情報処理学会論文誌 Vol.56 No.2 492–502 (Feb. 2015)

検索範囲を考慮したクエリスケジューリングによるCassandraの応答性能向上

福田 諭[1,a)] 川島 龍太[1] 齋藤 彰一[1] 松尾 啓志[1]

受付日 2014年5月14日, 採録日 2014年11月10日

概要：クラウドサービスの普及により，日々膨大なデータが生成されており，大規模なデータの管理に対する需要が増加している．それにともない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である分散 Key Value Store（KVS）が広く利用されている．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ものも存在する．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をスケジューリングし，高速に実行可能なクエリを優先的に実行することにより，検索クエリ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，評価を行った結果，クエリのスケジューリングにより，単一検索の平均応答時間を最大で 80%，範囲検索の平均応答時間を最大で 20%短縮できることを確認した．

キーワード：分散 KVS，Cassandra，クエリスケジューリング

Improving Cassandra Query Scheduling in Consideration of Query Range

Satoshi Fukuda[1,a)] Ryota Kawashima[1] Shoichi Saito[1] Hiroshi Matsuo[1]

Received: May 14, 2014, Accepted: November 10, 2014

Abstract: A management of large-scale data becomes more important, along with the spread of cloud services. Distributed Key Value Store (KVS) is a datastore which manages data across multiple machines. Since distributed KVSs manage data which consists of simple key-value pair, they can achieve scalability easily. Distributed KVSs are widely used in many services managing large-scale data, such as Facebook and Twitter. Distributed KVSs provide interfaces to access key-value pair by simply specifying the key. In this paper, we refer to a query which only obtains a value from a key as a single query. Some distributed KVSs support a range query which obtains multiple values from a key range. However, under mixed query workloads that consist of single and range queries, single queries (which can be executed faster) are forced to wait until preceding range queries are finished. And this results in the increase of average response time. We propose an approach to reduce the average response time by query scheduling. We implemented our method on Cassandra, and evaluation results showed a reduction of the average response time of single queries by 80% and the average response time of range queries by 20%.

Keywords: Distributed Key Value Store, Cassandra, query scheduling

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

[1] 名古屋工業大学
Nagoya Institute of Technology, Nagoya, Aichi 466–8555, Japan
[a)] fukuda@matlab.nitech.ac.jp

© 2015 Information Processing Society of Japan

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する必要があるため，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困難である．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 Dynamo [2] や Bigtable [3]，memcached [4]，Cassandra [5], [6] などの分散 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 の取得という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要求のみを処理するため，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．しかし分散 KVS はいまだ発展途上であり，その実装にはいくつかの問題がある．

本研究では分散 KVS の実装の 1 つである Apache Cassandra に着目し，複数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．既存の Cassandra の実装では複数の種類の検索処理が混在する場合に，平均応答時間が増加するという問題がある．提案手法では検索クエリのスケジューリングにより高速に処理可能なクエリの優先度を上げ，低速なクエリ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．計算機シミュレーションにより提案手法の有効性を確認したのち Cassandra 上に実装し，NoSQL 向けのベンチマークを用いて性能評価を行った．その結果，データを要求するクライアント数の増加にともない，通常の Cassandra では平均応答時間の増加が見られたが，提案手法では単一検索の平均応答時間を最大で 80%，範囲検索の平均応答時間を最大で 20%短縮できることを確認した．

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．6 章で関連研究について述べ，最後に 7 章でまとめを行う．

2. Apache Cassandra

2.1 概要

Apache 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 の分散デザインと Google Bigtable 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あげられる．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．

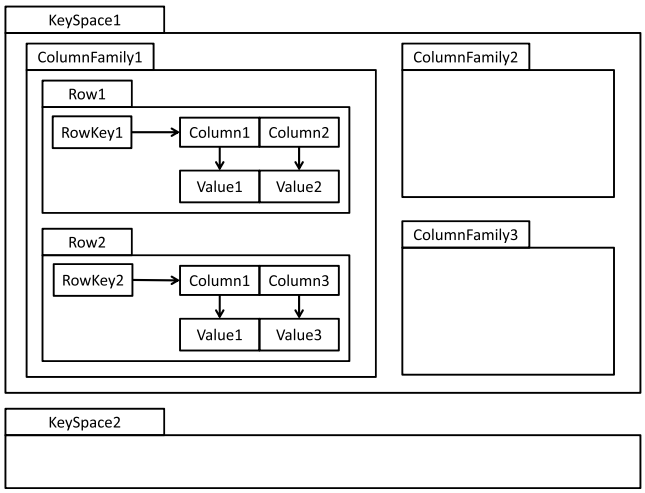

図 1 Cassandra のデータモデル
Fig. 1 Data model of Cassandra.

