

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

中 村 俊 介^{†1} 首 藤 一 幸^{†1}

既存のクラウドストレージは読み出し性能または書き込み性能のいずれか一方を重視した設計がなされている。この性能の性質は分散のための機構とは独立しており、ストレージエンジンに大きく依存する。我々はこの点に注目し、同一のシステム内でストレージエンジンの選択が可能でモジュラーなクラウドストレージ MyCassandra を開発してきた。しかし、MyCassandra は既存のクラウドストレージと同様に読み出しと書き込みの性能を両立させるということはできていない。

そこで本研究では、読み出し性能と書き込み性能の両立が可能なクラウドストレージ MyCassandra Cluster を提案する。MyCassandra Cluster は、異なるストレージエンジンの MyCassandra ノードを組合せたクラスタを構成する。そして、それぞれの各ノードが得意とするリクエストを同期的に処理し、得意でないリクエストを非同期で処理するように、プロキシで割り振る。これにより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が構築可能である。

元の Cassandra と比較したところ、Cassandra と同等の高い書き込み性能を保ちつつ、読み出しの遅延は最小 11.6%、全体のスループットは最大 6.53 倍となることを確認した。

A Cloud Storage Supporting both Read-Intensive and Write-Intensive Workloads

SHUNSUKE NAKAMURA^{†1} and KAZUYUKI SHUDO^{†1}

A cloud storage with persistence perform well only with a read-heavy or write-heavy workload. There is a trade-off between read-optimized and write-optimized design of a cloud storage. It is dominated by the storage engine, which is a software component managing data on disks. A storage engine is an independent component and it can be pluggable with an adequate software design though today's cloud storages are not always modular. We have developed a modular cloud storage MyCassandra to demonstrate that a cloud storage can be read-optimized and write-optimized by a modular design. Various types of storage engines can be introduced to MyCassandra and determined with what

workload MyCassandra performs well. MyCassandra proved that the modular design enables a cloud storage to adapt to workloads.

In this paper, we present a technique to build a cloud storage performing well with both read-heavy and write-heavy workloads. A MyCassandra Cluster consists of MyCassandra nodes with different types of storage engines, read-optimized, write-optimized and on-memory (read-and-write-optimized). A query is routed to nodes which process the query efficiently. A cluster maintains consistency between data replicas with a quorum protocol. A MyCassandra Cluster showed comparable performance with the original Cassandra with write-heavy workloads, and it showed considerably better performance with read-heavy workloads. In comparison with Cassandra, read latency was 11.6% and read throughput was 6.53 times.

1. はじめに

RDBMS が満たしきれない要求に応える分散データストアとして NoSQL や key-value store といったクラウドストレージが注目されている。これらの共通の特徴は従来の RDBMS と比べてデータモデルやクライアント側の機能を制限し、また整合性についての条件を緩めることで、負荷の分散を比較的容易にし、ノード数をスケールアウトさせやすい点である。

しかしながら、それぞれのクラウドストレージは様々な点で異なっている。例えば、データモデルには key-value 形式や多次元マップ形式があり、分散の構成は master/worker 型、ツリー型、非集中分散型があり、データを全てメモリ上で保持するものや永続化が可能なものの、複製の配置方法やそれを同期/非同期で行うかなどいうように様々な設計方針がある。

また、従来のクラウドストレージは書き込み性能重視のものと読み出し性能重視のものがある。例えば、Apache Cassandra¹⁾²⁾ や Apache HBase³⁾ は書き込み性能重視であるが、一方、Yahoo Sherpa⁴⁾ や sharded MySQL⁵⁾ (MySQL の sharding 構成) は読み出し性能重視である。この性質の違いは、分散のための機構には依存せず、システム内のストレージエンジンに大きく依存する。我々はこの点に注目し、前者と後者にモジュール化し、同一システム内で後者のストレージエンジン部分を選択可能な分散データストア MyCassandra⁶⁾ を開発し、実際にストレージエンジンの選択によって、書き込み性能と読み出し性能が大きく変わることを確認してきた。

^{†1} 東京工業大学
Tokyo Institute of Technology

しかし、MyCassandra も読み出しと書き込みのどちらか一方の性能しか重視できていないという点では、既存のクラウドストレージと同じである。そこで、我々はこの欠点を補う為に、MyCassandra Cluster を提案する。すなわち、異なるストレージエンジンの MyCassandra ノードを組合せたクラスターを構築し、読み出し性能または書き込み性能に最適化されたストレージエンジンへ適切にリクエストの振り分けを行わせることで、同一クラウドストレージ内で読み出しと書き込み両方の性能を両立できることを示す。

本稿の構成は以下のとおりである。2章で研究背景として、開発に用いた Apache Cassandra と開発した MyCassandra を説明する。3章で MyCassandra Cluster について提案する。4章で性能評価に用いたベンチマーク YCSB について述べた後に、性能測定の結果を示して考察する。5章で性能面以外について議論する。6章で本研究に関連するモジュラーなクラウドストレージと読み出し/書き込み性能の両立に関する研究を挙げる。7章で本研究の貢献と今後の課題をまとめる。

