Vol.2016-OS-138 No.16
2016/8/9

分散キーバリューストアにおける アクセス頻度を考慮した階層化ストレージ手法の提案

鴨下 将成[1]　川島 龍太[1]　松尾 啓志[1]

概要：大規模で高速なデータ処理を実現するため，スケールアウトが容易で分散処理に適している分散キーバリューストアが多くの企業で採用されている．二次記憶装置として，SSD を使用することで，応答遅延やスループットが改善するという報告がされている．しかし，SSD は HDD と比べてビットコストが 6 から 12 倍高いため，全てのデータを SSD に格納することは，コストパフォーマンスの点で課題がある．そこで，大量のデータを扱う分散キーバリューストアにおいて，アクセス速度が 2 から 3 倍以上異なる HDD と SSD といった二次記憶装置を併用する際は，低速な二次記憶装置に対するアクセスを最小限に抑える手法が必要である．本論文では，二次記憶装置として SSD と HDD を対象とする．データを保存する際には，そのデータに対するアクセス頻度を考慮し，アクセス頻度の比較的高いデータを SSD へ，比較的低いデータを HDD へ保存することで，保存先の二次記憶装置のアクセス速度を区別して保存する手法を提案する．提案手法の実装は，Apache Cassandra へ行った．その結果，保存先の二次記憶装置を選択する従来の Cassandra と比較して，提案手法では，平均スループットは約 15%の性能向上，平均読み出しレイテンシは約 15%の削減を確認することができた．また，保存するデータのアクセス頻度を考慮して SSD を用いることの効果も調査し，アクセス頻度の偏り具合がより大きい場合において，提案手法が有効であることも確認できた．

キーワード：分散キーバリューストア，階層化ストレージ，SSD，Cassandra

1. はじめに

近年，ビッグデータやクラウドコンピューティングの普及によって，大規模で高速なデータ処理が必要とされている [1]．そのため，従来のリレーショナルデータベースよりも，スケールアウトが容易で分散処理に適している分散キーバリューストアが，クラウドサービスを提供する多くの企業で採用されている [2]．

これまで，分散キーバリューストアの二次記憶装置としては HDD を用いることが一般的であったが，アクセス速度が高速な SSD を使用する調査が実施され，Apache Cassandra においては応答遅延が最大 20 倍，スループットは最大 80 倍改善するという報告がされている [3]．しかし，SSD は HDD と比べ，ビットコストが 6〜12 倍高い [4] ため，大規模で高速なデータ処理を行う際は，SSD のみを用いてシステムを構築することはコスト面において課題がある．そこで，大容量で高速な分散キーバリューストアの実現のため，データのアクセス頻度に応じて，アクセス頻度の比較的高いデータをアクセス速度が高速な SSD に配置して，異なる特長をもった二次記憶装置を使い分けることで性能向上を狙う研究が行われている [5][6]．

しかし，Apache Cassandra[3][7] では複数の二次記憶装置を利用可能だが，HDD と SSD のアクセス速度の特長を活かすように協調利用するための仕組みは実装されていない．本論文では，アクセス頻度を考慮して二次記憶装置を使い分ける機能を Cassandra に追加する．二次記憶装置として SSD と HDD を対象とし，データの書き込み時にはアクセス頻度の高いデータは可能な限り SSD を用いることで性能向上を図る．Cassandra では，二次記憶装置に保存されるデータは String Sorted Table (SSTable) というデータ構造で管理される．提案手法では，SSTable を生成する際，アクセス頻度の高いデータのみを SSD の SSTable に保存し，そうでないものを HDD の SSTable に保存する．こうすることで，SSD にアクセス頻度の高いデータが集中し，HDD に対するアクセスによる性能低下を抑制しつつ，データ容量の増加が実現可能である．

