Vol.2013-HPC-140 No.21
2013/8/1

クエリースケジューリングによる分散キーバリューストアの応答性能向上

福田 諭[1]　津邑 公暁[1]　齋藤 彰一[1]　松尾 啓志[1]

概要：ネットワークの高速化やクラウドサービスの普及により，大規模なデータの管理に対する需要が増加している．単一の計算機で大規模なデータ管理をすることは，性能やコスト，拡張性に問題があるため，複数の計算機で分散してデータを管理することが提案されている．複数の計算機でデータを分散して管理するデータストアとして分散 Key Value Store(KVS) がある．分散 KVS はデータを Key と Value という単純な構造で管理するため，容易にスケールアウト可能であるという利点があり，大規模なデータを扱うサービスなどで注目を集めている．分散 KVS では，データの Key を指定することで対応する Value を取得する検索処理が可能である．また分散 KVS の実装によっては，Key の範囲を指定することによりその Key 間に属する Value を取得する検索処理が可能な実装もある．しかし，単一検索と範囲検索が混在する環境においては，高速に実行終了が可能な単一検索命令の実行が待たされてしまい，結果として平均応答時間が増加するという問題がある．そこで本研究では複数の検索クエリーをスケジューリングすることにより，データを要求するクエリーの平均応答時間を短縮させる手法を提案する．提案手法を分散 KVS の Cassandra 上に実装し，評価を行なった結果，クエリーのスケジューリングによる平均応答時間の短縮を確認した．

1. はじめに

ネットワークの高速化やクラウドサービスの普及により，ネットワーク上に保存されるデータ量が爆発的に増加している．それにともない大規模なデータの管理に対する需要も増加している．

従来からデータの管理に利用されてきた関係データベース [1] は，データに対し関係代数に基づく複雑な演算処理が可能である．またトランザクションにより，強力な一貫性もサポートしている．しかし関係データベースはこれらの複雑な機能をサポートしているため，多数の計算機で処理を分散しても性能を向上させることが難しい．また大規模なデータに対する結合演算に大きなコストがかかるという問題もある．そのため関係データベースでは大規模なデータ管理の需要を満たすことができないケースが増えている．

そこで分散 Key Value Store(KVS) と呼ばれるデータベース管理システムが注目を集めている．分散 KVS はデータを Key と Value という単純な構造で管理するデータベースであり，関係データベースにおける関係代数演算などの複雑な演算処理はサポートせず，Key に対応する Value の取得といった単純な検索処理のみをサポートする．分散 KVS はこのような単純な構造であるため，多数の計算機で処理を分散することで容易に性能をスケールアウトさせることが可能であるという利点がある．また分散 KVS は一貫性の制限を緩和することで，処理性能を重視するように設計されている．このような設計のため，分散 KVS では，関係データベースで処理することができないような大規模なデータを扱うことが可能である．しかし分散 KVS は未だ発展途上であり，その実装にはいくつかの問題がある．

分散 KVS の問題の一つとして，複数の種類の検索クエリーが混在する場合の，応答時間の増加がある．分散 KVS では，ある Key を指定することで，その Key に対応した Value を取得する検索処理を行う．また分散 KVS の実装によっては Key の範囲を指定することにより，その Key 間に属する複数の Value を取得する検索処理などが可能な実装もある．ここでは単一の Key に対応する Value を取得する検索処理を単一検索，二つの Key 間に属する複数の Value を取得する検索処理を範囲検索と呼ぶ．この単一検索と範囲検索のような処理時間の異なる二つの検索処理が混在する状況においては，高速に実行可能な単一検索命令の実行が低速な範囲検索命令の実行に待たされてしまい，結果として高速な応答時間が実現可能な単一検索までも応