2. 研究背景

クラウドストレージを用いる大規模なシステムでは、次にどのデータが参照・更新されるかを予測することは難しい。そのため、全てのデータがメモリ上に収まらない限りは読み出し時にディスクへのランダム I/O が発生してしまう。書き込み時にランダム I/O を行うよりも、ディスクへのシーケンシャル I/O のみで書き込み処理を済ませるログ記録方式の方が、高いスループットを実現できるが、一方このような方式では、読み出し時はログから一連の更新結果を拾い集める必要があるために効率が悪い。つまり、既存のクラウドストレージには読み出し性能と書き込み性能の間にトレードオフがある。

このようなトレードオフを考慮した上で、同一のクラウドストレージ内で読み出し/書き込み性能の両立を図るためには、それぞれの処理に最適化されたストレージエンジンを持つノードの組合せによって、クラスターを構成すれば良いと考えられる。

そもそも、既存のクラウドストレージにはストレージエンジンを選択するという自由度は無く、固定された設計がなされている。そこで、我々はストレージエンジンの選択が可能であるモジュラーなクラウドストレージ MyCassandra を開発してきた。本研究ではこの MyCassandra をベースとして、読み出し/書き込み性能を両立させる MyCassandra Cluster を提案する。

以下では、実装のベースに用いた Apache Cassandra, MyCassandra を説明する。

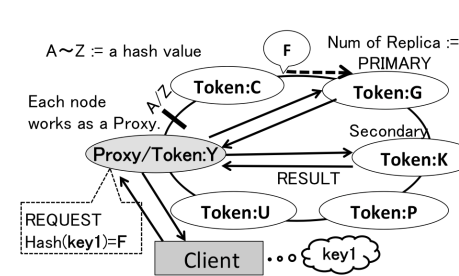


図 1 Consistent hashing
Fig. 1 Consistent hashing

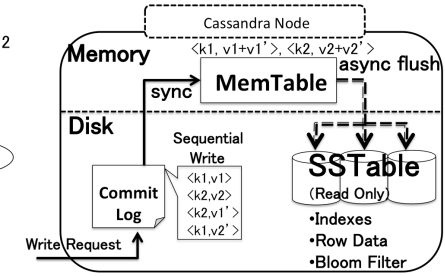


図 2 Cassandra の書き込みの流れ
Fig. 2 Write process in a Cassandra node

2.1 Apache Cassandra

Apache Cassandra は、Facebook 社が開発し、Apache Project としてオープンソース化したクラウドストレージである。複数のデータセンターにまたがる数百台で運用可能なスケラビリティや、非集中で単一故障点の無いことによる高い可用性を特徴とする。

2.1.1 Consistent Hashing

Cassandra は consistent hashing というアルゴリズムを用いてノードに対してデータの割り当てを行う。具体的には、リング状の ID 空間（図 1）を考え、ノードとデータの双方に ID を割り当てる。例えば、リクエストのキー（key1）のハッシュ値が F である場合は、ID 空間上で F から時計回りに最もノード ID が近いノード G が担当し、リクエストに対して応答する。Cassandra ではノード ID を Token、担当ノードをプライマリノードと呼ぶ。

consistent hashing の利点はハッシュ関数により各ノードの担当データ数が均等になり易く、負荷分散が容易であることと、データの担当範囲を集中管理するサーバーが存在しないので、単一故障性が無く、高い可用性を提供できることである。各ノードは gossip プロトコルによる定期的な情報交換により他の全ノード ID と IP アドレスを把握することで、任意のノードがプロキシとしてクライアントの要求に応じることができる。

2.1.2 Bigtable のストレージエンジン

Cassandra のストレージエンジンは Log Structured Merge Tree⁷⁾ を採用した Google Bigtable⁸⁾ のそれを模しており、Commit Log, MemTable, SSTable から構成される。

書き込み処理の流れ（図 2）を以下に示す。

- (1) まず、永続化の為にディスク上の Commit Log に行に対する差分をシーケンシャルに書きこむ

- (2) 次に、読み出し性能の為にメモリ上のマップである MemTable を更新し、クライアントに応答する
- (3) MemTable が保持するデータ量または未フラッシュ時間についての閾値を超えた場合に、古いデータをディスク上に SSTable として書きだす (フラッシュする)。この際、書きだしたデータは、Commit Log から削除する

この方式の利点は、ディスクへの書き込みが常にシーケンシャルで高速である点と、一度ディスク上に書かれた内容は変更されないため、ディスクに対する書き込みロックが不要となり、常に書き込み要求に応答可能である点である。一方、欠点は、読み出し時に、指定 key の差分データを複数の SSTable から読み出してマージする処理が必要となる為に複数のランダム I/O が発生してしまい、読み出し性能が犠牲になる点である。データは必ず Commit Log か SSTable 上にデータがあるため、永続性は保たれる。

2.1.3 レプリケーション

Cassandra の複製は、リング状の ID 空間においてプライマリノードの時計回り方向 $N-1$ ノードに配置される。例えば図 1 において、レプリカ数 2 の場合はハッシュ値 F のデータはノード G とその右隣のノード K に配置される。ラックやデータセンターを考慮した複製の配置方法も選択できる。

2.2 MyCassandra

MyCassandra は、Cassandra をベースとし、「分散のための機構」と「ストレージエンジン」に分離したクラウドストレージである。これにより、ストレージエンジンの選択を可能とした。