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 という入れ子構造で 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 Column のコンテナであり，関係データベースにおけるテーブルに近い役割を持つ．ColumnFamily 内の複数の Column は行としてまとめられ RowKey によって識別される．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．このような高機能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

Cassandra では，複数の計算機でデータを分散して管理する．そのためデータの Key（RowKey）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 [7] に基づいて行う．ConsistentHashing 法ではデータの Key のハッシュ

© 2015 Information Processing Society of Japan

値をとり，そのハッシュ値をノードに対応付ける．しかし，ConsistentHashing 法に基づいたデータ配置では，Key どうしの連続性が失われるため範囲検索処理が非効率になり，またスケールアウトもしない．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．

Cassandra では ConsistentHashing 法に基づくデータ配置法のほかに，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，2 つの Key 間に属する複数の Value を管理するノードを限定可能なため，範囲検索において効率的にデータを問合せ可能である．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．これらより，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を受信したノードは，Key をもと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．ここで，クライアントからのデータ要求メッセージを**クエリ**，クエリを受け取ったクラスタ内のノードが，データを管理するノードに対してデータを要求するメッセージを**リクエスト**，リクエストを受け取ったクラスタ内のノードが自ノードの処理キューに投入する命令を**リクエスト命令**とそれぞれ呼ぶ．

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 2 の左側のリングは 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側にある Key A-F，G-L，M-R，S-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 2 の右側のノードは検索クエリを送信するクライアントである．

単一検索では，まず (1) 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を送信する．例ではクライアントは Key が S であるデータを要求するクエリを A-F の Key を管理しているノードに送信している．(2) クエリを受信したノードは Key S-Z を管理するノードに対して，Key S 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．(3) Key S-Z を管理するノードでは Key S に対応するデータを検索し，結果を問合せ元のノードに送信する．クライアントからクエリを受信したノードは，(4) 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．(1) クライアントは単一検索と同様の方法でクエリを送信する．ただし単一検索とは異なり，クライアントは 2

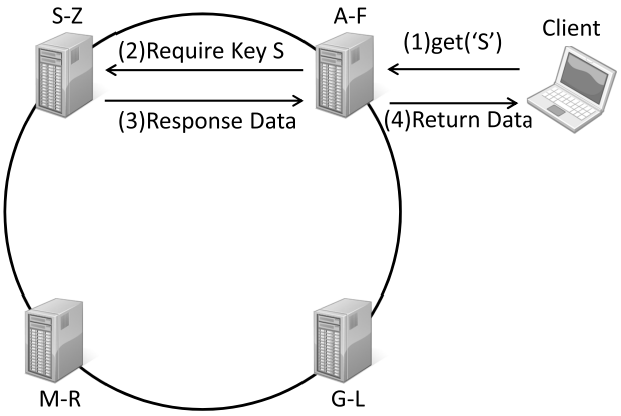

図 2 単一検索の処理の流れ
Fig. 2 A flow of the single query.

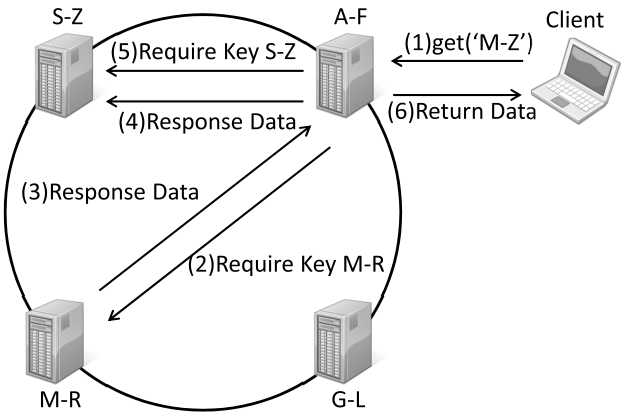

図 3 範囲検索の処理の流れ
Fig. 3 A flow of the range query.

