D-016

I/O スケジュールの制御による KVS ノード追加時間の短縮に関する一考察
A Study on Reduction of Node Joining Time by Controlling I/O Scheduling

森光 由輝† 水野 修† 山口 実靖†
Morimitsu Yuki† Mizuno Osamu† Saneyasu Yamaguchi†

1. はじめに

大規模 SNS にかかる負荷は社会の動きに合わせて変動している．また，大規模 SNS の普及などにより，データベース管理システムのスケーラビリティの確保や動的スケールが重要視されるようになり，分散型 DB の KVS（Key−Value Store）が注目されるようになった．KVS はシンプルなデータベース構造であるため高いスケーラビリティを持っている．Cassandra などの KVS システムは，稼働中に新規ノードの追加や既存ノードの削除が可能であり，高負荷時はノード数を増やしデータベース管理システムの性能を向上させ，負荷が低くなればノード数を減らし省電力化させることができる．このためには，短い時間でのノードの追加や削除が重要であると考えられる．ノード追加時間やノード削除時間の削減方法として Actcache と NLL が提案されている[1].

本稿では，KVS の一つである Cassandra における I/O スケジューラと動的ノード追加時間の関係について調査し，さらなるノード追加時間の短縮について考察する．

2. KVS，Cassandra

KVS とは，Key を指定して Value を得る仕組みのデータベース管理ソフトウェアである．RDBMS などより機能が単純になっているが，高いスケーラビリティを得ることができる． KVS の代表的な実装の一つに Cassandra がある．Cassandra は Facebook 社が開発した KVS であり，現在は Apache Software Foundation のトップレベルプロジェクトである．

Cassandra のノード配置には，仮想ノードを用いる手法と，用いない手法の二通りの方法が存在する．本稿では，前者を仮想ノード配置と，後者を非仮想ノード配置と呼ぶ．まず，非仮想ノード配置について述べる．Cassandra を構成する各ノードはトークンと呼ばれるハッシュ値を持ち，図 1 の様にトークンをもとにリング状のハッシュ空間に配置される．リング上の各ノードは，ハッシュ値が自身のトークン値以下でかつ直前ノードのトークン値より大きい範囲を担当する．KVS の書き込みまたは読み込みをする際には Key をハッシュ関数にかけ，そのハッシュ値を担当するノードが読み込みや書き込みを実行するノードとなる．ただし，後述のレプリカが存在する場合，それを持つノードも実行するノードの対象となる．

Cassandra ではデータベースの複製（レプリカ）の数を指定することが可能である．レプリカ数は初期設定では 1 であるが，2 以上の値にすることによって耐障害性を向上させることができる．レプリカは上記の担当ノードの後続ノードに配置される．

稼働中のシステムに新規ノードを追加する場合，新規ノードのトークン値から新規ノードの担当範囲が決まり，その範囲を担当している稼働ノードから担当範囲分のデータを取得する．稼働中のシステムからノードを動的に削除する場合，削除されるノードは担当している範囲のデータを新しく担当するノードに転送する．

次に仮想ノード配置について述べる．仮想ノード配置では，物理ノードが複数（初期設定では 256 個）の仮想ノードを持ち，各物理ノードは自身が持つ仮想ノードの担当範囲の合計を担当する．仮想ノードのトークン値は，図 2 の様にランダムに決められる．仮想ノードを用いることにより，物理ノードが数台しか存在しない環境であっても，ハッシュ空間上では仮想的に数百台のノード（仮想ノード）が存在していると認識させることが可能である．非仮想ノード配置でも定常時のデータの分散を均等にする（あるいは偏りを持たせる）ことは，トークン範囲を均等にする（あるいは偏りを持たせる）ことにより実現できるが，あるノードの担当範囲は連続している 1 つの領域であるためノード追加，削除時のデータの追加，削除の処理は特定のノードに対して集中的に行われることとなる．これに対して，仮想ノード配置では，ノード追加，削除時のデータの追加削除の処理も分散して行われ，均等に（あるいは偏りを持たせて）行うことができる．

