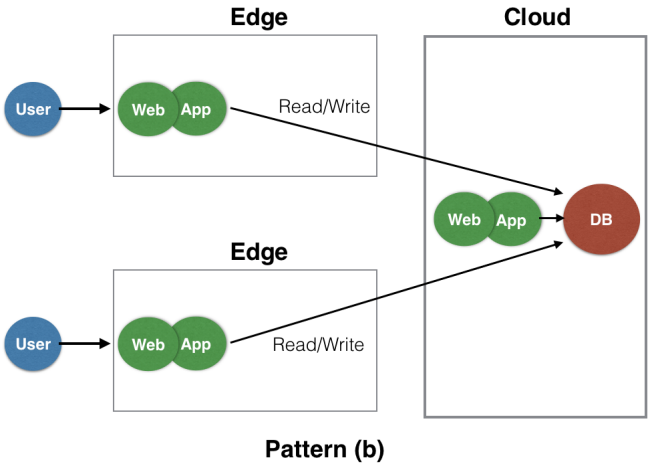

図 3: パターン (b)

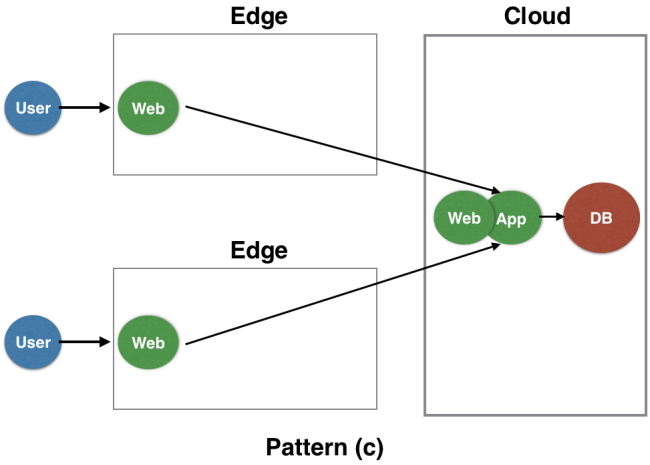

図 4: パターン (c)

期待ちが発生し，非同期にレプリケーションすることによりリーダー・フォロワー間かつフォロワー間の一貫性が緩くなり，に関する問題が発生することがある．

超個体型データセンターのように，クラウド内通信と比較しデータセンター間のレイテンシの大きい環境では，書き込みが全ノードに伝搬する前に別のノードが次の書き込みを開始する可能性が高くなるため，一貫性を維持するためにキャッシュの更新時に待ち時間を要するはずである．強めの一貫性モデルを採用すると，全体のスループットが低下する恐れがあるため，性能を優先するのであれば，結果整合性などの弱い一貫性モデルを採用する必要がある．このような一貫性と性能のトレードオフに対して，データベースミドルウェアのみで解決することは難しく，アプリケーション仕様やコードにて一貫性の考慮が必要となる．

図 3 に示すパターン (b) については，アプリケーションサーバとデータベースサーバ間のレイテンシが他のパターンと比較し大きいという性質がある．アプリケーションサーバが動的コンテンツを生成したり，API(Application Programming Interface) の処理をするには，1 回以上データベースに問い合わせることになる．したがって，利用者と Web サーバ間で HTTP のリクエストとレスポンスを 1 往復する処理の間に，アプリケーションサーバからデータベースサーバへのクエリと結果取得の往復が少なくとも 1 回以上あるため，クラウドのみのパターンと比較し，却って応答が遅くなる可能性がある．

図 4 に示すパターン (c) については，Web サーバとアプリケーションサーバ間のレイテンシが大きいという性質がある．パターン (c) において，HTTP リダイレクトや静的コンテンツの配信など，Web サーバのみで完結する処理は，利用者の近傍で処理が完結するため，応答性能が高くなる．また，Web サーバとアプリケーションサーバ間の接続を永続化することにより，TCP の 3-way ハンドシェイクのためのラウンドトリップ時間を抑えられる．Web サーバとアプリケーションサーバ間は HTTP の往復は 1 回となるため，パターン (b) のように応答速度が低下することはない．しかし，アプリケーションサーバが Web アプリケーションとしての処理の大部分を実行するという前提から，エッジの利点を生かせていない構成といえる．