© 2015 Information Processing Society of Japan

つの Key を指定する．例では M から Z の範囲の Key を持つデータを要求するクエリを A-F の Key を管理しているノードに送信している．(2) クエリを受信したノードは Key M-R を管理するノードに対して，Key M-R 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．(3) Key M-R を管理するノードでは Key M-R に対応するデータを検索し，結果を問合せ元のノードに送信する．クライアントからクエリを受信したノードは次に，(4) Key S-Z を管理するノードに対して，Key S-Z 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．(5) Key S-Z を管理するノードでは Key S-Z に対応するデータを検索し，結果を問合せ元のノードに送信する．(6) クライアントからクエリ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 検索処理の問題点

1 つの Key に対応する Value を取得する単一検索は，2 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．一方範囲検索では結果として取得されるデータサイズの増加にともない，実行時間が増加する [8]．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 1 つの問題として Cassandra での範囲検索の逐次実行があげられる．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，クエリを受け取ったノードが，各ノードへのリクエストを逐次的に送信することによる性能低下も存在する．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ともな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，以下の 3 つの手法を提案する．

- 単一検索の優先実行
- 範囲検索の並列要求
- 範囲検索どうしのスケジューリング

以降で各手法について詳細を説明する．

3.1 単一検索の優先実行

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．具体的には単一検索リクエスト命令が各ノードのリクエストキューに投入される場合に，他のすべての範囲検索リクエスト命令よりも前方に単一検索リクエスト命令を投入する．本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の増加を抑制する．ただし本スケジューリングでは，単一検索を無条件で優先実行するため，範囲検索がスタベーション状態になる可能性がある．しかしスタベーションはエージングなどによって容易に解決可能であるため，本稿では考慮しない．

3.2 範囲検索の並列要求

範囲検索の並列要求により，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 1 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．

3.3 範囲検索どうしのスケジューリング

範囲検索どうし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．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の優先度を上げる．検索範囲の広さは，範囲検索の並列要求で 1 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．

3.3.1 検索範囲のみでの優先度付けの問題点

単純に検索範囲のみで優先度を設定した場合，範囲検索の平均応答時間が増加する場合がある．たとえば Client1 が A-F，G-L，M-R を管理する 3 ノードにまたがるデータを要求する範囲検索クエリを送信し，次に Client2 が M-R，S-Z を管理する 2 ノードにまたがるデータを要求する範囲検索クエリを送信した場合を考える．このとき，S-Z を管理するノードが高負荷である場合，各ノードのリクエストキューは図 4 のように，A-F，G-L のキューに 1 つずつ，M-R のキューに 2 つ，S-Z に多数のリクエストが挿入されている状態になる．この場合，検索範囲による優先度を考慮して Client2 が送信した M-R，S-Z を管理する 2 ノードにまたがるデータを要求する範囲検索クエリを Client1 が送信したクエリよりも優先すると，他のノードからの複数の要求がリクエストキューに含まれる S-Z を担当するノードのために Client2 の応答時間は短縮されず，Client1 の応答時間は増加する．結果として Client1 と Client2 の範囲

© 2015 Information Processing Society of Japan

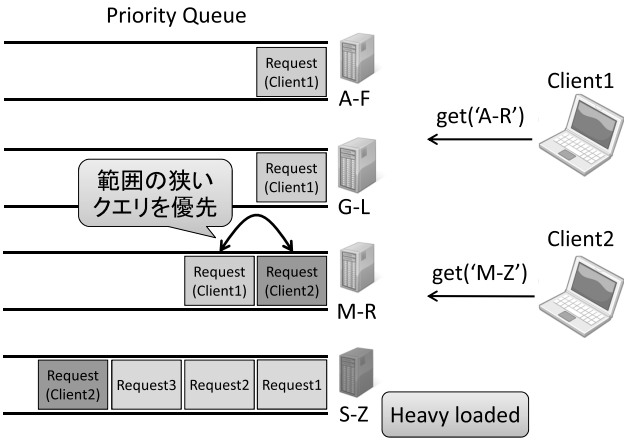

図 4　検索範囲のみでの優先度付けで応答性能が低下する場合
Fig. 4　Determinating priority of range queries depending on the range size cause response performance degradation.