本論文の構成は以下のとおりである．まず，第 2 章では，研究背景として本論文で実装した分散キーバリューストア

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

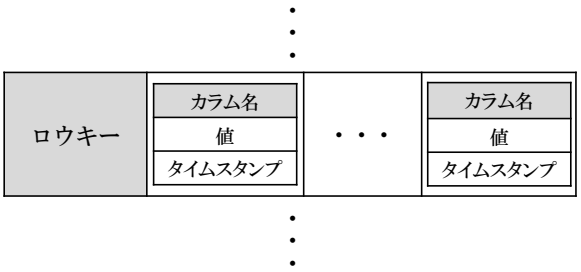

図 1 Cassandra のデータ構造

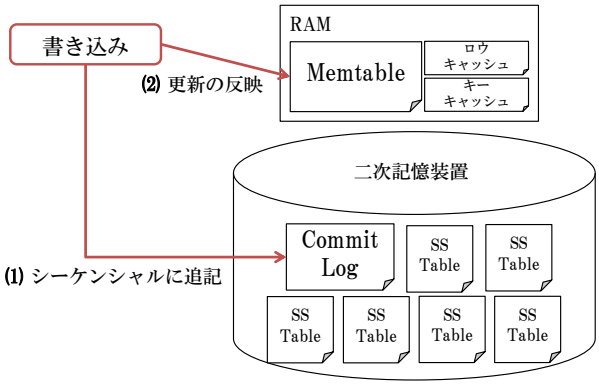

図 2 Cassandra の書き込み処理

である Apache Cassandra における，書き込みと読み込み動作について説明する．次に第 3 章で本論文での提案手法と，その実装について説明する．第 4 章で提案手法を適用したときの性能評価と考察を行い，第 5 章で関連研究を紹介する．最後に，第 6 章でまとめと今後の課題を述べる．

2. Apache Cassandra

2.1 概要

Apache Cassandra[7] は，Facebook によって開発されたオープンソースの分散キーバリューストアである．Cassandra は Amazon Dynamo[8] の Distributed Hash Table (DHT) と Google Bigtable[9] のデータモデルの特徴をそれぞれ併せ持っており，大規模なデータを管理するために設計されている．ノード間でのレプリケーションが可能で，さらに単一障害点もないため，高可用性を実現できる．以下では，Cassandra が二次記憶装置にデータを保存する際に用いるデータ構造と，具体的な書き込み，読み込み処理について説明する．

Cassandra では，図 1 に示すデータ構造を用いてデータを管理している．格納されるデータの最小単位であるカラムはカラム名，値，タイムスタンプを持つ．このカラムを Value とし，ロウキーを Key とした，Key と Value の組としてデータを格納し，これをロウとする．データのアクセス時には必ずロウキーを用いる．タイムスタンプはそのカラムが書き込まれたときの時刻を示しされている．

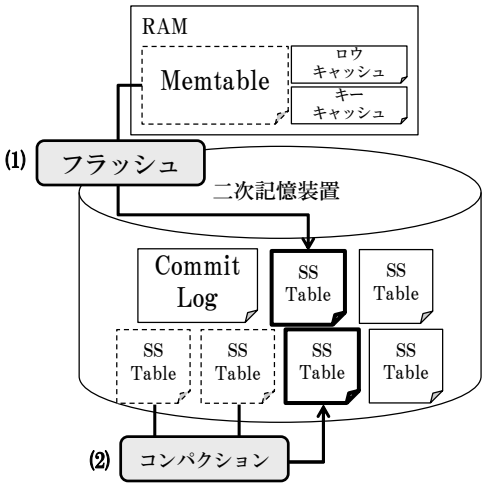

図 3 Cassandra のフラッシュ/コンパクション処理

2.2 書き込み処理

Cassandra の書き込み処理の概要を図 2 に示す．書き込みの際には，データの永続性を保証するため，追記型のログである二次記憶装置上の CommitLog に，Cassandra は書き込むデータのみをシーケンシャルに書き加える．こうすることで，高い書き込みパフォーマンスを実現している．また，一次記憶装置である RAM 上に, 二次記憶装置へのライトバックキャッシュのように動作する Memtable を持っており，CommitLog への書き込みが行われた後，図 2(2) で示すように，Memtable に対して書き込まれたデータの反映が行われる．書き込み対象となるロウキーが既に Memtable に存在する場合は，Memtable 上で書き込まれたデータへ直接更新され，存在しない場合には Memtable 上に新たに領域を確保し，データを書き込む．

この Memtable は，ユーザの指定した閾値を超える容量になると，二次記憶装置上へ SSTable として書き込まれるフラッシュ処理が行われる．図 3 において，Memtable が図 3(1) のフラッシュ処理によって SSTable として二次記憶装置に書き込まれる．このとき，書き込まれた SSTable の読み出しを効率的に行うため，キーの辞書順でソートされて保存される．二次記憶装置上で SSTable を構成する要素には，格納されているロウキーのインデックス，ブルームフィルタ [10]，そしてロウキーとカラムからなるデータなどがある．インデックスは目的のデータが保存されている SSTable ファイルの記録と，ファイルにおけるオフセットを記録するために使用され，ブルームフィルタは目的のキーが SSTable に存在するかどうかの確認に使用される．Cassandra では書き込み時のスループットの向上を重視した設計となっているため，書き込み時は Memtable と CommitLog に書き込むだけで，二次記憶装置上の SSTable にはアクセスしない．よってフラッシュの際に既に二次記憶上に古いデータが存在していたとしても，上書きは行われず，毎回新たな SSTable が二次記憶装置上に作成される．