[1] 名古屋工業大学
Nagoya Institute of Technology

© 2013 Information Processing Society of Japan

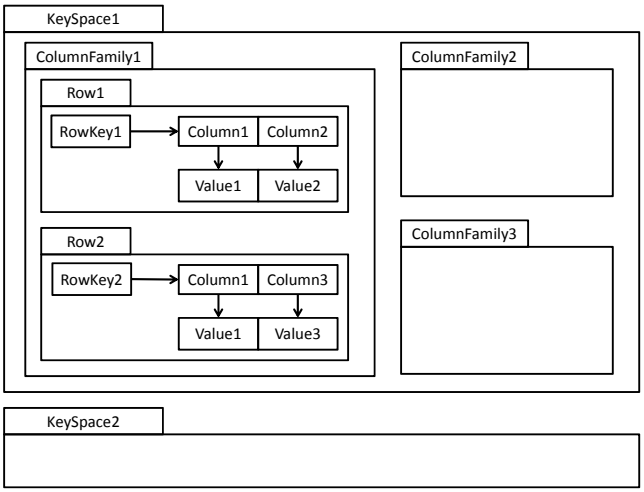

図 1 Cassandra のデータモデル

答時間が増加するという問題がある．

そこで本研究では分散 KVS の実装の一つである Apache Cassandra[2] に着目し，複数の種類の検索処理が混在する状況において，平均応答時間を短縮させるための手法を提案する．提案手法では検索クエリーのスケジューリングにより高速に処理可能なクエリーの優先度を上げ，低速なクエリーの優先度を下げることで，平均応答時間の短縮を実現する．提案手法を Cassandra 上に実装し，NoSQL 向けのベンチマークである YCSB を用いて性能評価を行なった．その結果，データを要求するクライアント数の増加に伴い，通常の Cassandra では平均応答時間の増加が見られたが，提案手法では平均応答時間の増加が抑えられていることを確認した．

以下，2 章で本研究で対象とする分散 KVS である Cassandra について説明する．次に 3 章で Cassandra の平均応答時間の短縮を実現する手法を提案し，4 章でその実装について述べる．その後，5 章で提案手法の評価を行う．最後に 6 章でまとめを行う．

2. Apache Cassandra

2.1 概要

本研究では分散 KVS の実装の一つである Apache Cassandra に着目する．Cassandra は Facebook 社が開発した分散 KVS であり，現在は Apache プロジェクトに寄贈されオープンソースで開発が続けられている．Cassandra は Amazon Dynamo[3] の分散デザインと Google Bigtable[4] のデータモデルをあわせ持つ分散データストアである．Cassandra の特徴として，高い可用性と耐障害性，分散型で単一障害点を持たないこと，ユーザによって設定可能な一貫性などが挙げられる．また，Cassandra は他の多くの分散 KVS とは異なり，範囲検索をサポートしている．

2.2 データモデル

Cassandra のデータモデルを図 1 に示す．一般的な KVS は 1 つの Key が 1 つの Value に対応するという単純なデータ構造を持つ．それに対し Cassandra は KeySpace,ColumnFamily,RowKey,Column という 4 つの Key により Value を管理する．KeySpace とは Cassandra のデータの中で最も外側にあるもので，関係データベースにおけるデータベースに近い役割を持つ．ColumnFamily は Row の集合の器であり，それぞれの Row 自体も Column の集合を持つ．Column は Cassandra のデータモデルで最も基本的な単位となるデータ構造であり，Column 名，Value，タイムスタンプの 3 つから構成される．このようなリッチなデータ構造により，一般的な分散 KVS よりも複雑なデータ管理が可能となる．

2.3 データ配置

分散 KVS では，複数の計算機でデータを分散して管理する．そのためデータと計算機の対応付けが必要になる．以下では Cassandra 上でデータを管理する計算機をノードと呼ぶ．通常の分散 KVS ではこの対応付けを ConsistentHashing 法 [5] に基づいて行う．ConsistentHashing 法ではデータの Key のハッシュ値を取り，そのハッシュ値をノードに対応付ける．この方法の利点として，分散 KVS を構成するクラスタにノードの追加・削除をしても，データとノードの対応付けを維持し続けることができるという点がある．しかし ConsistentHashing 法は，Key をハッシュ関数にかけるため，Key 同士の連続性が失われるという問題がある．そのため，ConsistentHashing 法に基づいてデータを配置した場合，範囲検索を実行するとクラスタ内のすべてのノードに問い合わせをする必要が出てくる．この検索方法は非効率であり，スケールアウトもしない．また検索の結果として取得したデータがランダムな順序で並んでいる可能性も高いため，検索終了後にソートが必要となる場合もある．これらの理由から ConsistentHashing 法に基づくデータの配置は，範囲検索に適さないと言える．

Cassandra では ConsistentHashing に基づくデータ配置法の他に，効率的な範囲検索処理を実現するために，Key をバイト列のまま扱うことで順序関係を維持するデータ配置もサポートしている．この配置方法では，二つの Key 間に属する複数の Value を管理するノードを限定するため，範囲検索において効率的にデータを問い合わせ可能である．また，取得するデータの順序関係は維持された状態にあるため，検索結果をソートする必要はない．そのため，クライアントが範囲検索を利用するならば後者の配置法を採用すべきである．ただしこの配置法は，一部のノードにデータが偏る恐れがあるという欠点がある．Key の頭文字からデータを配置するノードを決定する場合を考えると，一般的に Key の頭文字の分布は均等ではないため，データの分