Cassandra には Autocompaction というファイル結合処理が存在する．Autocompaction 処理は 4 つの同サイズのデータファイルが作成されたとき，新しい 1 つのファイルを作成し，既存の 4 つのファイルを統合する処理である．Compaction 処理によって読み込み性能の向上やディスクスペース利用の最適化が実現される．新規のノードを追加したときなどは，そのノードが大量のデータを受信してそれらデータをファイルに格納し，Autocompaction が発生することが多い．

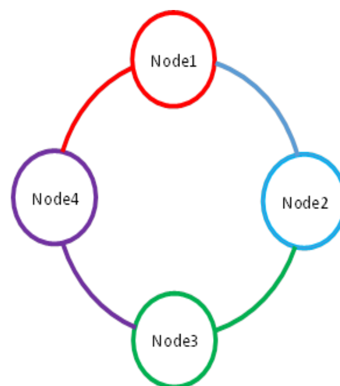

図 1 非仮想ノード配置

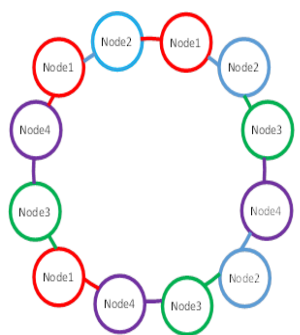

図 2 仮想ノード配置

3. ActCache

OS はページキャッシュ内のデータを LRU に基づき管理し，通常は頻繁にアクセスされるファイルをキャッシュ内に格納し続ける．しかし，Cassandra におけるデータベース運用中のノード追加処理などの様に大量の I/O が発生する

† 工学院大学大学院 工学研究科 電気・電子工学専攻

Copyright © 2017 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.

様な状況では，頻繁にアクセスされるデータが頻繁にキャッシュから破棄されると考えられる[1]．具体的には，“data file”への大量のアクセスが“index file”をキャッシュから破棄させていると考えられる．よって“index file”をページキャッシュ内に固定的に格納することによりノード追加処理時間の短縮が可能であると考える．文献[1]にて“index file”をページキャッシュ内に固定的に格納し続ける手法が提案され，これは“Act Cache”と呼ばれる．

4. LowLatency オプションの無効化

ノード追加処理実行中に新規ノードは多くのデータを受け取り，データ量が増えることによって Autocompaction が実行される．この処理はディスクに大きな負荷を与える．Linux OS に実装されているスケジューラである CFQ には LowLatency というオプションがあり，本オプションを有効にすると，各プロセスのスライスタイム(I/O 処理を行える時間)はシステムに設定されている target_latency(初期設定 300[ms])の値に基づいて決定されるようになる．この有効化により，各プロセスのスループットよりも公平性を重要視する様にできる．LowLatency オプションを無効化すると，プロセスは与えられたタイムスライスの全てを使用できるようになり，大規模 I/O などの性能は向上する．初期設定では LowLatency は有効化されている．文献[1]にて LowLatency オプションを無効化することにより I/O 使用率を低下させノード追加処理時間を短縮する手法が提案されており，これは“NoLowLatency(NLL)”と呼ばれる．

また，ActCache と NLL を併用した手法は“AC+NLL”と呼ばれる．

5. I/O スケジューラとノード追加時間

本章では I/O スケジューラとノード追加時間の関係について考察する．I/O スケジューラの一つである CFQ の ExpireTime を初期値（1 倍），2 倍，3 倍とし，ActCache を用いた状態でのノード追加時間および AutoCompaction 時間を測定した．ExpireTime は各プロセスに与えられるタイムスライスで，大きいほど各プロセスは連続して I/O 処理を行うことが可能となる．大規模 I/O 処理などは高いシーケンシャル性で連続的に処理することが可能となり，さらなる性能向上が期待できる．