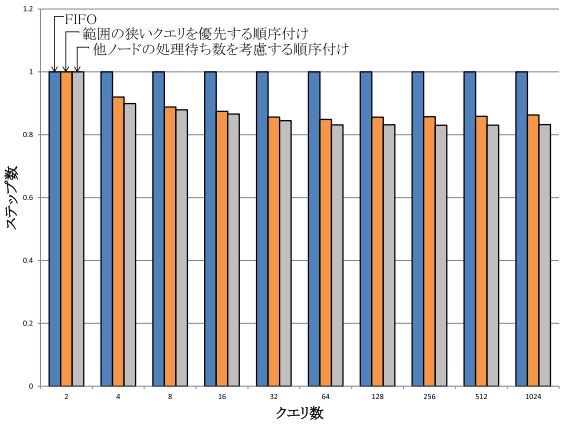

図 5　シミュレーション結果
Fig. 5　Simulation results.

検索クエリの平均応答時間は増加する．

検索範囲のみでの優先度設定による平均応答時間の増加を抑制するためには，各ノードが他のノードの負荷を考慮してスケジューリングする必要がある．すなわち，多くのリクエストがキューに溜まっているような負荷の高いノードを範囲に含むクエリの優先度を下げることで，平均応答時間の増加を抑制することができると考えられる．そのようなスケジューリングの効果を検証するために，計算機シミュレーションを行い平均応答時間を調べた．

3.3.2　シミュレーション概要

範囲検索どうしのスケジューリングにおいて，検索範囲のみを考慮した場合と，検索範囲に加えノードの負荷状態を考慮した場合との性能を比較するために計算機シミュレーションを行った．シミュレーションの概要を以下に示す．

(1) ノードに対応する優先度付きキューを，ノード数分生成する．
(2) 優先度付きキューの中からランダムに 2 つを選び，検索範囲の開始位置と終了位置を決定する．
(3) 検索範囲のノードに対応するキューに，リクエストに対応するオブジェクトを投入する．
(4) (2) から (3) の動作をクエリ数分繰り返す．
(5) ステップ数を 0 に初期化する．
(6) ステップ数をインクリメントして，すべてのキューから先頭のオブジェクトを取り出す．
(7) あるクエリのリクエストに対応するすべてのオブジェクトが取り出された場合，そのときのステップ数をそのクエリの応答時間とする．
(8) すべてのキューが空になるまで (6) から (7) の動作を繰り返す．
(9) すべてのクエリの応答時間の平均をとり，平均応答時間を算出する．

以上のシミュレータの動作により，検索範囲の開始位置と終了位置が一様分布であるような範囲検索の平均応答時間を算出することができる．またノードに対応する優先度付きキューの順序付けを変更することでクエリのスケジューリングを実現することもできる．本シミュレーションでは優先度付きキューの順序付けは FIFO，検索範囲の狭いクエリを優先する順序付け，他のノードの処理待ち数を考慮する順序付けの 3 つを設定した．他のノードの処理待ち数を考慮する順序付けでは，次の 2 段階でスケジューリングを行う．第 1 段階では，検索範囲の狭いクエリを優先する場合と同じ順序付けでリクエストを投入する．第 2 段階では，検索範囲内に投入したリクエスト命令の平均処理待ち数を算出する．クエリの検索範囲内のリクエスト命令の平均処理待ち数が少ないリクエスト命令の優先度を下げ，多いリクエスト命令の優先度を上げる．このような順序付けにより他のノードの負荷を考慮したスケジューリングを実現する．

3.3.3　シミュレーション結果と考察

シミュレーション結果を図 5 に示す．本シミュレーションではノード数を 12 に設定し，クエリ数を 2 から 1,024 の間で変化させた．グラフの縦軸は FIFO による順序付けでのクエリの平均ステップ数を 1 として正規化した値，横軸はクエリ数である．シミュレーション結果より，検索範囲の狭いクエリを優先する順序付けは FIFO に比べステップ数が約 14%削減されており，他のノードの処理待ち数を考慮する順序付けは FIFO に比べステップ数が約 17%削減されていることが確認できる．この結果より，検索範囲の狭いクエリを優先する順序付けは FIFO に比べて大きな優位性があると考えられる．一方で，他のノードの処理待ち数を考慮する順序付けは，検索範囲の狭いクエリを優先する順序付けと比べてステップ数の削減率はそれほど大きなものではない．他のノードの処理待ち数を考慮する順序付けでは処理待ち数の多いノードを検索範囲に含むようなリクエスト命令の優先度を下げ，処理待ち数の少ないノードにまたがるリクエスト命令の優先度を上げる．そのため，処