© 2013 Information Processing Society of Japan

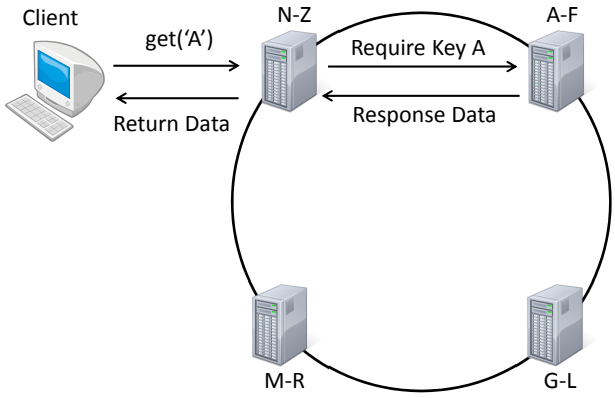

図 2 単一検索の処理の流れ

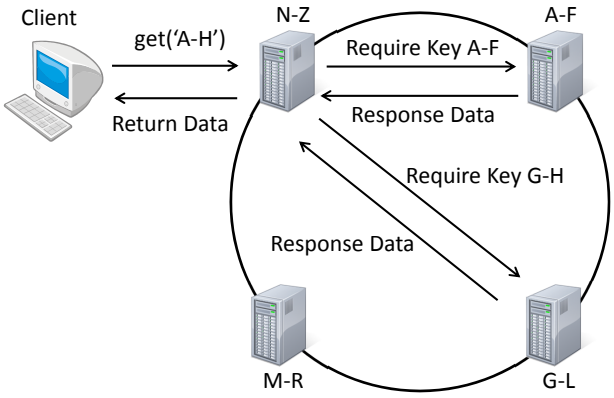

図 3 範囲検索の処理の流れ

散が不均等になる可能性が高い．そのため，範囲検索を利用しない場合，通常の ConsistentHashing 法に基づくデータの配置法を採用すべきである．

本研究では，単一検索と範囲検索などの，異なる種類の検索処理が混在する状況を想定するため，後者の配置法を採用する．

2.4 データ検索の流れ

Cassandra は複数のノードによってクラスタを構成することができる．クライアントは Cassandra のクラスタを構成している任意のノードに検索クエリーを送信することで，結果を受け取る．クライアントから検索クエリーを受信したノードは，Key を元にそのデータを管理しているノードのリストを作成し，そのリストのノードに対して，リクエストを送信する．その後，クライアントから検索クエリーを受信したノードはレスポンスを待ち，クライアントに結果を返す．以下では Cassandra がサポートしている単一検索と範囲検索という 2 種類の検索処理の詳細について説明する．

2.4.1 単一検索

Cassandra における単一検索の処理の流れを図 2 に示す．図の右側のリングは Cassandra クラスタを表し，リング上にはクラスタを構成するノードが配置されている．クラスタ上の各ノードの上下にある Key A-F,G-L,M-R,N-Z のアルファベッドは，そのノードが管理するデータの Key の範囲を示している．図の左側のノードは検索クエリーを送信するクライアントである．

単一検索では，まずクライアントが Cassandra のクラスタを構成する任意のノードに対しクエリーを送信する．例ではクライアントは Key が A であるデータを要求するクエリーを N-Z の Key を管理しているノードに送信している．クエリーを受信したノードは Key A-F を管理するノードに対して，Key A に対応するデータを問い合わせるリクエストを送信する．Key A-F を管理するノードでは Key A に対応するデータを検索し，結果を問い合わせ元のノードに送信する．クライアントからクエリーを受信したノードは，データを管理しているノードから受け取った結果をクライアントに送信する．

ただし，クライアントがクエリーを送信したノードが Key に対応するデータを持っていた場合，そのノードはそのまま自ノードのデータを検索し結果をクライアントに送信する．