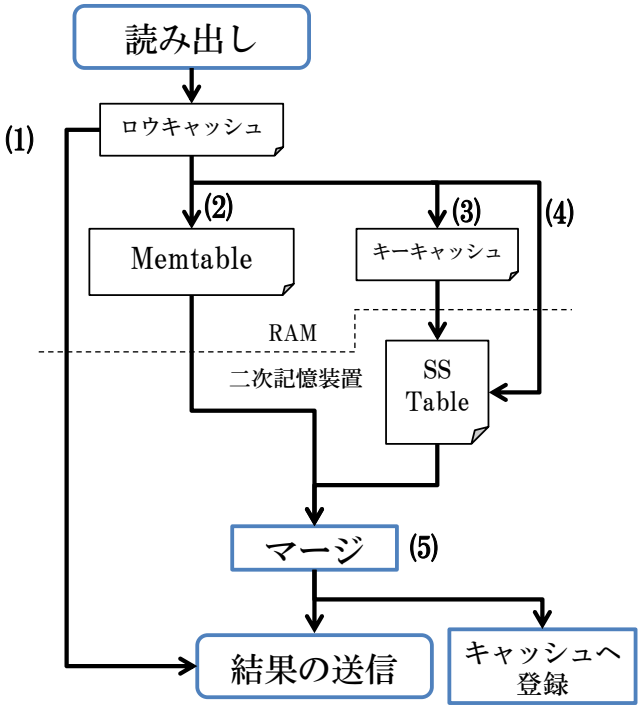

図 4 Cassandra の読み出し処理

2.3 読み出し処理

Cassandra の読み出し処理の概要を図 4 に示す．あるロウキーに対する読み出し要求を受け取ったノードは，まず RAM 上のロウキャッシュに該当するロウキーが存在するかどうかを確認する．Cassandra にはロウキャッシュとキーキャッシュの二種類のキャッシュが RAM 上に存在する．ロウキャッシュはロウ単位でキャッシュを行い，キーキャッシュは二次記憶装置内に保存されている SSTable のインデックスを保持している．ロウキャッシュにヒットした場合は，(1) のように RAM 上から直接読み出しが行われる．ヒットしない場合は，Memtable と SSTable の確認が並行して行われる．SSTable を読み出す際には対応するキーキャッシュを確認する．キーキャッシュで該当のロウキーがヒットすれば，対応する SSTable を読み出し，ヒットしない場合は，ノードに存在する全ての SSTable を対象として，ブルームフィルタを用いて対象のキーを含む SSTable を特定する．最後に，対象となるロウキーに対応するカラムが新旧複数存在した場合，それらをマージしてまとめ (5)，最新のタイムスタンプを持つカラムをクライアントへ返送する．

Cassandra では，フラッシュ回数の増加に伴って重複するデータが増え，読み出し時にマージしなければいけない新旧のデータが増えることで読み出しパフォーマンスが低下するという欠点がある [11]．この問題を解決するために，複数の SSTable が存在する場合は，それらの重複するデータをマージする作業であるマイナーコンパクションが行われる．

コンパクションを行う際の対象となる SSTable の選び方について述べる．コンパクションの対象となる SSTable は，SSTable のデータサイズを用いて決定される．ノードに様々なデータサイズの SSTable が存在する場合，それらをデータサイズの順にソートする．次に，データサイズの近い SSTable 同士で組を作成する．その後，各組が含む SSTable の累計の読み取り回数の合計が最も多い組がコンパクション対象となる．

表 1 パラメータの定義

$P(x)$	二次記憶装置 x を選択する確率
N	ノードの持つ二次記憶装置の集合
S_x	二次記憶装置 x における空き容量

2.4 ストレージ選択

Cassandra では，SSTable を保存する二次記憶装置として SSD を用いることで，性能向上することが確認されている [3]．加えて，複数の二次記憶装置を SSTable の保存先として設定することもできる．しかし，最大容量やアクセス速度において異なる特長を持つ二次記憶装置を，同時に利用することは考慮されていない．二次記憶装置を選択する条件は，表 1 のパラメータを用いた式 (1) である．

$$P(x) = \frac{S_x}{\sum_{n \in N} S_n} \quad (1)$$

二次記憶装置 x が選択される確率 $P(x)$ は，ノードの持つ二次記憶装置 N に含まれる全ての二次記憶装置 n の空き容量 S_n の合計を分母とし，二次記憶装置 x の空き容量 S_x を分子として計算される．よって，相対的に空き容量の多い二次記憶装置が高い確率で SSTable の保存先として選択される設計になっている．したがって，Cassandra で HDD と SSD の両方を用いる際，HDD と比較して相対的に最大容量の小さい SSD が保存先として選択される確率は低い．