3. 分散協調キャッシュの関連技術

キャッシュの一貫性を保持しつつ，どのようにしてオンデマンド転送とレプリケーション転送のレイテンシの影響を最小化するかという観点で，分散協調キャッシュの関連技術を整理する．

3.1 オブジェクトキャッシュ

オブジェクトキャッシュはアプリケーションが任意のオブジェクトをキャッシュするための機構である．オブジェクトキャッシュのためのデータストアとして，アプリケーションプロセスがネットワーク通信して利用するネットワーク型の分散キャッシュシステムと，アプリケーションプロセスと同一のメモリ空間にキャッシュデータを保持する組み込み型の分散キャッシュシステムがある．いずれもキャッシュシステムにおいても，キャッシュのデータストアとして，キーバリューストア [15] が利用されることが多い．

ネットワーク型の分散キャッシュシステムとして，Memcached[16] や Redis[17] が広く利用されている．これらのキャッシュシステムを複数のキャッシュノードへ負荷分散するために，ハッシュ法を利用することにより，オブジェクトのキーと分散先のキャッシュノードを紐づけし，オブジェクト単位で分散させてキャッシュノードへデータを配置する．この手法は，キャッシュノードの追加または削除時に，オブジェクトのキーとノードの紐づけが変更され，大部分のキーがキャッシュミスし，性能が低下するという課題がある．そこで，オブジェクトの配置アルゴリズムとして，ノードの増減時に，オブジェクトのキーとノードの割り当てをなるべく変更せずに分散するためにコンシステントハッシュ法 [18] が利用される．

groupcache[19] は組み込み型のオブジェクトキャッシュであり，Go 言語 [20] のライブラリとして実装されている．groupcache は複数のノードが協調して一つのキャッシュ空間を作成可能であり，ローカルノードに当該キャッシュがなければ，ピアノードへアクセスし，ピアノード上のキャッシュを取得するようになっている．さらに，groupcache はフォールバック後に確率的にローカルノードのキャッシュにデータを書き込み，次回以降当該オブジェクトのキャッシュはローカルノードで取得可能となる．また，ピアノードはコンシステントハッシュ法により選択されるため，ローカルでのキャッシュミス時にすべてのピアにフォールバックしなくてよい．groupcache は更新と削除をサポートしておらず，LRU(Least Recently Used) によりキャッシュ

が破棄されないかぎり，普遍のデータを格納する．

3.2 クエリリザルトキャッシュ

データベースミドルウェアの層でキャッシュする場合，データベースへのクエリの結果をキャッシュするクエリリザルトキャッシュを利用する．

MySQL[21] のクエリリザルトキャッシュは，データベースサーバ自体がキャッシュ機構をもつため，追加のソフトウェアなしにクエリリザルトキャッシュを利用できる．テーブルに更新があれば，当該テーブルの変更が結果に影響するクエリのキャッシュを破棄するため，テーブルとキャッシュ間で一貫性を維持する．

ProxySQL[22] は，MySQL プロトコルを解釈するプロキシサーバであり，クエリリザルトキャッシュ，クエリルーティング，MySQL サーバのフェイルオーバといった機能をもつ．ProxySQL は独立したプロセスとして起動するプロキシであることから，ProxySQL のクエリリザルトキャッシュは，MySQL サーバよりもよりフロントエンドに近い位置に配置することが可能である．例えば，アプリケーションサーバと同一のホスト上に配置することにより，アプリケーションサーバと MySQL サーバ間の RTT を削減できる．しかし，TTL(Time to Live) を利用してキャッシュを無効化するため，アプリケーションに対して古くなったデータを返す可能性がある．また，ProxySQL は，複数のピアノードと協調してキャッシュデータを同期したり，キャッシュプールを共有することはない．