図 3 に、Cassandra と MyCassandra での各ノード内でのデータの読み書きに関する部分を示す。MyCassandra は、Cassandra のリクエストを受理する部分と各ストレージエンジンとの間に Storage Engine Interface を設けている。このインタフェースが規定する関数 (connect, put, get) を実装することで MyCassandra のストレージエンジンとして追加できる。現在、MyCassandra は Bigtable, MySQL, Redis という性質の異なる 3 つのストレージエンジンを備える。

2.2.1 ストレージエンジン

MyCassandra で選択可能なストレージエンジンについて述べる。

Bigtable ストレージエンジンは、2.1.2 で紹介した Cassandra のログ構造のクラウドストレージをそのまま用いている。

MySQL ストレージエンジンへのアクセスには JDBC を使用する。本研究では、MySQL

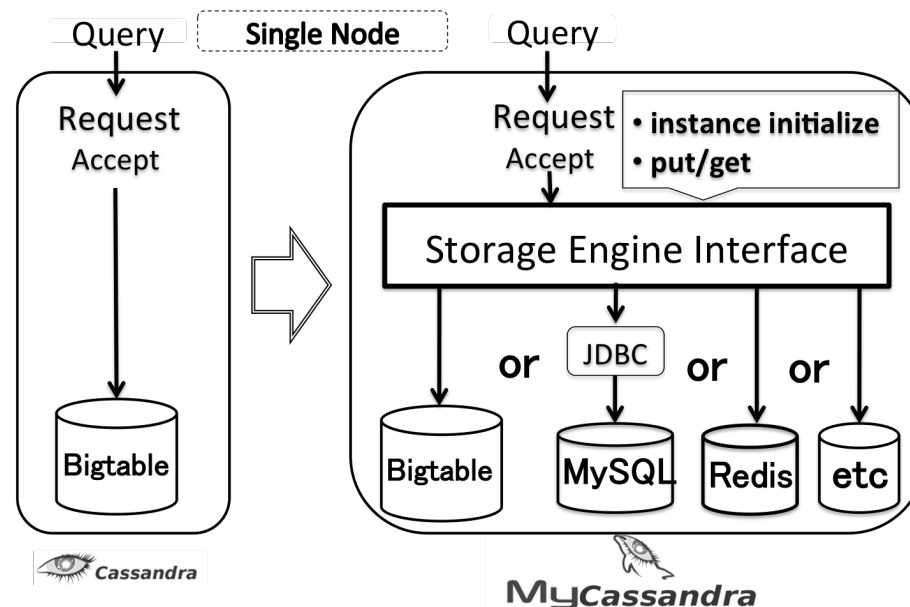


図 3 MyCassandra の Storage Engine Interface
Fig. 3 Storage Engine Interface of MyCassandra

内ストレージエンジンとして InnoDB を用いる。また、ストアードプロシージャを用いて、ストレージエンジンとしては不要である SQL 文のパーサを回避している。

Redis⁹⁾ ストレージエンジンは memcached と同様に key と value のペアをメモリ上に保持するメモリベースの key-value store である。スナップショットをディスクに非同期で書きこむ永続化機能や、メモリによる制限を緩和する仮想メモリ機能をもつ。ただし、本研究ではこれらの機能は使わず、単純なメモリベースのストレージエンジンとして用いる。アクセスには Redis 固有の API を用いる。

3. 提案手法・設計

本研究では、読み出し性能と書き込み性能の両立が可能なクラウドストレージ MyCassandra Cluster を提案する。本節ではその手法と設計について説明する。MyCassandra Cluster は、ストレージエンジンの異なる 3 種類のノード、すなわち、書き込み性能重視、読み出し

性能重視、メモリベースのノードから構成される。

データの複製をこれら3種類のノードに配置し、書き込みリクエストは書き込み性能が高いノードに振り分け、読み出しリクエストは読み出し性能が高いノードに振り分ける。これにより書き込み性能と読み出し性能の両立を達成できる。

複製間の整合性は、quorum (多数決) プロトコルで保つ。これは元の Cassandra が提供する機能である。MyCassandra Cluster は、3種類のノードに複製を置くので、複製数 N は3以上である必要がある。そして例えば $N=3$ の場合は、読み出しの合意数 R と書き込みの合意数 W が2以上であれば、quorum プロトコルを満たす。

3.1 ノード配置と負荷分散

MyCassandra は、Cassandra 同様、consistent hashing を用いてデータの担当ノードを決める (2.1.1 節, 2.1.3 節)。今回の実験ではデータ量を均等に分散させるためにノード ID を等間隔に固定して割り当てる。データの複製は、リング状 ID 空間においてプライマリノードから時計回りに N ノードを選択して配置する。その際、ストレージエンジンの異なる3種類のノードがひと通り含まれるように、 N ノードを選択する。

また本手法では、負荷を均一にするために1台のホストに書き込み性能重視、読み出し性能重視、メモリベースの3つの MyCassandra ノードを配置する方針をとる。つまり、各ホストはいずれかの種類のプライマリノードと、それとはストレージエンジンの種類が異なる2つの複製ノードを配置する。これは、各ホストに1ノードずつ配置するナイーブな方法では、1) ワークロードによって負荷が不均一になってしまい、また2) 資源を十分に生かすことができないからである。1) は、例えば読み出しの比率が多いワークロードの場合は、読み出しに割り振られるノードに負荷が集中してしまう。2) は、全データをメモリベースのノードのディスクが使われず、資源が無駄となる。また、1ノードに複数のストレージエンジンを持たせ、その種類ごとにリング状 ID 空間に仮想ノード¹⁴⁾として、割り当てる設計方針も考えられるが、ストレージエンジンの起動・削除といった運用の容易さから別プロセスとして動かすべきと考えた。