© 2015 Information Processing Society of Japan

理待ち数の少ないノード群を検索範囲とするようなクエリに対応するリクエスト命令が優先的に実行され，それらのクエリの応答時間が短縮される．この順序付けによって応答時間が短縮されるクエリは処理待ち数の少ないノード群を検索範囲とするクエリである．つまりこのスケジューリングが効果を発揮するのは処理待ち数の少ないノードを検索範囲とするクエリが数多く存在する状況である．しかし処理待ち数の少ないノードを検索範囲とするクエリが数多く存在する状況は起きえないため，他のノードの処理待ち数を考慮する順序付けはあまり大きな効果を得られなかったと考えられる．また，他のノードの処理待ち数を考慮する順序付けでは，リクエスト命令を投入するたびに他のノードの処理待ち数の情報を取得する必要がある．そのため，この順序付けを実現するためにはノードどうしでの通信処理が必要になる．1 つのクエリをスケジューリングする際には，そのクエリに対応するリクエスト命令の挿入位置に関する情報の通信処理と処理待ち数の平均値を通知するための通信処理という 2 回の通信処理を，クエリの検索範囲がまたがるノード数分行う必要がある．また検索範囲の開始位置と終了位置を，1 から全ノード数の間で一様分布で決定した場合，検索範囲がまたがるノード数は全ノード数に比例する．そのため他のノードの処理待ち数を考慮する順序付けの実現にはノード数とクエリ数の積に比例する通信処理が必要となる．このように他のノードの処理待ち数を考慮する順序付けは，検索範囲の狭いクエリを優先する順序付けと比べて大きな削減効果が期待できないにもかかわらず多くの通信処理を必要とするため，検索範囲の狭いクエリを優先する順序付けの方が大規模なシステムにおいては優位性が高いと考えられる．そこで本稿では検索範囲の狭いクエリを優先するスケジューリングのみを実装し，その効果を検証した．

3.4 一貫性制御

本提案手法ではレプリカの一貫性については保証していない．一般に分散 Key Value Store は関係データベースが保証しているような強い一貫性が要求されるアプリケーションでは利用されておらず，結果整合性と呼ばれる一貫性モデルを許容するアプリケーションで利用されている．結果整合性は，「長い時間が経った後にレプリカ全体が一貫した状態になる」というきわめて楽観的なポリシである．本提案手法は結果整合性を許容するアプリケーションでの利用を想定しているため，一貫性については結果整合性モデルを採用する．

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）[9] 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 **6** 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．たと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．

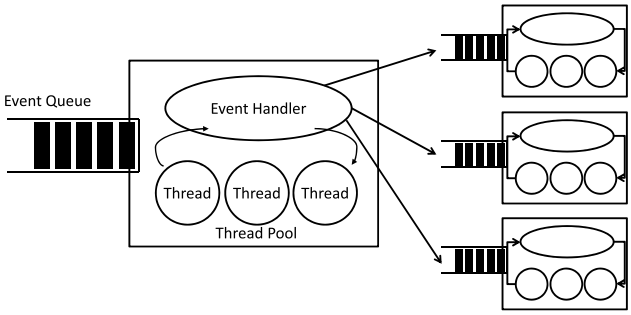

図 **6** SEDA の各 Stage
Fig. 6 Each stages of SEDA.

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 範囲検索の並列要求

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最大取得数を指定して範囲検索クエリを送信する．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では応答時間の増加につながる．

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 1 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 ではシステムの管理者向けに，各ノードの保持するデータ数の概算，ColumnFamily への読み出し・書き込み回数，ディスク使用率などの情報を統計情報

© 2015 Information Processing Society of Japan

として収集している．本提案手法では，それらの情報のうち，各ノードの保持するデータ数についての情報を範囲検索リクエストの返信メッセージにピギーバックし，ノード間で共有する．範囲検索リクエストの送信時には以下の式を満たす最小のノード数分だけリクエストを送信する．

$$\sum_{i=1}^{N} estimatedKeyCount[i] \geq countValue$$

ここで $i = 0, \ldots, N$ は検索範囲のノードを示し，$estimatedKeyCount[i]$ はノード i が保持する Key 数の概算，$countValue$ はクライアントの指定したデータの最大取得数を示す．この式の左辺はノード $i = 1, \ldots, N$ の保持している合計データ数の予測値を表す．このとき $i = 0$，つまり検索範囲の開始ノードが保持する Key 数は含まないものとしている．これは検索範囲の開始 Key は，$i = 0$ であるノードの担当する Key 範囲の開始と一致するとは限らず，$i = 0$ 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ためである．もちろんこの実装では，検索範囲内のノードに余分な検索リクエストが送信される可能性がある．しかし，この余分なリクエストによって返却される結果は，クライアントへの結果の返却時には無視されるため，大きな問題とはならないと考えられる．

4.3 範囲検索どうし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要求の際に予測した，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．

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）[10] 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 を実行するノードとした．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
Table 1 The specification of Cassandra cluster nodes.