DBProxy[23] は，エッジ上に配置された各アプリケーションサーバが，中央のデータベースサーバに対するクエリ結果をキャッシュする．クエリ結果が他のクエリによりキャッシュされたデータによる構築できるのであれば，未キャッシュのクエリの結果を返すという巧妙なクエリ処理が可能である．DBProxy は，中央集権的に一貫性を管理することにより，各キャッシュは中央データベースに対してコヒーレントとなっている．しかし，クエリキャッシュを利用する複数のステートメントを含むトランザクションに対応できるようなトランザクションの観点での一貫性は保証されない．

Ferdinand[13] は，分散されたデータベースクエリリザルトキャッシュによる，動的コンテンツ配信のための協調動作するプロキシである．トピックベースの Publish/Subscribe モデルにより，キャッシュの一貫性を維持しつつ，分散ハッシュテーブルによるオーバーレイネットワーク内で必要なキャッシュを発見し，オンデマンド転送する．また，DBProxy と同様に，複数のステートメントを含むトランザクション向けの一貫性を緩めている．実験の結果，キャッシュミスすると中央データベースサーバから読み出す単純手法と比較し，3 倍程度の性能向上があった一方で，エッジ上のキャッシュノード間とキャッシュノードと中央データベースの間のレイテンシを大きくした環境では，レイテンシが増加すると単純手法よりもスループットが低下するという結果を示した．

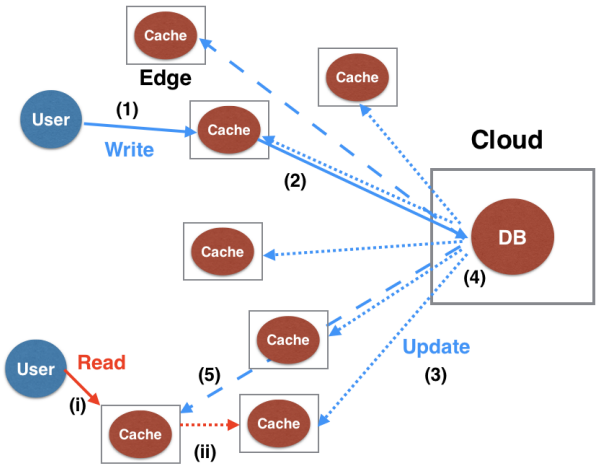

図 5: クォーラムシステムを基にしたキャッシュ管理

DBProxy または Ferdinand のような各アプリケーションサーバが一貫したキャッシュを持つためのアプローチにおいて，一貫性を保つためにキャッシュの更新時に同期的なレプリケーションによる更新待ち時間，またはデータ読み出し時のオンデマンド転送による別ノードから転送時間が必要となる．

4. 提案手法

Web アプリケーションのデータ読み込み性能を向上させるために，アプリケーション開発者に追加の負担をかけないという要件のもとに，強い一貫性を採用する必要がある．さらに，利用者とクラウドの間のネットワークレイテンシを低減させるために，データベース層でクエリ結果をエッジ上でキャッシュする．先行手法の課題となるキャッシュ更新時の同期待ち時間または読み込み時の転送時間を低減させるために，クラウドからネットワーク的距離の小さいエッジ群には同期的に更新を伝搬させ，残りのエッジには非同期に更新を伝搬させ，読み込み時には同期的に更新されているエッジのどれか一つからキャッシュを取得するクォーラムシステムを基にしたキャッシュ管理アーキテクチャを提案する．本手法により，キャッシュの更新時の同期待ち時間をネットワーク的距離の大きいエッジに律速されることなく，クラウドよりも近傍にあるエッジからキャッシュを読み込むために，同期待ち時間または転送時間を低減できる．