1台のホストに複数のノードを置く際、同一のホストに複数の複製が配置されないようにする必要がある。つまり、プロキシが、複製を配置するノードを決める際、異なるホスト上に配置された N ノードを選ぶようにする。具体的には、プライマリノードから時計回り N ノードを選ぶ際、既に選択済みノードと同一ホストに配置されるノードをスキップする。

3.2 メンバシップ管理

MyCassandra Cluster では、クライアントが接続した任意のノードがプロキシとしてス

トレージエンジンを考慮した適切なリクエストの振り分けを行う仕組みが必要となる。そのため全ノードが全ノードのストレージエンジンの種類を把握する必要がある。

そこで、本実装では Cassandra のメンバシップ管理に用いられる gossip プロトコルを利用する。Cassandra では、gossip プロトコルを用いてハートビートを含んだメッセージを他のノードと定期的に交換することで、メンバシップ管理を行う。MyCassandra Cluster では、このメッセージにストレージエンジンの種類を加えることで、互いにこの種類を把握できるようにしている。

3.3 読み出しと書き込み

図4に MyCassandra Cluster の読み出しと書き込みの流れを示す。書き込み/読み出し性能を両立させるために、書き込みリクエストは書き込み性能が高いノードに振り分け、読み出しリクエストは読み出し性能が高いノードに振り分ける。データの整合性を保ちながら性能を確保するために、quorum プロトコルという多数決による手段を用いる。以下、 N は複製数、 W は書き込み合意数、 R は読み出し合意数を表す。

ここで具体的な書き込み/読み出し処理の流れを説明する。

書き込み処理の流れを以下に示す。

- 1) データ担当ノードのうち、書き込み性能重視のノードとメモリベースのノードの合計 W ノードに書きこんで、成功すれば ACK をクライアントに知らせる
- 2-a) 成功した場合、読み出し性能重視の $N - W$ ノードには非同期で書きこむ
- 2-b) 失敗した場合、読み出し性能重視のノードも含めた合計 W ノードからの書き込みも待ってから ACK をクライアントに知らせる

手順1が失敗しなければ、読み出し性能重視なノードへの書き込み遅延は全体の書き込み性能には影響を与えない為、遅延は書き込み性能重視なノードのものに抑えられる。

読み出し処理の流れを以下に示す。

- 1) データ担当ノードのうち、読み出し性能重視のノードとメモリベースの合計 R ノードからデータを取得し、整合性を確認する
- 2-a) もし、読み出しが成功し、整合性がとれていれば、データをクライアントに返す
- 2-b) もし、読み出しの失敗または整合性がとれていなければ、書き込み性能重視も含めて合計 R ノードの整合性の取れたデータを待ち、データをクライアントに返す
- 3) 残りの $N - R$ ノードのデータの取得を待ち、もし整合性のとれていない古いデータを保持するノードがあれば、そのノードへの更新を非同期で行う (元の Cassandra の read repair 機能)

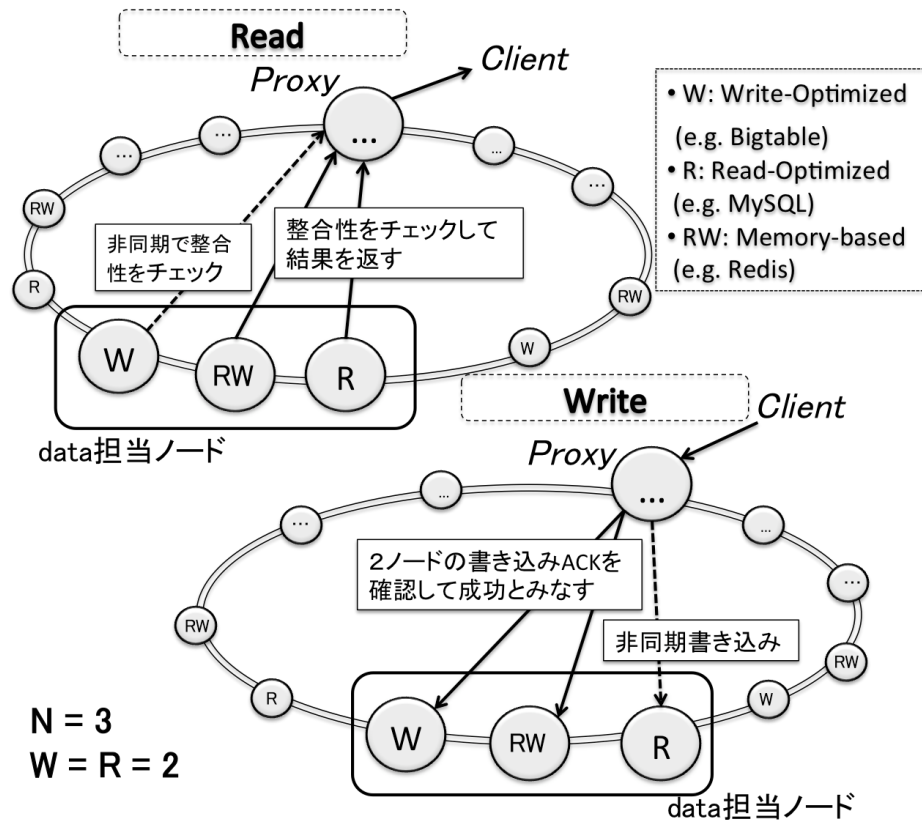


図 4 読み出しと書き込みの流れ (N=3 の場合)
Fig.4 Flow of read and write (in case of N = 3)