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

表 2 YCSB を動作させるノードの性能
Table 2 The specification of a client node.

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

5.2 実験内容

検索クエリを送信する前に，あらかじめ 1 件あたり約 1 KB のデータ 10,000,000 件を Cassandra ノードに挿入した．各データは Key に応じて 1,000 個の ColumnFamily に分けて格納している．Cassandra の範囲検索では，ColumnFamily ごとにデータを検索する．そのためデータを ColumnFamily に分けることで，1 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で取得するデータ量は最大で約 10 MB となる．ColumnFamily を分割せず，ただ 1 つの ColumnFamily 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 ColumnFamily 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．

本評価実験ではクライアントは検索クエリのみを送信する．これは提案手法が検索処理の応答性能向上を目的としているためである．このような読み出し処理のみのワークロードでは，ニュースアプリのような更新処理よりも検索処理の方が多く，1 度更新が行われた後は数多くのクライアントから検索処理のみが行われるというケースをシミュレートできると考えられる．

クエリ送信実験では，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 2,000 クエリを送信した．

© 2015 Information Processing Society of Japan

範囲検索クエリでは，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 1 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．本実験では，各 ColumnFamily ごとに約 10,000 件のデータが挿入されているため，1 度の範囲検索で検索対象となるノードは，1 ノードからクラスタを構成する全ノード数である 12 ノードとなる．検索クエリを送信するクライアント数を 125，250，500，1,000 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)　範囲検索の並列要求のみを実装した Cassandra

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

(P+RaS)　範囲検索の並列要求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要求，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 7**，範囲検索の平均応答時間を**図 8** 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 9**，範囲検索の平均応答時間を**図 10** に示す．

図 7 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 1,000 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．

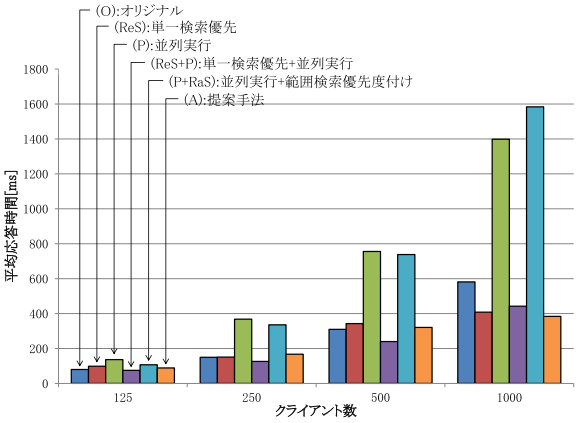

図 7　単一検索と範囲検索の比率 9 : 1 での単一検索の平均応答時間
Fig. 7　Average response time of single queries when sending single and range queries at a ratio of 9 : 1.

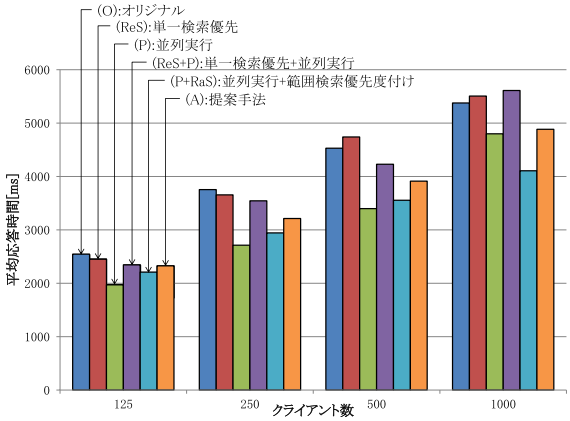

図 8　単一検索と範囲検索の比率 9 : 1 での範囲検索の平均応答時間
Fig. 8　Average response time of range queries when sending single and range queries at a ratio of 9 : 1.