3. 提案手法

3.1 概要

本論文では，Cassandra の SSTable に着目し，その保存先について HDD と SSD を階層化し，それらを区別してロウキー単位で動的に保存先を選択する手法を提案する．SSTable を書き込む際には，保存するロウのアクセス頻度に応じて保存先を HDD と SSD で選択する．この手法を用いると，任意の大きさの SSTable が作成可能であるため，SSD へ保存するロウを SSD に空き容量がある限り，任意の数だけ保存することが可能となる．また，フラッシュ時にもアクセス頻度に応じて保存先を選択するため，アクセス頻度の比較的低いデータは直接 HDD へ保存することが可能となる．したがって，限られた容量の SSD に比較的アクセス頻度の高いロウのみを保存するため，ロウに対してのアクセスに偏りがある場合に性能の向上が期待できる．以下では，SSTable が新たに作成される処理である，フラッシュおよびコンパクションの 2 つの場合の保存先の選択方

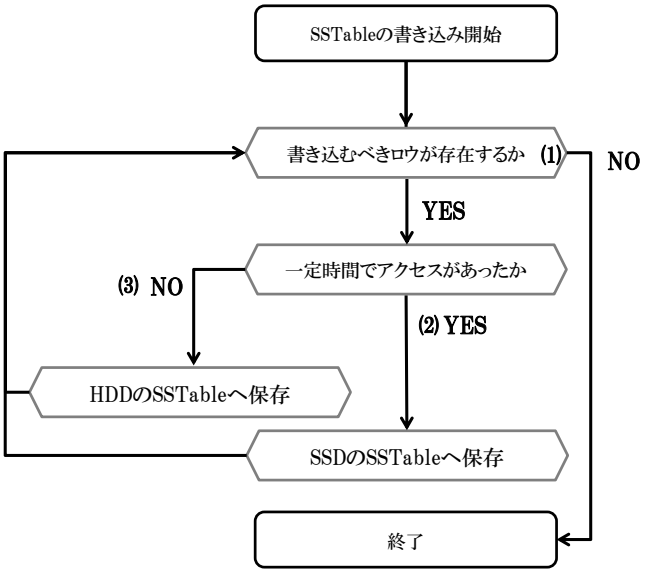

図 5 提案手法のフローチャート

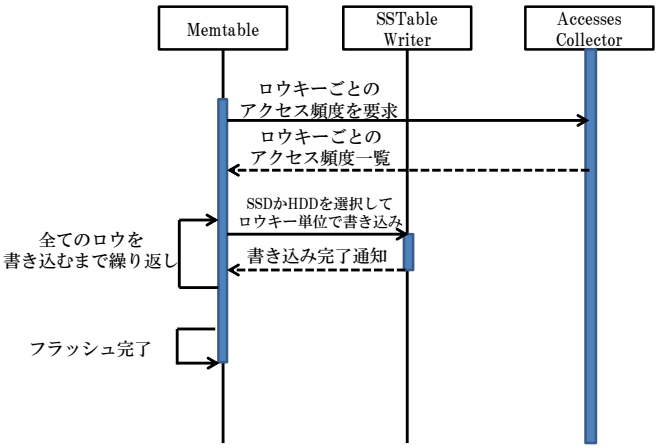

図 6 実装したフラッシュ処理

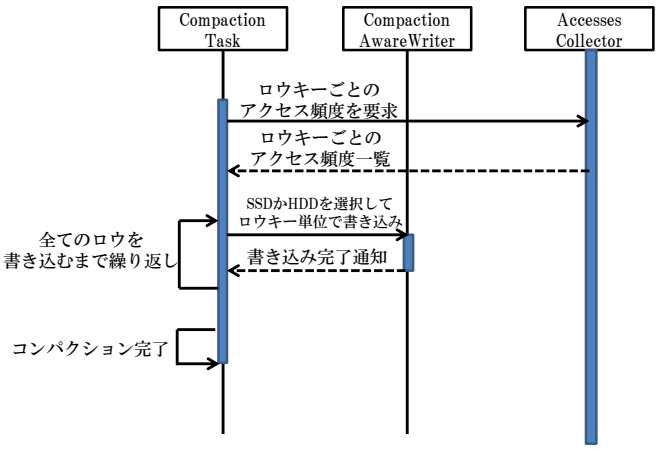

図 7 実装したコンパクション処理

法について説明する．

従来の Cassandra では，新しく SSTable を作成する際には，式 (1) を用いて SSTable の保存先を決定している．提案手法では，一定時間ごとにアクセスのあったロウキーについて，ユーザが決定したアクセス回数以上のアクセスがあったロウキーのみ記録し，それらをアクセス回数順でソートすることによってロウキーごとにアクセス頻度の情報を収集する．こうして求めたアクセス頻度の情報を用いて，新たに SSTable を作成する際は，ロウキー単位の一定時間のアクセス頻度に基づいて保存先を決定する．これを図 5 を用いて説明する．SSTable を新たに書き込む際，一定時間で一定回数以上のアクセスがあったかどうかをロウキー単位で確認する (図 5(1))．もしアクセスがあれば (図 5(2))，SSD の SSTable へ該当のロウを保存する．そうでなければ，HDD の SSTable へ保存する (図 5(3))．このように，提案手法では一度のフラッシュ，もしくはコンパクションで HDD と SSD にそれぞれ 1 つずつ SSTable を作成する．SSD に作成される SSTable には，一定回数以上のアクセスがあったロウのみが保存され，HDD に作成される SSTable には，一定回数未満のアクセスがあったロウのみが保存される．したがって，新たに SSTable を作成する際は，アクセス頻度の高いロウのみを SSD へ書き込むことが可能になる．