手順 1 が失敗しなければ、書き込み性能重視なノードへの読み出し遅延は全体の読み出し性能には影響を与えない為、遅延は読み出し性能重視なノードのものに抑えられる。整合性がとれない原因の多くは、読み出し性能重視のノードから得たデータのバージョンが古く、メモリベースのノードのデータと整合性が取れていないことに起因する。この場合でも、非同期に整合性の修正を行うため、次回以降のアクセスには書き込み性能重視のノードからの応答を待つことなく、結果を返すことが出来る。データ整合性を保ちつつも、読み出し遅延

を基本的には読み出し性能重視のノードの遅延、最悪でも元の Cassandra と同じ大きさの遅延で読み出せると期待できる。

このような操作を行うためには、プロキシがリクエストを適切なノードへ転送する必要がある。Cassandra はネットワーク近接性に依拠して順序付けを行い、リクエストを送信する順番を決めるが、MyCassandra Cluster では gossip プロトコルで取得した担当ノードのストレージエンジンで順序付けする。つまり、書き込み処理時には書き込み性能重視、メモリベース、読み出し性能重視の順で、読み出し処理時には読み出し性能重視、メモリベース、書き込み性能重視の順で、リクエストの送信先を順序付ける。

4. 性能評価

MyCassandra Cluster が読み出しと書き込みの性能が両立できることを性能評価する。評価には以下の MyCassandra のストレージエンジンを用いる。

- 書き込み性能重視: Bigtable (Cassandra 0.6.2)
- 読み出し性能重視: MySQL (6.0.10-alpha)
- メモリベース: Redis (2.0.4)

性能測定には Yahoo!’s Cloud Serving Benchmark (YCSB)¹⁰⁾ を用いる。

4.1 YCSB

YCSB は、様々なクラウドストレージを公平に評価することを目的として、Yahoo! Research が開発したオープンソースのベンチマークフレームワークである。実アプリに近いコアワークロードが用意されている。

YCSB では、読み出し処理と書き込み処理の回数の比率をユーザが指定できる。YCSB がクラウドストレージに対して読み出しと書き込みを実行し、ワークロード全体のスループットと、各処理に要した時間、つまり遅延を集計する。

YCSB が行う処理は次の 3 フェーズから成る。

- (1) load phase: 初期データをロード
- (2) warmup phase: ワークロードを実行してデータストアのキャッシュをウォームアップ
- (3) transaction phase: ワークロードを実行してスループットと遅延を計測・集計

表 1 に、今回の測定で用いる 4 種類のワークロードを示す。書き込み比率が高い Write-Only と Write-Heavy、読み出し比率が高い Read-Only、Read-Heavy を用意し、各ワークロードに対応する実アプリの例と、読み出しと書き込みの比率を表している。Write-Only、Read-Only とはそれぞれ読み出し、書き込みがバックグラウンドで行われ、性能には影響

表 1 YCSB ワークロード
Table 1 YCSB workloads

Workload	App. Ex.	Read	Update
Write-Only	<i>Log</i>	0%	100%
Write-Heavy	<i>Session</i>	50%	50%
Read-Heavy	<i>Photo Tagging</i>	95%	5%
Read-Only	<i>Cache</i>	100%	0%

を与えない。各ワークロードにおいて、アクセス対象データの分布として Zipf 分布を用いる。Zipf 分布とは、データ鮮度とは関係なく人気によってアクセス頻度が決まるようなアプリのデータアクセス分布を確率としてモデル化したものであり、一部のデータへのアクセス頻度が常に高く、大部分のデータへのアクセス頻度が低くなるような分布である。

表 2 に実験パラメータを、表 3 に実験環境を示す。複製数 $N=3$ として、前節の通り、MyCassandra Cluster は 1 台に 3 ノードを立ち上げるので $3 \times 6=18$ ノード、Cassandra は実マシン 6 台上にそれぞれ 1 ノード計 6 ノードとしている。

4.2 実験結果と考察

図 5、図 6 にクライアント数を調整して 1 秒あたり 5,000 回のクエリを発行した際の読み出しと書き込みの平均遅延をワークロードごとに示す。書き込み遅延は、MyCassandra Cluster はいずれのワークロードにおいても、元の Cassandra と同等に小さい。一方、読み出し遅延は、MyCassandra Cluster は Cassandra よりも小さく、Read-Heavy のワークロードでは Cassandra の 14.8%、Read-Only のワークロードでは 11.5%であるという結果が得られた。

図 7 にクライアント数を 40 として負荷をかけたときの全体のスループットをワークロードごとに示す。こちらの結果でも、MyCassandra Cluster は Write-Only のワークロードでは Cassandra と同等の高いスループットを達成している。一方、読み出しの比率の高い