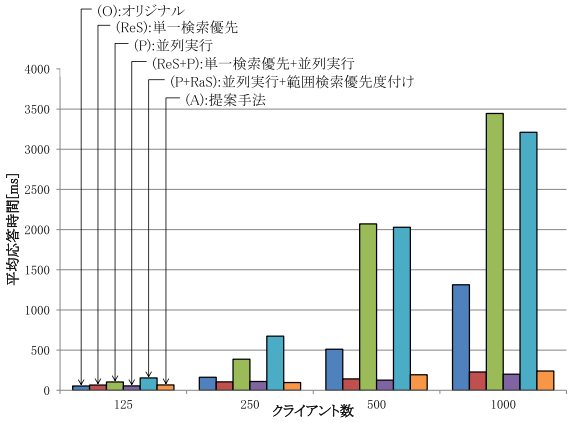

図 9　単一検索と範囲検索の比率 5 : 5 での単一検索の平均応答時間
Fig. 9　Average response time of single queries when sending single and range queries at a ratio of 5 : 5.

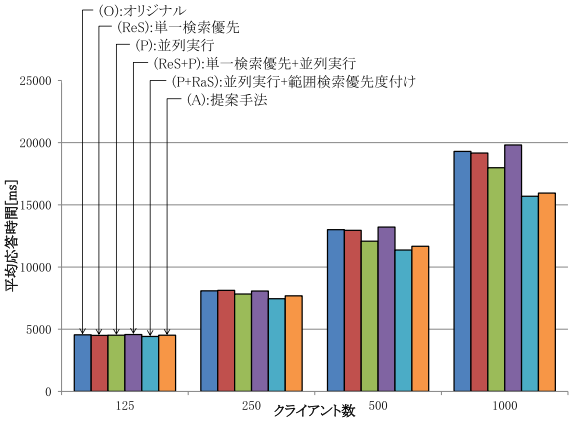

図 10　単一検索と範囲検索の比率 5 : 5 での範囲検索の平均応答時間
Fig. 10　Average response time of range queries when sending single and range queries at a ratio of 5 : 5.

© 2015 Information Processing Society of Japan

(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とも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 1,000 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要求により，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 8 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 1,000 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 1,000 の場合に並列要求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 9 より，(O) をはじめ単一検索を優先実行する実装が施されていない Cassandra の場合，クライアント数の増加にともない，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 1,000 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 1,000 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，範囲検索の並列要求により，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 10 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 1,000 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要求と，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要求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．

これら 3 つの手法のうち，範囲検索リクエストの並列要求のみを利用する場合は単一検索の応答性能の悪化に繋るため，単一検索リクエストの優先実行と併用して利用する必要がある．ただし 3 つの手法をすべて利用することで，単一検索と範囲検索両方の応答性能を向上させることができるため，すべてを利用することが望ましいと考えられる．本評価実験ではノード数を変更した場合の検証を行っていない．これは本提案手法のスケジューリングは各ノードで独立に行われるため，平均応答時間の短縮はノード数には依存しないためである．ノード数を増加させた場合でも，それに応じてクライアント数やデータ数を増加させることで，本評価結果と同様に検索クエリの平均応答時間を短縮することが可能であると考えられる．

6. 関連研究

Pirzadeh らは分散 Key Value Store における範囲検索の実装法の検証や性能の評価をした [8]．彼らは分散 Key Value Store の実装の 1 つである Voldemort 上で範囲検索処理を実現し，Cassandra，HBase での範囲検索との性能比較を行っている．性能評価より，多くのクライアントが検索範囲の広い範囲検索クエリを送信する場合にクエリの平均応答時間が増加していることが示されている．本研究ではクエリの優先度付けにより，このような平均応答時間の増加を抑制できると考えられる．

Escriva らは HyperDex と呼ばれる高性能，強い一貫性，高可用性および耐障害性を保証する新しい分散 Key Value Store を設計し，Cassandra，MongoDB と性能比較をしている [11]．HyperDex では，Key の prefix や suffix を Key 以外の属性 (secondary attributes) として保持することで，単一の Key に対する検索処理と同様の方法で範囲検索を実現している．この実装により，範囲検索を単一検索と同等の性能で実現している．しかし，HyperDex においても取得する Key の範囲やデータ数を大きくしていくことで，単一検索と範囲検索の性能差が出てくることは十分に考えられるため，そのような非常に負荷の高いワークロードにおいては本研究でのクエリの優先度付けが有効であると考えられる．

7. おわりに

分散 Key Value Store での検索の応答性能を改善するた

© 2015 Information Processing Society of Japan

めに，検索処理のスケジューリングと範囲検索の並列要求を提案をした．本研究では単一検索と範囲検索という 2 種類の検索クエリが混在する状況を想定し，それらのクエリに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 Key Value Store の実装の 1 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ともな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，これが改善されることを確認した．また範囲検索についても，並列要求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．今後の課題としては，より多くの実用的なワークロード，シチュエーションでの性能評価，さらなる応答性能の向上が期待できるスケジューリングの考案や，クエリの公平性についての考察などがあげられる．

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] Codd, E.F.: A relational model of data for large shared data banks, *Comm. ACM*, Vol.13, No.6, pp.377–387 (1970).