2.4.2 範囲検索

Cassandra における範囲検索の処理の流れを図 3 に示す．クライアントは単一検索と同様の方法でクエリーを送信する．ただし単一検索とは異なり，クライアントは二つの Key を指定する．例では A から H の範囲の Key を持つデータを要求するクエリーを N-Z の Key を管理しているノードに送信している．クエリーを受信したノードは Key A-F を管理するノードに対して，Key A-F に対応するデータを問い合わせるリクエストを送信する．Key A-F を管理するノードでは Key A-F に対応するデータを検索し，結果を問い合わせ元のノードに送信する．クライアントからクエリーを受信したノードは次に，Key G-L を管理するノードに対して，Key G-H に対応するデータを問い合わせるリクエストを送信する．Key G-L を管理するノードでは Key G-L に対応するデータを検索し，結果を問い合わせ元のノードに送信する．クライアントからクエリーを受信したノードは，データを管理しているノードから受け取った結果をクライアントに送信する．

2.5 検索処理の平均応答時間の増加

一つの Key に対応する Value を取得する単一検索は，二つの Key 間に属する複数の Value を取得する範囲検索に比べて高速に実行可能である．一方 Cassandra では，各ノードでのデータの検索処理をリクエストの到着順で実行する．そのため，リクエストの到着順によっては高速に処理可能な単一検索が，検索処理を完了するために複数のノードからの検索結果を必要とする範囲検索処理により実行を待たされることで，単一検索の応答時間が増加するという問題がある．

また，もう一つの問題として Cassandra での範囲検索の

© 2013 Information Processing Society of Japan

逐次実行が挙げられる．Cassandra の範囲検索処理の実装方法では，検索範囲が複数のノードをまたぐ場合に，クエリーを受け取ったノードは各ノードへのリクエストを逐次的に送信する．この方法では，範囲検索の応答時間は各ノードでの処理時間の和となるため，検索範囲がまたぐノード数の増加に伴い応答時間が増加するという問題がある．

本研究では，上述した Cassandra における応答時間の増加を抑制するための手法を提案する．

3. 提案手法

本研究の提案手法を説明する．本研究では，Cassandra の検索処理の応答性能を改善するために，3 つの手法を提案する．

まず単一検索と範囲検索のリクエストのスケジューリングにより，単一検索の応答時間を短縮する．このスケジューリングでは高速に処理可能な単一検索の優先度を上げ，低速な範囲検索の優先度を下げることで，単一検索を範囲検索よりも優先して実行する．スケジューリングにより，高速な処理が低速な処理に待たされることによる応答時間を短縮する．

次に範囲検索の並列実行により，範囲検索の応答時間を短縮する．オリジナルの Cassandra の範囲検索の実装では，検索範囲が複数のノードにまたがる場合に，各ノードへのリクエストを逐次的に送信する．この各ノードへのリクエストの送信を，検索範囲がまたがるノードに一度に送信することで，各ノードでの検索処理を並列に実行し，範囲検索の応答時間を短縮する．

最後に範囲検索同士のスケジューリングにより，範囲検索の応答時間を改善する．このスケジューリングでは範囲検索の検索範囲に応じて優先度を設定する．検索範囲が狭い範囲検索の優先度を上げ，検索範囲の広い範囲検索の優先度を下げることで，より少ないノードからの検索結果で要求を満たすことのできるクエリーを優先して実行する．検索範囲の広さは，範囲検索の並列実行で一度に送信したノード数，つまり並列に検索処理を実行しているノード数を基準とする．また，範囲検索同士のスケジューリングでは，他ノードでのリクエストの進捗により優先度を変更する．同じ検索範囲の広さを持つ範囲検索クエリー同士であっても，一方があと 1 ノードで検索範囲内の検索がすべて完了し，もう一方が検索範囲内の検索が全く完了していない状況では，前者のリクエストを優先した方が待ち時間は短縮可能である．そこで各ノードでのリクエストの処理状況を通知し合うことにより，優先度の動的な変更を行う．これらのスケジューリングにより，範囲検索の応答時間を短縮する．

以上の提案手法により，Cassandra の検索処理の応答性能を向上させる．

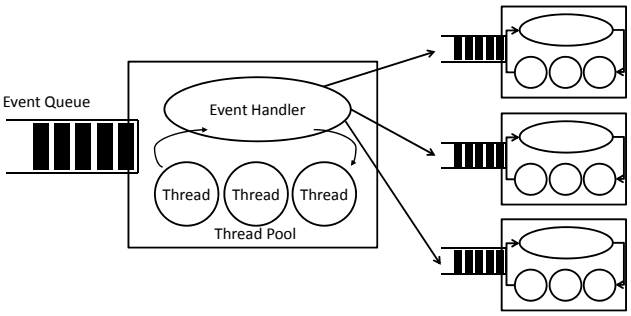

図 4 SEDA の各 Stage

4. 実装

4.1 単一検索と範囲検索のスケジューリング

Cassandra は Stage Event Driven Architecture (SEDA)[6] を実装している．SEDA はアプリケーションを，複数の Stage と呼ばれる単位に分割して管理する．各 Stage は図 4 に示すように Event Queue，Thread Pool，Event Handler で構成され，各 Stage で Event を受け渡しながら処理を進める．例えば検索処理の場合，検索リクエスト命令を Event Queue に投入する．Thread Pool は Event Queue 上のリクエスト命令を取り出し，Event Handler によって実行する．このように Event の実行とリクエストの受信処理を分離することで，リクエスト数が増えた場合でも，スレッド数が増加してスループットが落ちることはない．