表 2 実験パラメータ

Table 2 Experiment parameters

ノード	実マシン × 6 台
クライアント	実マシン × 1 台
ロードレコード件数	1,000 万件
quorum	$(N, W, R)=(3, 2, 2)$
key サイズ	最大で長さ 10 の文字列
value サイズ	100B のカラム × 10 計 1KB

表 3 実験環境

Table 3 Machine configurations

OS	Linux 2.6.35.6-48.fc14.x86_64
CPU	2.40 GHz Xeon E5620 × 2
Memory	32GB RAM
Disk	1TB SATA HDD × 2 (表 4 と同じ)
JVM	Java SE 6 Update 21

ワークロードでは、MyCassandra Cluster は Cassandra よりも高く、Read-Heavy のワークロードでは 1.49 倍、Read-Only のワークロードでは 6.53 倍のスループットを達成した。

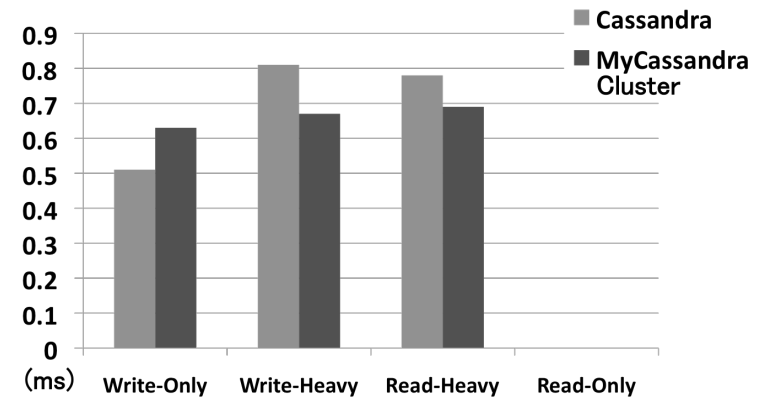


図 5 ワークロードごとの書き込み遅延
Fig. 5 Write latency for each workload

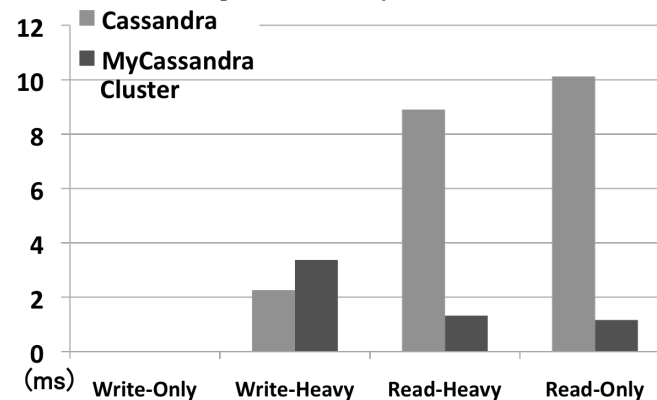


図 6 ワークロードごとの読み出し遅延
Fig. 6 Read latency for each workload

ところで、Write-Heavy (read:write=50:50) のワークロードでは、元の Cassandra の方が MyCassandra Cluster よりも高い性能を発揮した。Write-Heavy のワークロードでは、

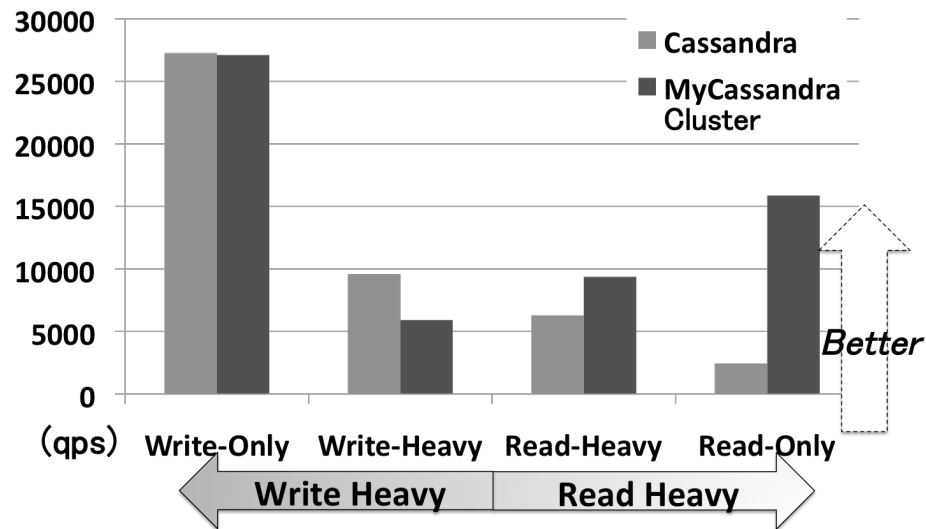


図 7 ワークロードごとのスループット
Fig. 7 Throughput for each workload

書き込み遅延は MyCassandra Cluster の方が小さいが、読み出し遅延は Cassandra の方が小さい。このことから MySQL において、非同期の大量の書き込み処理が、並行して行われる同期的な読み出し処理に影響を与え、結果として読み出し遅延の劣化となっていると考えられる。