また，従来では対象となる SSTable の保存先は考慮されず，データサイズとアクセス回数のみを用いて対象の SSTable を選択していた．提案手法では，HDD に保存されている SSTable をコンパクションする際，累計のアクセス回数が最も多い SSTable の組をコンパクション対象として選択する．これは，HDD に保存されているアクセス頻度の高いロウをコンパクション対象とし，可能な限り SSD へ保存するためである．提案手法でコンパクションを行う際，SSD にロウを保存するだけの空き容量が存在しない場合では，コンパクション対象の SSTable を SSD に保存されている SSTable に限定して，コンパクション対象を選択する．これは，従来のコンパクションではデータサイズをもとにして対象を選択しているのに対して，提案手法では SSTable が保存されている二次記憶装置も考慮しなくてはならないためである．累計のアクセス回数のみではなく，二次記憶装置も考慮することで，HDD に保存されているアクセス頻度の高いデータに注目してコンパクション対象を選択することができる．

3.2 実装

提案手法を実装したフラッシュ処理について，図 6 に示す．Memtable でフラッシュ処理が開始されると，まず始めに AccessCollector へ最新のロウキーごとのアクセス頻度を要求する．AccessCollector は，3.1 章で述べたように，一定時間ごとにアクセスのあったロウキーについて，アクセス頻度の情報を Cassandra を実行している間は常に収集し続けている．これは，Cassandra では管理者用ツールとして利用できる nodetool で用いられている，topPartitions を用いて実装した．次に，受け取ったロウキーごとのアクセス頻度の一覧を用いて，SSTable として SSTableWriter

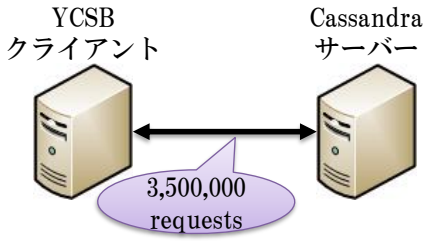

図 8 評価環境

で書き込みを行う際に，ロウキー単位で書き込み先を SSD もしくは HDD で選択する．アクセス頻度の一覧に書き込むロウのロウキーが含まれていれば，アクセス頻度が高いとみなして SSD へ保存する．含まれていなければ，アクセス頻度が低いとみなして HDD へ保存する．全てのロウを SSTable として書き込むまでこれを繰り返し，フラッシュ処理は完了する．コンパクション処理の場合については，図 7 で，CompactionTask がコンパクション対象の SSTable を受け取った段階から示す．フラッシュ処理と同様，AccessCollector へ最新のロウキーごとのアクセス頻度を要求し，CompactionAwareWriter で書き込みを行う際に，SSD もしくは HDD で選択する．全てのロウを書き込むと，コンパクションは完了する．

次に提案手法におけるコンパクション対象の決定方法について述べる．実装は Cassandra でデフォルトとして用いられる SizeTieredCompactionStrategy に行った．SizeTieredCompactionStrategy では，コンパクション対象の候補となる複数の SSTable を bucket という組で扱う．提案手法では，この bucket について，HDD に保存されている SSTable のみで構成される bucket，SSD に保存されている SSTable のみで構成される bucket，従来と同様の二次記憶装置を考慮しない SSTable で構成される bucket の 3 種類を用意する．コンパクションが実行されるスレッドのうち，少なくとも 1 つのスレッドでは HDD に保存されている SSTable のみの bucket をコンパクション対象とする．これは，アクセス頻度の低いロウとして一旦 HDD に保存され，その後アクセス頻度が高くなったロウでも，SSD へ移動することによって，ワークロードのアクセス分布が変化した場合でも性能が落ちることのないようにするためである．また，SSD に新しい SSTable を保存するだけの空き容量が無くなった場合では，SSD に保存されている SSTable のみの bucket を対象とする．これは，コンパクションによって SSD に保存されているロウで新旧の重複があるものを 1 つに統合することと，ワークロードのアクセス分布が変化した場合でも，SSD に保存されているアクセス頻度の低いロウを HDD へ追い出すようにするためである．