[2] Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., DeCandia, G., Hastorun, D. and Vogels, W.: Dynamo: Amazon's highly available key-value store, *ACM SIGOPS Operating Systems Review*, Vol.41, No.6, pp.205–220 (2007).

[3] Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R.E., Chang, F., Dean, J.: Bigtable: A distributed storage system for structured data, *ACM Trans. Computer Systems*, Vol.26, No.2 (2006).

[4] DangaInteractive: memcached, 2010, available from ⟨http://www.memcached.org/⟩.

[5] Lakshman, A. and Malik, P.: Cassandra – A decentralized structured storage system, *ACM SIGOPS Operating Systems Review*, Vol.44, No.2 (2010).

[6] Apache Cassandra, available from ⟨http://cassandra.apache.org/⟩.

[7] Karger, D., Lehman, E., Leighton, T., Levine, M., Lewin, D. and Panigrahy, R.: Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web, *ACM Symposium on Theory of Computing*, pp.654–663 (1997).

[8] Pirzadeh, P., Tatemura, J., Po, O. and Hacigumus, H.: Performance evaluation of range queries in key value stores, *J. Grid Comput.*, Vol.10, No.1, pp.109–132 (2012).

[9] Welsh, M., Culler, D. and Brewer, E.: Seda: An architecture for well-conditioned, scalable internet services (2001).

[10] Tam, E., Ramakrishnan, R., Sears, R., Cooper, B.F. and Silberstein, A.: Benchmarking cloud serving systems with YCSB (2010), available from ⟨http://github.com/brianfrankcooper/YCSB⟩.

[11] Escriva, R., Wong, B. and Sirer, E.G.: HyperDex: A distributed, searchable key-value store, *Proc. ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, SIGCOMM '12*, New York, NY, USA, pp.25–36, ACM (2012).

[12] Fukuda, S., Kawashima, R., Saito, S. and Matsuo, H.: Improving response time for cassandra with query scheduling, *2013 1st International Symposium on Computing and Networking* (*CANDAR*), pp.128–133 (2013).

福田 諭 （学生会員）

1991 年生．2013 年名古屋工業大学工学部情報工学科卒業．現在，同大学大学院工学研究科創成シミュレーション工学専攻博士前期課程在籍．分散システム等に興味を持つ．

川島 龍太 （正会員）

2005 年岩手県立大学ソフトウェア情報学部ソフトウェア情報学科退学．2007 年同大学大学院博士前期課程修了．2010 年総合研究大学院大学複合科学研究科情報学専攻博士課程修了．同年株式会社 ACCESS 入社．2013 年名古屋工業大学大学院助教，現在に至る．SDN，システムソフトウェア等の研究に従事．博士（情報学）．電子情報通信学会，IEEE 各会員．

齋藤 彰一 （正会員）

1993 年立命館大学理工学部情報工学科卒業．1995 年同大学大学院博士前期課程修了．1998 年同大学院博士後期課程単位習得中退．同年和歌山大学システム工学部情報通信システム学科助手．2003 年同講師，2005 年同助教授．2006 年名古屋工業大学大学院助教授，2007 年同准教授，現在に至る．オペレーティングシステム，インターネット，セキュリティ等の研究に従事．博士（工学），ACM，IEEE-CS 各会員．

© 2015 Information Processing Society of Japan

松尾 啓志　（正会員）

1983 年名古屋工業大学工学部情報工学科卒業．1985 年同大学大学院修士課程修了．同年松下電器産業（株）入社．1989 年同大学院博士課程修了．同年名古屋工業大学電気情報工学科助手．講師，助教授を経て，2003 年同大学大学院教授，現在に至る．分散システム，分散協調処理に関する研究に従事．工学博士．電子情報通信学会，人工知能学会，IEEE 各会員．

© 2015 Information Processing Society of Japan