この問題は、次の方法で改善できると考えている。現在、リクエストとストレージエンジンの種類に基づいてリクエストを振り分けているところ、1) ノードごとのアクセス負荷、2) 各ノードにおける参照の局所性を考慮した振り分けにより、この問題を緩和できる。つまり、読み出し性能重視のノードのアクセス負荷が大きく、データを書きこんですぐ読み出す場合は書き込み性能重視のノードから優先的に読み出した方が性能が良くなると考えられる。

4.3 SSD を用いた場合の性能

クラウドストレージではディスクへのランダム I/O が性能についてのボトルネックとなる。ここまでは HDD を用いて実験を行ってきた。一方で、クラウドストレージで用いるディスク装置として、SSD が急速に普及しているため、SSD を用いた場合の性能を確認した。

表 4.3 に HDD と SSD の諸元とディスク I/O ベンチマーク Bonnie++¹¹⁾ によるシーケン

シャルアクセス性能と 1 秒あたりのランダムシーク回数を示す。各マシンにこの SSD を 2 台搭載した。Cassandra では CommitLog、SSTable を別のディスクに配置し、MyCassandra Cluster ではランダム I/O が頻繁に発生する Bigtable の SSTable と MySQL を別のディスクに配置した。

YCSB の 4 つのワークロードのスループットを図 8、図 9 に示す。結果としては、Cassandra では、Read-Only のワークロードにおいて SSD が HDD の 3.37 倍のスループットを得られた。MyCassandra Cluster では、Read-Heavy 時に SSD が HDD の 1.50 倍のスループットが得られた。読み出し比率が高いワークロードでの性能は、SSD による向上は大きく、特に Cassandra での性能向上が大きい。一方、書き込み比率が高いワークロードでの性能は、HDD と SSD でほぼ同じとなった。これは Bigtable の書き込みはシーク性能が低いディスク装置でも高い性能を発揮できるように設計されているためであると考えられる。

5. 議 論

本節では MyCassandra Cluster の設計に関する得失について議論する。

5.1 メモリベースのストレージエンジンの使用

ノード停止時においても、Cassandra の機能である read repair と hinted handoff によって複製数は保たれる。

また、メモリベースのノード上のデータがメモリから溢れた場合、解決手段としては 1) ストレージエンジンに備わる永続化機能を用いるか、2) read repair によって他の永続型のストレージエンジンのノードから失ったデータを新たに取得する方法がある、ただし、いずれの場合も性能のデメリットはある。

表 4 HDD と SSD の環境
Table 4 Disk configurations

	HDD	SSD
Vender	Western Digital	Crucial
Model	WDC WD1002FBYS	CTFDDAC128MAG
Capacity	1TB	128GB
Seq. Write	75915 KB/s	85279 KB/s
Seq. Read	139780 KB/s	283133 KB/s
Seek	200.6 /sec	8502.0 /sec

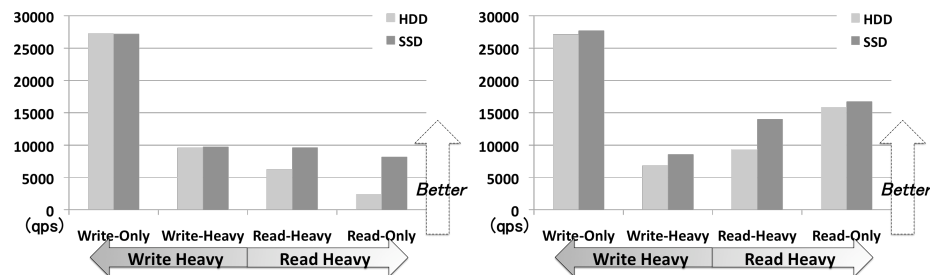


図 8 HDD/SSD 上での Cassandra のスループット
Fig.8 Throughput for Cassandra on HDD vs. SSD

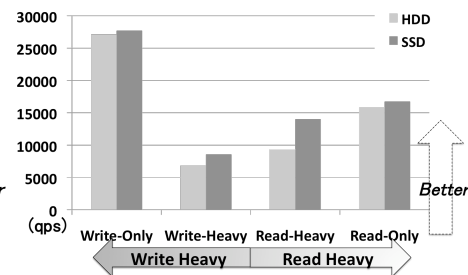


図 9 HDD/SSD 上での MyCassandra Cluster のスループット
Fig.9 Throughput for MyCassandra Cluster on HDD vs. SSD

5.2 ディスク/メモリの使用率

本手法は、ストレージエンジンの I/O 性質に基づいてホストあたりに複数台のディスクを搭載する必要がある。また、メモリに関してはホスト 1 台のリソースを 3 ノードで分割して用いている。これは、負荷の分散が均等である場合に、実験結果に表れている通り、元の Cassandra と同等、もしくはそれ以上の性能が得られている。一方、負荷が不均等の場合に関する調査は今後の課題となる。フラグメント化によるデメリットと仮想ノードによる負荷分散に関わる恩恵がどのように性能に関わってくるか、評価・考察したい。

6. 関連研究

モジュラーなクラウドストレージと読み出し/書き込み性能の両立に関する関連研究を挙げる。

Anvil¹²⁾ は、粒度の細かいコンポーネント dTable を組み合わせて構成されるデータストアである。アプリケーションのデータアクセスパターンに応じたデータストアを構成できる。データストアをモジュラーに構成しようという点が本研究と共通しているが、分散データストアを対象としているわけではない。