表 2 評価に用いた計算機

OS	Ubuntu12.04
CPU	Core i5-4460 / 3.2GHz
RAM	2GB
SSD	500MB, 1 台
HDD	500GB, 1 台

4. 評価実験

提案手法の有効性を確認するため，提案手法を実装した Apache Casandra 2.2.2 と NoSQL 向けのベンチマークである YCSB(Yahoo! Cloud Serving Benchmark)[12] を用いて性能評価を行った．実験に使用した計算機の性能は表 2 の通りである．実験環境は図 8 に示す．今回は単一ノードにおける性能向上を確認するため，Cassandra を動作させる計算機は 1 台とした．Cassandra で設定できる容量に関係するパラメータは全てデフォルト設定とした（ロウキャッシュは無効，キーキャッシュは有効）．YCSB を動作させる計算機として，表 2 と同じ性能のものを 1 台用意し，Cassandra と同じネットワーク上に配置して評価を行った．YCSB において実行するワークロードは Read 比: 50% Write 比: 50%で，想定アプリケーションはセッションストアである．YCSB で挿入する 1 つのレコードのデータサイズは，13 bytes のキーと 1 つ 100bytes のフィールドが 10 個のデフォルトの条件を用いた．この条件のレコードを 25,000,000 レコード挿入し，ワークロードで実行される命令数を 3,500,000 命令とした．命令分布はアクセス頻度に偏りのある，zipfian 分布とした．YCSB では，命令分布のアクセス頻度の偏りを示すパラメータを変更することによってアクセス頻度の偏り具合を制御すること可能である．本評価では，このパラメータが s=0.99 の場合と，より偏り具合の小さい s=0.90 の場合の 2 種類のアクセス頻度の分布で評価を行った．HDD のみで構成されたシステムへ SSD を導入する環境を想定するため，ワークロードの初期状態では，データは全て HDD に保存されているものとした．アクセス回数の情報については，10 秒ごとに一度でもアクセスがあったロウキーを，提案手法におけるアクセス頻度が高いロウキーと設定した．

式 (1) の確率に基づいて二次記憶装置を選択する従来の Cassandra，アクセス頻度を考慮して SSD を用いる提案手法を実装した Cassandra，提案手法においてアクセス頻度を考慮しないが，SSD を優先的に用いる実装をした Cassandra(SSD を優先) で，10,000 秒間におけるスループットの変動を 100 秒間の平均スループットでプロットし，s=0.90 で評価した結果を図 9，s=0.99 で評価した結果を図 10 に示す．提案手法において，アクセス頻度を考慮することなく SSD の優先度を最大化した場合と比較して，s=0.90 で評価した場合では 5,000 秒付近から，s=0.99

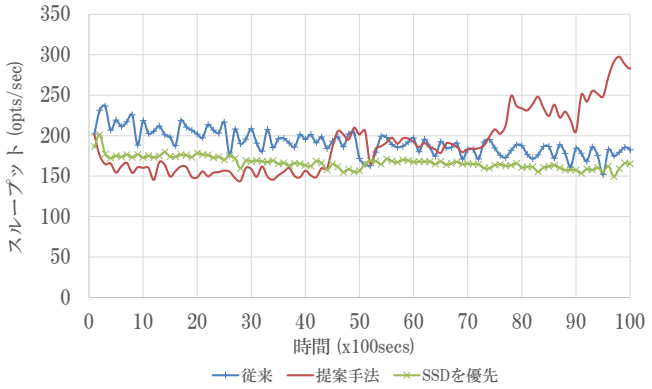

図 9 スループットの時間変化での比較 (s=0.90)

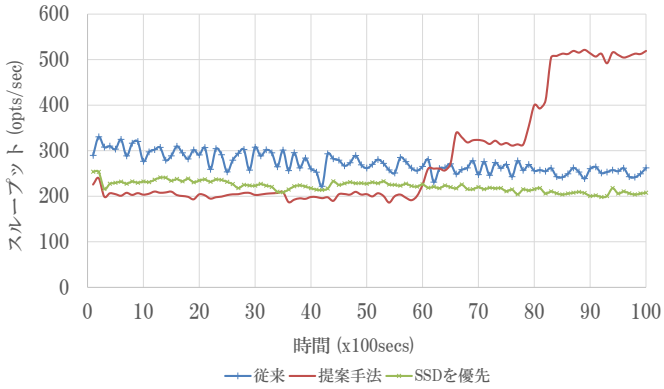

図 10 スループットの時間変化での比較 (s=0.99)