4.1 クォーラムシステムを基にしたキャッシュ管理

図 5 にキャッシュ管理のアーキテクチャを示す．各エッジのキャッシュが更新されるまでの処理の流れは次のようになる．

(1) 利用者が発行する書き込み要求を利用者の近傍のエッ

ジが受け付ける．
(2) クラウド上のデータベースサーバへ更新クエリが転送される．
(3) データベースサーバからクラウド近傍のエッジ群へ，更新クエリと同一トランザクション内で同期的に更新を通知する．
(4) データベースサーバは更新通知後にトランザクションをコミットする．トランザクションがアボートされた場合は，(3) で通知した更新を無効化するために，キャッシュの無効化を通知する．
(5) データベースサーバは非同期に (3) で通知したエッジ以外のすべてのエッジに更新を通知する．

次に，各エッジのキャッシュが利用者から読み込まれるまでの処理の流れは次のようになる．
(i) 利用者が発行する読み込み要求を利用者の近傍のエッジが受け付ける．
(ii) 近傍のエッジは，自身がクラウドの近傍のエッジであるかそうでないかを参照し，そうであれば自身のキャッシュを参照する．そうでなければ，クラウド近傍のエッジのうち，自身からネットワーク距離の近いものを選択し，当該エッジ上のキャッシュを参照する．

クラウドの近傍のエッジ群を判定するために，あらかじめ各エッジとクラウド間ネットワークレイテンシを計測し，データベースサーバが結果を保持しておく．システム管理者が設定したネットワークレイテンシの閾値内のエッジを近傍のエッジとみなし，データベースサーバは近傍のエッジの判定結果を保持しておくことにより，近傍のエッジを認識できる．一方で，各エッジがクラウド近傍のエッジかどうかを認識するために，エッジは近傍のエッジの判定結果をデータベースサーバから通知を受け取る．さらに，各エッジがクラウド近傍のエッジのうち自身からネットワーク距離の近いものを選択するために，あらかじめ自身と他のすべてのエッジとの間のネットワークレイテンシを計測し，保持しておく．

4.2 応答性能を最大化するための適応的なクォーラム調整

提案手法は，システム管理者がネットワークレイテンシの閾値を設定することにより，キャッシュ更新時の同期待ち時間による性能影響と読み込み時の転送時間の性能影響の均衡を保つ．しかし，実際には，書き込み主体のアクセス傾向であれば，同期回数が増加し，同期待ち時間による性能影響が大きくなり，読み込み主体のアクセス傾向であれば，転送回数が増加し，転送時間による性能影響が大きくなる．したがって，アプリケーション単位のアクセス傾向に応じて，ネットワークレイテンシの閾値を調整し，応答性能を向上させることが可能である．

そこで，アプリケーション単位で応答性能を最大化するために，クラウドの近傍と判定するためのレイテンシの閾値を適応的に調整する手法を提案する．具体的には，アプリケーション単位で応答性能を監視しながら，レイテンシの閾値を増減させて，応答性能の変化を確認しつつ，応答性能が向上する方向に閾値を変更していく．

4.3 エッジの故障に対する回復

提案手法では，エッジが故障したときに，エッジが更新通知を受け取れないことにより，後に回復したときに，キャッシュの中身がクラウド上のデータベースの中身と比べて古くなるという課題がある．そこで，更新通知を受け取らなかったエッジをクラスタから除外させ，当該エッジのキャッシュを最新のものと同期した後に再度クラスタへ組み込む手法を提案する．

まず，エッジの故障判定は，データベースサーバの更新通知時にタイムアウトすることにより検知する．次に，故障したエッジをクラスタから除外するために，データベースサーバが当該エッジからクエリを受け付けないようにする．エッジが故障から回復するときに，エッジはデータベースサーバから自身が故障判定されていることを受け取ると，自身のエッジ内の Web サーバへ自身が故障から回復中であることを通知し，Web サーバは利用者から受信した要求を近傍のエッジかまたはクラウドへ転送する．最後に，回復中のエッジは，キャッシュを破棄したのちに，近傍のエッジまたはクラウドからキャッシュを同期し，同期完了後にクラウドから更新通知の受け付けを再開する．