Cassandra は 11 の Stage に分離されている．Stage には検索処理を管理する READ Stage，データの挿入処理を管理する MUTATION Stage などがある．

本提案手法では, 検索処理を管理する Stage である READ Stage の Event Queue に優先度付きキューを利用することにより，検索処理のスケジューリングを実装した．優先度は，自ノードからの単一検索リクエスト命令，他ノードからの単一検索リクエスト命令，範囲検索リクエスト命令の順に高く設定した．

4.2 範囲検索の並列実行

オリジナルの Cassandra の範囲検索では，クライアントは検索範囲の開始となる Key，終了となる Key，そしてその範囲で最大いくつのデータ (Value) を取得したいかを示す取得数を指定して範囲検索クエリーを送信する．この取得数の制限により，範囲検索処理は検索範囲の途中であっても処理を終了する場合がある．そのためオリジナルの Cassandra の実装ではクライアントの指定した取得数を満たすまで，検索範囲のノードを逐次的に検索する．この実装は検索範囲がまたがるノード数が少ない範囲検索クエリーでは特に問題にならないが，検索範囲のまたがるノード数が多い範囲検索クエリーでは応答時間の増加につながる．

© 2013 Information Processing Society of Japan

そこで本提案手法では，検索範囲のノードへのリクエストを一度に送信することで，各ノードでの処理を並列に実行する．Cassandra は，各ノードが保持するデータ数などの情報を統計情報として収集している．その情報をクラスタ内で共有し，検索リクエストを送信するノード数の予測に利用する．本提案手法では，検索範囲内で，クライアントが指定したデータの最大取得数を満たすだけのノード数に検索リクエストを送信する．ただし，検索範囲の開始となる Key を管理するノードが保持するデータ数は考慮しない．ノードの管理しているデータ数は，そのノードが担当する Key の範囲に含まれるデータ数である．範囲検索の検索範囲の開始となる Key は，その Key を管理しているノードの担当する Key の範囲の開始と一致するとは限らず，そのノードが保持するデータをすべて取得できるとは限らない．そのため，本実装ではこのノードが保持するデータ数は 0 として扱う．

4.3 範囲検索同士のスケジューリング

範囲検索リクエスト命令の優先度付けでは，検索範囲の広さによって優先度を設定する．検索範囲の広さは範囲検索の並列実行の際に予測した，検索リクエストを送信するノード数を基準にする．リクエストが送信されるノード数が少ない範囲検索リクエスト命令の優先度を高く設定し，ノード数が多い範囲検索リクエスト命令の優先度を低く設定する．ただし，リクエストが送信されるノード数が同じ場合には，クライアントの指定したデータの最大取得数の少ないリクエスト命令を優先する．これらの優先度付けに基づいて，READ Stage の Event Queue でスケジューリングする．

また，検索範囲のノードからリクエストの結果が送信されるたびに，まだリクエストの処理が完了していないノードに対して，クエリーの処理がすべて完了するまでに必要なノード数の情報を送信する．情報を受信したノードでは，対応するリクエスト命令を Event Queue から検索し，その優先度を，そのクエリーよりも処理がすべて完了するまでに必要なノード数が多い他のクエリーのリクエスト命令よりも高くする．これにより，検索範囲中で検索が完了していないノード数が少ない範囲検索リクエスト命令を優先して実行する．

5. 評価

複数クライアントによる単一検索と範囲検索のクエリーの送信実験により，提案手法を実装した Cassandra とオリジナルの Cassandra との性能比較を行なった．本評価実験ではクライアントプログラムとして Yahoo! Research が開発したオープンソースのベンチマークツールである YCSB(Yahoo! Cloud Serving Benchmark)[7] を使用した．