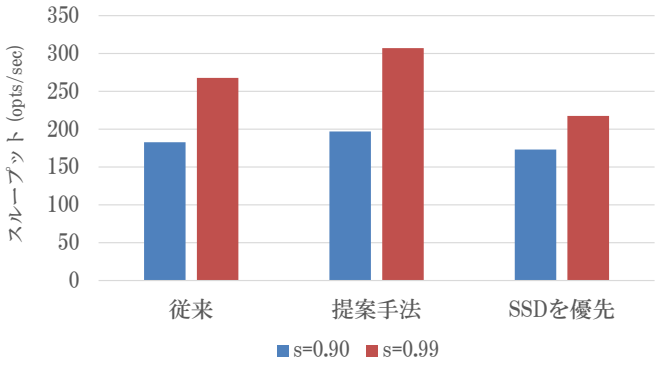

図 11 平均スループットの比較

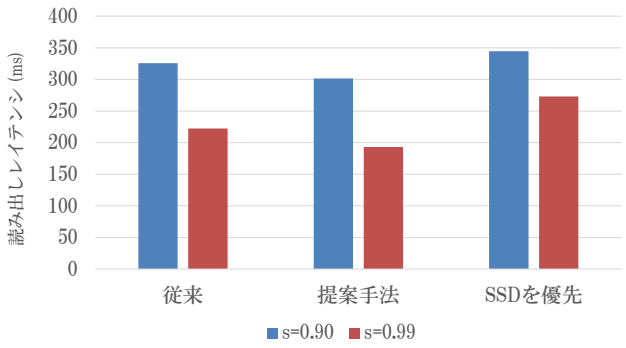

図 12 平均読み出しレイテンシの比較

で評価した場合では 6,000 秒付近からスループットが向上していくことが確認した．これは，アクセス頻度を考慮することによって，SSD からの読み出し回数が増加したこ

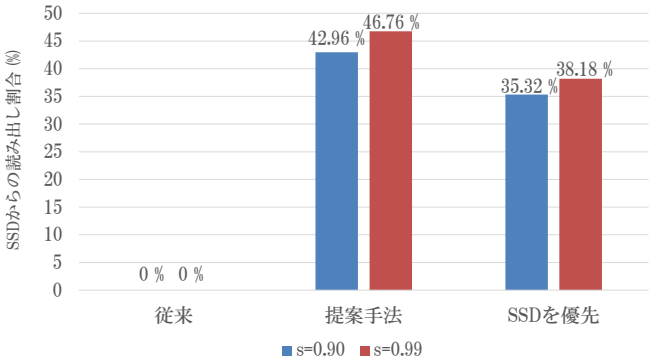

図 13 SSD からの読み出し割合の比較

とが理由として考えられる．SSD を優先する場合，どちらにおいても従来と比較して性能が悪化しているのは，提案手法では従来よりもコンパクション処理が多く発生するため，読み出しのスループットがコンパクション処理の頻繁なディスク I/O に影響を受けたためであると考えられる．次に，アクセス頻度の偏り具合を変化させた場合の，平均スループットを評価した結果を図 11 に示す．提案手法において，偏りの小さい s=0.90 の場合と比較して，偏りの大きい s=0.99 の場合では約 50%のスループット向上が確認できた．アクセス頻度の偏り具合が大きいほど，SSD を優先する場合と比較してスループットが向上することも確認できた．

平均読み出しレイテンシを評価した結果を図 12 に示す．提案手法では，読み出しレイテンシが従来や SSD を優先する場合と比較して削減できていることがわかる．s=0.99 の場合，s=0.90 の場合と比較して従来では約 31%の削減に対して，提案手法では約 36%読み出しレイテンシが削減できたという結果となった．

最後に，SSD からの読み出し平均回数を評価した結果を図 13 に示す．ワークロードにおいて，全ての読み出しリクエストに対して SSD から読み出された割合を表している．s=0.90 と s=0.99 のどちらの場合においても，提案手法では SSD を優先する場合と比較して，多くのデータが読み出されていることを確認した．つまり提案手法では，比較的アクセス頻度の高いデータを中心に SSD へ保存していることがわかる．また，従来の Cassandra では，SSD からの読み出し平均回数は 0%となった．これは，従来の Cassandra では空き容量に基づく確率で二次記憶装置を選択しているため，HDD と比較して相対的に空き容量の少ない SSD は保存先として選択されなかったためである．

5. 関連研究

文献 [5] では，SSD と HDD を協調利用することで Cassandra における容量の増加と性能の向上を実現している．具体的には，次の 2 つの手法が提案されている．一つ目は RAM 上に存在するロウキャッシュの拡張として SSD 上に 2nd ロウキャッシュを設けることで，ロウキャッシュの容

量を増加させる手法である．しかし，ロウキャッシュはロウキーに対して更新操作ほとんど行われないという状況でしか用いることが推奨されていないため [13]，性能向上に貢献しないと言える．二つ目は，SSTable のデータからカラム名だけを抜き出し，スキーマカタログとして SSD に保存する手法である．多くの場合複数のロウではカラムが共通であるため，SSTable 上の複数のロウで重複したカラム名が格納されることになる．しかし，この手法では，読み出し要求されたデータは SSD ではなく HDD に保存されているため，アクセス頻度を考慮して HDD と SSD を区別して使い分けていない．