既存ノード 3 台，新規ノード 1 台で Cassandra システムを構築し，AutoCompaction 無効化状態におけるノード追加処理と，ノード追加後における AutoCompaction 処理を行った．データベースの作成かつ読み込み負荷には YCSB（Yahoo Cloud Serving Benchmark）を使用し，測定はレコード数 1600 万件(約 17[GB])のデータベースで行った．ノード追加処理中は，Cassandra システムに 120 スレッドによる 120[ops/sec]の読み込み要求を行った．使用した計算機の仕様は表 1 の通りである．ノード追加時間を図 3 に示す．LL は LowLatency オプションを有効にした場合を表す．図 3 より，ExpireTime を 2 倍にすることにより，さらなるノード追加時間の短縮ができていることがわかる．しかし，ExpireTime を 3 倍にするとノード追加時間は増加しており，適切なチューニングが必要であることが分かる．

図 4 に ExpireTime 変更時の Autocompaction 時間を示す．図 4 の結果から，NLL の AutoCompaction 時間は ExpireTime を増加させるとわずかに増加するが，2 倍であれば増加は極めて小さいことがわかる．

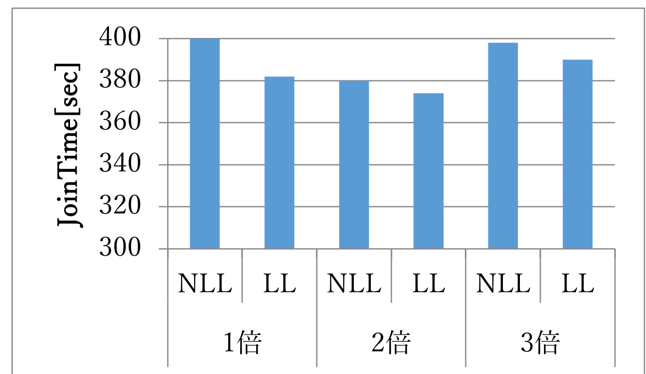

図 3 ExpireTime 変更時のノード追加時間

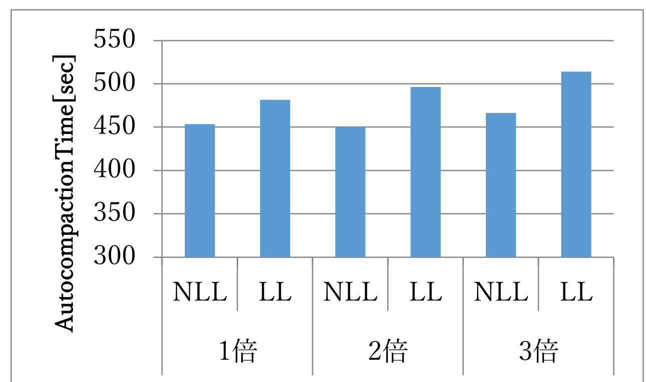

図 4 ExpireTime 変更時の AutocompactionhTime

表 1 測定環境

	Node1～4	クライアントPC
OS	CentOS 6.6	CentOS 5.5
CPU	AMD Turion II Neo N54L Dual-Core Processor 2196MHz	Intel Core i3 530 2.93GHz
Memory	4	2
HDD	ST2000DM001-1CH164	VB0160EAVEQ

6. おわりに

本稿では ActCache 手法と NLL 手法に着目し，ExpireTime を増加させることによるノード追加時間のさらなる削減について考察した．

今後は Autocompaction 時間の短縮に向けて調査を行っていく予定である．

謝辞

本研究は JSPS 科研費 26730040, 15H02696, 17K00109 の助成を受けたものである．

本研究は、JST、CREST JPMJCR1503 の支援を受けたものである．

参考文献

[1] Saneyasu YAMAGUCHI and Yuki MORIMITSU, "Improving Dynamic Scaling Performance of Cassandra," IEICE Transactions on Information and Systems, Vol. E100.D (2017) No. 4 pp. 682-692, DOI: http://doi.org/10.1587/transinf.2016DAP0009

Copyright © 2017 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.