5.1 評価環境

本評価実験では Cassandra バージョン 1.1.2 を用い，12 台のノードからなる Cassandra クラスタを構築した．またクエリーを送信するクライアントノードを 1 台用意し，YCSB-0.1.4 をインストールした．Cassandra クラスタとクライアントノードはすべて同一のハブで接続した．Cassandra クラスタの各ノードの性能を表 1 に，クライアントノードの性能を表 2 にそれぞれ示す．

表 1 Cassandra クラスタの各ノードの性能

OS	Linux 2.6.38-8-amd64 Ubuntu 11.04 Server
CPU	Intel(R) Core(TM) i5 CPU 750 @ 2.67GHz
Memory	8GB
Network	1000BASE-T

表 2 YCSB を動作させるノードの性能

OS	Linux 3.5.0-23-amd64 Ubuntu 12.04 Server
CPU	Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz
Memory	32GB
Network	1000BASE-T

5.2 実験内容

検索クエリーを送信する前に，あらかじめ 1 件あたり約 1KB のデータ 10,000,000 件を Cassandra ノードに挿入した．各データはキーに応じて 1,000 個のカラムファミリに分けて格納している．Cassandra の範囲検索では，カラムファミリごとにデータを検索する．そのためデータをカラムファミリに分けることで，一回の範囲検索リクエストで取得するデータ数が分割され，メモリの不足や，データ転送時間の増大を防ぐことができる．これにより，1 回の範囲検索クエリーで取得するデータ量は最大で約 10MB となる．カラムファミリを分割せず，ただ一つのカラムファミリに 10,000 件のデータを挿入し，検索する場合，メモリキャッシュ上にすべてのデータが展開される可能性がある．Cassandra は一般的にメモリキャッシュ上に乗り切らないような大規模なデータの管理を想定しているため，本評価実験ではカラムファミリを分けてデータの挿入を行った．またメモリキャッシュ上にデータが残らないように，ベンチマークプログラムを起動する前に Cassandra クラスタを構成する各ノードでページキャッシュをクリアしている．

クエリー送信実験では，単一検索と範囲検索を 9:1 と 5:5 の割合で，全クライアントで合計 2000 クエリーを送信した．範囲検索クエリーでは，検索範囲の開始となる Key として，挿入されている Key の中からランダムに一つ選択する．またデータの最大取得数は 1 から 10,000 の範囲から一様分布で決定する．検索クエリーを送信するクライアン