Cloudy¹³⁾ は、クラウドストレージをモジュラーに構成しようという提案である。ストレージエンジン以外にも、ルーティング処理やロードバランスの方式もコンポーネント化し、選択可能とすることを提案している。性能は報告されていない。

Amazon Dynamo¹⁴⁾ は、Amazon 社がウェブ上で提供するサービス向けに開発した key-

value store である。ストレージエンジンとして Berkeley DB や MySQL、メモリ上のバッファを用いることができる。ストレージエンジンを選択する際の指標の例として、格納データのサイズが挙げられている。Dynamo で用いることのできるストレージエンジンでは、読み出し性能と書き込み性能を調整することはできない。

MyCassandra⁶⁾ は読み出し性能重視と書き込み性能重視の選択が可能であり、分散環境での定量的な評価を行っている。また、本研究ではこの MyCassandra をベースとして読み出し性能重視、書き込み性能重視を選択するだけでなく、その両方の性能を両立している。

書き込み性能と読み出し性能の両立する索引アルゴリズムの研究はなされている。FD-Tree¹⁵⁾ は、SSD 上における効率的な索引アルゴリズムであり、B+tree 並の検索速度と LSM-tree 並の更新速度という成果が得られている。対象が SSD に限られている。また、クラウドストレージへの適用が期待される。

7. まとめと今後

本研究では、クラスター内で複数種類のストレージエンジンを組み合わせることで、読み出し/書き込み性能を両立できるようなクラウドストレージの提案と実装を行った。元の Cassandra と同等の高い書き込みスループットを達成しつつ、読み出し性能でも高い性能を達成した。以下に、今後の課題を挙げる。

Amazon EC2 などの大規模なクラスターを用いて本手法のスケラビリティの検証を行う。ただし、Cassandra の構造ではプロキシ/データノード 1 台のアクセスパスは変わらないため、ノード数の影響は受けないと考えられる。

ラックやデータセンターとのネットワーク近接性を考慮しつつも、ノードごとに読み出し/書き込み性能のどちらに優れるのか考慮して、複製を配置する。

アプリケーションの読み書きの比率に対して、最高のスループットが得られるようなストレージエンジンの構成比率を検討する。

クラウドストレージの性能は読み書きの比率によってのみ決まるものではない。書き込まれたデータがどれくらいの頻度や間隔でアプリケーションから読み出されるかに依存する面もある。例えば、MyCassandra Cluster において書き込んですぐ読み出す必要がある場合はデータの同期的な読み出しを結果がまだ反映されていない読み出し性能重視なノードからではなく、書き込み性能重視なノードから行った方が遅延を小さく抑えられると考えられる。このようにデータアクセスの特性を考慮する。

謝辞 本研究は科研費(22680005)の助成を受けたものである。

参 考 文 献

- 1) The Apache Software Foundation: Cassandra, <http://cassandra.apache.org/> (2010).
- 2) Lakshman, A. and Malik, P.: Cassandra - A Decentralized Structured Storage System, *Proc. LADIS '09* (2009).
- 3) The Apache Software Foundation: HBase, <http://hadoop.apache.org/hbase/> (2010).
- 4) Cooper, B.F., Ramakrishnan, R., Srivastava, U., Silberstein, A., Bohannon, P., Jacobsen, H., Puz, N., Weaver, D. and Yerneni, R.: PNUTS: Yahoo!'s Hosted Data Serving Platform, *Proc. VLDB '08* (2008).
- 5) Oracle: MySQL, <http://www.mysql.com/>.
- 6) 中村俊介, 首藤一幸: 読み出し性能と書き込み性能を選択可能なクラウドストレージ, 情報処理学会研究報告, 2011-OS-116(-10) (2011).
- 7) O'Neil, P., Cheng, E., Gawlick, D. and O'Neil, E.: The Log-Structured Merge-Tree (LSM-Tree), *Acta Informatica*, Vol.33, No.2, pp.351-385 (1996).
- 8) Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A. and Gruber, R.E.: Bigtable: A Distributed Storage System for Structured Data, *Proc. OSDI '06*, Vol.7, pp.205-218 (2006).
- 9) Redis: Redis, <http://redis.io/> (2010).
- 10) Cooper, B.F., Silberstein, A., Tam, E., Ramakrishnan, R. and Sears, R.: Benchmarking Cloud Serving Systems with YCSB, *Proc. SOCC '10* (2010).
- 11) Coker, R.: Bonnie++, <http://www.coker.com.au/bonnie++/>.
- 12) Mammarella, M., Hovsepian, S. and Kohler, E.: Modular Data Storage with Anvil, *Proc. SOSP '09* (2009).
- 13) Kossmann, D., Kraska, T., Loesing, S., Merkli, S., Mittal, R. and Pfaffhauser, F.: Cloudy: A Modular Cloud Storage System, *Proc. VLDB '10* (2010).
- 14) DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Voshall, P. and Vogels, W.: Dynamo: Amazon's Highly Available Key-value Store, *Proc. SOSP '07* (2007).
- 15) Li, Y., He, B., Yang, R.J., Qiong, L. and Ke, Y.: Tree Indexing on Solid State Drives, *VLDB '10* (2010).