また，LevelDB[14] で複数ストレージを使用し，SSD と HDD の階層化を実装する研究もなされている [6]．既存研究では，LevelDB で用いるアーキテクチャ上で上位 2 つのレベルを SSD へ，それ以降のレベルを HDD へ配置することで，SSD と HDD で保存先ストレージを階層化して扱っている．しかし，任意のデータサイズで階層化できないことや，アクセス頻度の低いデータも必ず上位のレベルを経由してから下位のレベルへ移動されるという問題がある．

6. まとめ

本論文では，分散キーバリューストア Apache Cassandra 上でのアクセス頻度を考慮した階層化ストレージ手法を提案した．SSTable を階層化の対象とし，SSTable が新たに作成されるフラッシュとコンパクションの二つの処理において，性能向上を実現するため，アクセス頻度に基いて SSTable の保存先を SSD と HDD の中から適切に選択する手法を提案し，実装および評価を行った．評価の結果，従来の Cassandra と比較して，提案手法を実装した Cassandra では，Read 比: 50% Write 比: 50%のワークロードで s=0.9 の場合において，約 50%の平均スループットの向上が確認できた．平均読み出しレイテンシにおいては，アクセス頻度の偏り具合を大きくした場合，提案手法では約 36%のレイテンシ削減が確認できた．

今後の課題としては，提案手法を実装した際のコンパクションによるオーバーヘッドの削減や，より早い段階からスループットに変化が見られるよう，ロウの移動処理を最適化することなどが考えられる．今回は SSD と HDD の性能比較差のみを詳細に検討するため，ノード 1 台の環境で実験を行ったが，今後はより実際的な環境として，複数台のノードによる実験を行う予定である．また，ReadHeavy，WriteHeavy のワークロードや，読み出し割合が実行中に変化するような現実的なワークロード，より長時間のワークロードでの評価実験，SSD の容量が異なる場合での評価実験なども予定している．

参考文献

[1] B. Brown J. Bughin R. Dobbs C. Roxburgh J. Manyika, M. Chui and A. H. Byers. Big data: The next frontier for innovation, competition, and productivity. Technical report, McKinsey Global Institute, 2011.

[2] A. E. Abbadi D. Agrawal, S. Das. Big data and cloud computing: current state and future opportunities. In *Proc. 14th International Conference on Extending Database Technology(EBDT)*, pp. 530–533, NY, USA, March 2011.

[3] Western Digital Technologies. Accelerating cassandra workloads using sandisk solid state drives, 2015.

[4] L. Fong W. Tan and Y. Liu. Effectiveness assessment of solid-state drive used in big data services. In *Proc. 21th IEEE International Conference on Web Services (ICWS)*, pp. 393–400, Alaska, USA, July 2014.

[5] M. Sadoghi P. Menon, T. Rabl and H.-A. Jacobsen. Cassandra: An ssd boosted key-value store. In *Proc. 30th IEEE International Conference on the Data Engineering (ICDE)*, pp. 1162–1167, Chicago, IL, USA, March 2014.

[6] M. Sadoghi P. Menon, T. Rabl and H.-A. Jacobsen. Optimizing key-value stores for hybrid storage architectures. In *Proc. 24th Annual International Conference on Computer Science and Software Engineering(CASCON)*, pp. 355–358, Ontario, Canada, Jun 2014.

[7] A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review(OSR)*, Vol. 44, No. 2, pp. 35–40, April 2010.

[8] M. Jampani G. Kakulapati A. Lakshman A. Pilchin S. Sivasubramanian P. Vosshall G. DeCandia, D. Hastorun and W. Vogels. Dynamo: amazon's highly available key-value store. In *Proc. 21st ACM Symposium on Operating Systems Principles(SOSP)*, Vol. 41, pp. 205–220, Stevenson, WA, October 2007.

[9] S. Ghemawat W. Hsieh D. Wallach M. Burrows T. Chandra A. Fikes F. Chang, J. Dean and R. Gruber. Bigtable: A distributed storage system for structured data. In *Proc. 7th Symposium on Operating System Design and Implementation(OSDI)*, pp. 205–218, Seattle, WA, November 2006.

[10] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, Vol. 13, No. 7, pp. 422–426, 1970.

[11] 中村俊介, 首藤一幸. 読み出し性能と書き込み性能を両立させるクラウドストレージ. 研究報告システムソフトウェアと オペレーティング・システム (OS), Vol. 2011, No. 24, pp. 1–9, 2011.

[12] E. Tam R. Ramakrishnan B. F. Cooper, A. Silberstein and R. Sears. Benchmarking cloud serving systems with ycsb. In *Proc. 1st ACM symposium on Cloud computing(SOCC)*, pp. 143–154, Indianapolis, IN, June 2010. http://github.com/brianfrankcooper/YCSB.

[13] データ・キャッシュの構成. http://docs.datastax.com/ja/cassandra-jajp/2.0/cassandra/operations/ops_configuring_caches_c.html.

[14] Leveldb, a fast key-value storage library by google. https://github.com/google/leveldb.