© 2013 Information Processing Society of Japan

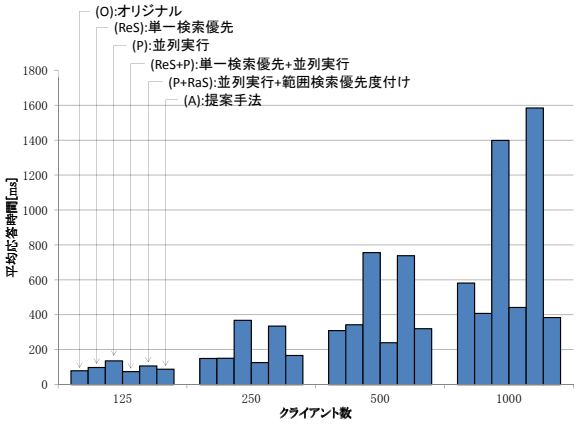

図 5 単一検索と範囲検索の比率 9:1 での単一検索の平均応答時間

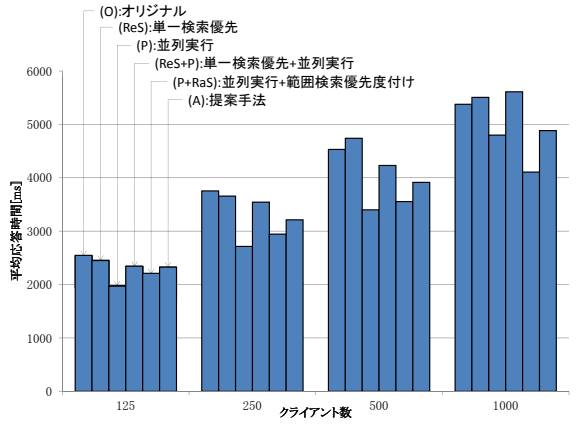

図 6 単一検索と範囲検索の比率 9:1 での範囲検索の平均応答時間

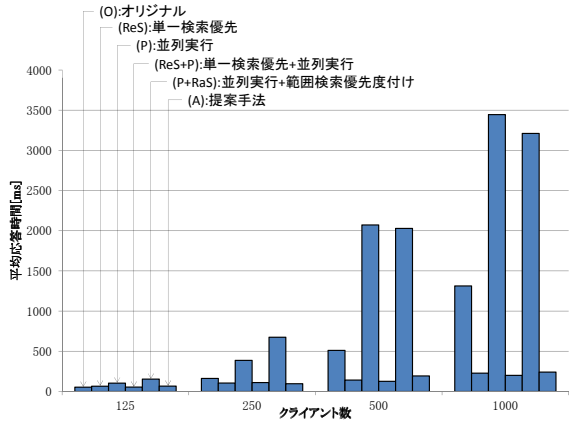

図 7 単一検索と範囲検索の比率 5:5 での単一検索の平均応答時間

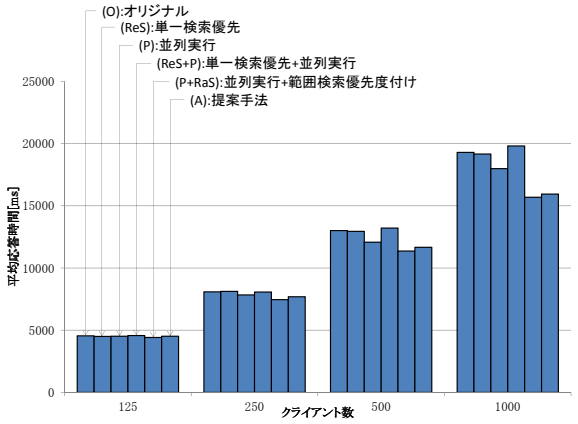

図 8 単一検索と範囲検索の比率 5:5 での範囲検索の平均応答時間

ト数を 125，250，500，1000 と増加させた場合の，単一検索と範囲検索の平均応答時間を測定した．

評価対象は，以下の 6 つである．

(O) オリジナルの Cassandra

(ReS) 単一検索の優先実行のみを実装した Cassandra

(P) 範囲検索の並列実行のみを実装した Cassandra

(ReS+P) 単一検索の優先実行と範囲検索の並列実行を実装した Cassandra

(P+RaS) 範囲検索の並列実行と範囲検索の優先度付けを実装した Cassandra

(A) 単一検索の優先実行と範囲検索の並列実行，範囲検索の優先度付けを実装した Cassandra

評価をそれぞれ各 10 回ずつ行い，最もスループットの良いものを採用した．なお本評価実験ではレプリケーションは行っていない．

5.3 評価結果

単一検索と範囲検索を 9:1 の割合で送信した場合の単一検索の平均応答時間を図 5，範囲検索の平均応答時間を図 6 に，単一検索と範囲検索を 5:5 の割合で送信した場合の単一検索の平均応答時間を図 7，範囲検索の平均応答時間を図 8 に示す．

図 5 より，クライアント数が 125,250,500 の場合には，単一検索の優先実行の効果は得られていないことが確認できる．しかし，クライアント数が 1000 の場合では，(ReS)，(ReS+P)，(A) は (O) に比べて約 25%の単一検索の平均応答時間の削減が確認できる．これは (O) では，クライアント数の増加により送信される範囲検索クエリーの絶対数が増えたためであると考えられる．範囲検索クエリーの数が増えると単一検索の処理が範囲検索の処理に待たされる可能性が高くなり，単一検索の平均応答時間が増加する．(ReS)，(ReS+P)，(A) では，範囲検索よりも単一検索を優先して実行するため，範囲検索クエリーの増加の影響が抑えられる．(P)，(P+RaS) では，クライアント数の増加に伴ない単一検索の平均応答時間が大幅に悪化している．クライアント数が 1000 の場合では，(O) に比べ，平均応答時間が約 2.4 倍になっている．これは範囲検索の並列実行により，各ノードでの範囲検索処理が並列化されるため，Cassandra ノード全体の範囲検索リクエスト命令の絶対数が増加することが原因だと考えられる．

図 6 より，(P) と (P+RaS) の範囲検索の平均応答時間が (O) よりも削減されていることが確認できる．特にクライアント数が 1000 の場合では，(O) に比べ約 10%の平均応答時間の削減を実現している．これは各ノードでの範囲

© 2013 Information Processing Society of Japan

検索処理を並列に実行することで，逐次実行の場合よりも検索結果を集める時間が減るためであると考えられる．また (P+RaS) では，クライアント数が 1000 の場合に並列実行のみの実装に比べて平均応答時間が削減されている．これは範囲検索クエリーの増加により，優先度付けが有効に働いたためであると考えられる．(ReS+P) では，範囲検索の平均応答時間は削減されていない．これは範囲検索のリクエストを各ノードに並列に送信しても，単一検索の方が優先的に実行されるため，効果が得られなかったと考えられる．ただし，範囲検索の優先度付けを併用している (A) では，応答時間が削減されている．