5. まとめ

本論文では，利用者とクラウドの間のネットワークレイテンシを低減させるために，エッジとして分散したデータセンターを活用することを前提に，Web アプリケーションの読み込み性能を向上させるために，アプリケーション開発者に追加の負担をかけずに，強い一貫性を持ちつつも，データベースクエリの結果をキャッシュすることに着目した．その上で，我々は，分散したデータセンター上のデータを透過的に扱える超個体型データセンターに向けて，分散したキャッシュの更新時と読み込み時のデータ転送時間のトレードオフを解決するために，クォーラムシステムを基にしたキャッシュ管理アーキテクチャの構想を提案した．

今後は，従来の P2P オーバーレイネットワークで提案されている手法と本研究との立ち位置を明確にしつつ，提案手法の実装と評価を進めていく．

参考文献

[1] Forrester Consulting: eCommerce Web Site Performance Today: An Updated Look At Consumer Reaction To A Poor Online Shopping Experience 2009.
[2] S Choy, B Wong, G Simon and C Rosenberg: The Brewing Storm in Cloud Gaming: A Measurement Study on Cloud to End-User Latency, *11th Annual Workshop on*

Network and Systems Support for Games, p. 2 2012.
[3] A M K Pathan and R Buyya: A Taxonomy and Survey of Content Delivery Networks, *Grid Computing and Distributed Systems Laboratory, University of Melbourne, Technical Report*, Vol. 4, No. 0, p. 1 2007.
[4] H Beheshti: Extending your application to the edge with Fastly.
[5] Lambda@Edge, https://aws.amazon.com/lambda/edge/.
[6] Cloudflare Workers, https://cloudflareworkers.com/.
[7] Fly Edge Applications, https://fly.io/.
[8] J Lin, W Yu, N Zhang, X Yang, H Zhang, and W Zhao: A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications, *IEEE Internet of Things Journal*, Vol. 4, No. 5, pp. 1125–1142 2017.
[9] B Kashif, K Osman, E Aiman and S U Khan: Potentials, Trends, and Prospects in Edge Technologies: Fog, Cloudlet, Mobile Edge, and Micro Data Centers, *Computer Networks*, Vol. 130, pp. 94–120 2018.
[10] 山口弘純, 安本慶一：エッジコンピューティング環境における知的分散データ処理の実現，電子情報通信学会論文誌 B， Vol. 101, No. 5, pp. 298–309 2018.
[11] M Satyanarayanan: The Emergence of Edge Computing, *Computer*, Vol. 50, No. 1, pp. 30–39 2017.
[12] 松本亮介, 坪内佑樹, 宮下剛輔：分散型データセンター OS を目指したリアクティブ性を持つコンテナ実行基盤技術，情報処理学会研究報告インターネットと運用技術（IOT），Vol. 2019-IOT-44, No. 27, pp. 1–8 2019.
[13] C Garrod, A Manjhi, A Ailamaki, B Maggs, T Mowry, O Christopher and T Anthony: Scalable Query Result Caching for Web Applications, *VLDB Endowment*, Vol. 1, No. 1, pp. 550–561 2008.
[14] X Liu, J Heo and L Sha: Modeling 3-tiered web applications.
[15] R Cattell: Scalable SQL and NoSQL Data Stores, *ACM SIGMOD Record*, Vol. 39, No. 4, pp. 12–27 2011.
[16] Memcached, https://memcached.org/.
[17] Redis, https://redis.io/.
[18] D Karger, E Lehman, T Leighton, M Levine, D Lewin, R Panigrahy: Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on The World Wide Web, *ACM Symposium on Theory of Computing*, Vol. 97, pp. 654–663 1997.
[19] Groupcache, https://github.com/golang/groupcache.
[20] A A Donovan, B W Kernighan: *The Go Programming Language*, Addison-Wesley Professional, 1st edition 2015.
[21] MySQL, http://www.mysql.com/.
[22] ProxySQL, https://proxysql.com/.
[23] K Amiri, S Park, R Tewari, and S Padmanabhan: DBProxy: A Dynamic Data Cache for Web Applications, *International Conference on Data Engineering*, pp. 821–831 2003.