図 7 より，(O) を始め単一検索を優先実行する実装を施されていない Cassandra の場合，クライアント数の増加に伴ない，単一検索と範囲検索が 9:1 の場合よりも単一検索の平均応答時間が大幅に悪化していることが確認できる．これは，範囲検索の割合が増加することで，多くの単一検索が範囲検索処理に待たされたためであると考えられる．これに対し，(ReS)，(ReS+P)，(A) では，平均応答時間が短縮されていることが確認できる．特にクライアント数が 1000 の場合では約 80%の平均応答時間の削減を実現している．(P) では，単一検索の平均応答時間が大幅に悪化しており，クライアント数が 1000 の場合では (O) に比べ平均応答時間が 2.4 倍になっている．これも単一検索と範囲検索が 9:1 の場合と同様，範囲検索の並列実行により，各ノードでの範囲検索処理が並列化され，Cassandra ノード全体の範囲検索リクエスト命令の絶対数が増加するためであると考えられる．

図 8 より，(P)，(P+RaS)，(A) では範囲検索の平均応答時間が削減されていることが確認できる．特に (A) はクライアント数が 1000 の場合に，(O) に比べ約 17%の平均応答時間の削減を実現している．この実験では，範囲検索の割合が大きいため範囲検索の並列実行と，優先度付けの両方が有効に働いたと考えられる．

以上の評価結果から以下のことが確認できる．

- 単一検索リクエストの優先実行は単一検索リクエストの応答時間削減のためには有効であるが，範囲検索の応答時間には影響を与えない
- 範囲検索リクエストの並列実行は単一検索リクエストの応答時間に悪影響を与えるが，範囲検索リクエストの応答時間削減には有効である
- 範囲検索リクエストの優先度付けは単一検索リクエストの応答時間には影響を与えないが，範囲検索リクエストの応答時間削減には有効である

また提案手法では，3 つの手法の効果がいずれも応答時間の短縮に効果的に作用していることが確認できる．

6. おわりに

分散キーバリューストアでの検索の応答性能を改善するために．検索処理のスケジューリングと範囲検索の並列実行を提案をした．本研究では単一検索と範囲検索という 2 種類の検索クエリーが混在する状況を想定し，それらのクエリーの優先度を付けることによりスケジューリングを実現した．

提案手法を分散キーバリューストアの実装の一つである Cassandra 上に実装し，ベンチマークプログラム YCSB により評価した．評価の結果，オリジナルの Cassandra では，クライアント数の増加に伴い単一検索の応答時間が大幅に悪化したのに対し，提案手法を実装した Cassandra では，これを改善することを確認した．また範囲検索についても，並列実行とスケジューリングの効果により応答時間が改善されることを確認した．

謝辞 本研究の一部は，科研費基盤研究 (C)24500113 による．

参考文献

[1] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, Vol. 13, No. 6, pp. 377–387, June 1970.

[2] Avinash Lakshman and Prashant Malik. Cassandra - a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, Vol. 44, No. 2, April 2010.

[3] Madan Jampani Gunavardhan Kakulapati Avinash Lakshman Alex Pilchin Swaminathan Sivasubramanian Peter Vosshall Giuseppe DeCandia, Deniz Hastorun and Werner Vogels. Dynamo: Amazon 's highly available key-value store. *ACM SIGOPS Operating Systems Review*, Vol. 41, No. 6, pp. 205–220, October 2007.

[4] Sanjay Ghemawat Wilson C. Hsieh Deborah A. Wallach Mike Burrows Tushar Chandra Andrew Fikes Robert E. Gruber Fay Chang, Jeffrey Dean. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems*, Vol. 26, No. 2, November 2006.

[5] David Karger, Eric Lehman, Tom Leighton, Mathhew Levine, Daniel Lewin, and Rina Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *In ACM Symposium on Theory of Computing*, pp. 654–663, 1997.

[6] Matt Welsh, David Culler, and Eric Brewer. Seda: An architecture for well-conditioned, scalable internet services, 2001.

[7] Erwin Tam Raghu Ramakrishnan Russell Sears Brian F. Cooper, Adam Silberstein. Benchmarking cloud serving systems with ycsb, 2010. http://github.com/brianfrankcooper/YCSB.

© 2013 Information Processing Society of Japan