D-038

I/O 最適化による KVS 動的ノード追加性能の向上に関する一考察
A Study on Improvement of KVS Dynamic Node Addition Performance by I/O Optimization.

御代川 翔平† 徳田 大輝† 山口 実靖†
Shohei Miyokawa Taiki Tokuda Saneyasu Yamaguchi

1. はじめに

近年，大規模 SNS が普及してきたことにより，データベース(DB)管理システムのスケーラビリティの確保が重要視されるようになった．そこで注目されたのが分散型 DB の KVS(Key-Value Store)である．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ち，またサーバ(ノード)稼働中に新規ノードの追加や既存ノードの削除が可能といった動的性能伸縮性を持っているため，高負荷環境で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といったことができる．

本稿では，動的性能伸張性について着目し，KVS の一つである Cassandra を用いて動的ノード追加性能について調査し，I/O 最適化によるノード追加性能向上についての考察を行う．

2. KVS(Key-Value Store)

KVS(Key-Value Store)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．機能が既存のデータベース(RDBMS など)より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 の一つに Cassandra[1]がある．

2.1 Cassandra

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．Amazon 社の Dynamo[2]の分散ハッシュテーブルと Google 社の BigTable[3]のデータモデルを併せ持ち，結果整合性の一貫性を持つ．

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，リング状のハッシュ空間にトークンをもとに配置される．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．読み込みまたは書き込みをする際には Key をハッシュ関数にかけ，そのハッシュ値から担当ノードを特定し読み込み，書き込みを実行する．また，稼働中のシステムに新規ノードを追加した場合，新規ノードはトークン値から担当範囲が決まり，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．

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，複数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．レプリカは担当ノードの後続ノードに配置される．

† 工学院大学大学院 工学研究科 電気・電子工学専攻
Electrical Engineering and Electronics, Kogakuin University Graduate School

3. ノード追加時間の評価

Cassandra は，ノードを追加することによって性能を拡張することができる．本章では，Cassandra システムにおける性能拡張処理に要する時間(join 命令を発行した時刻から，状態が Joining から Normal に変わる時刻までの時間)の評価を行う．

評価環境は既存ノードの PC3 台，クライアント PC 1 台，新規追加ノード用の PC1 台で構成される．データベースの作成にはベンチマークソフトの YCSB(Yahoo Cloud Serving Benchmark)[4]を使用した．測定はレコード数 1600 万件(約 17[GB])のデータベース，レプリカ数 3 で行った．

3.1 ノード追加のボトルネック

ノード追加を行ったときの既存ノードと新規ノードの CPU・I/O 使用率を図 1 に示す．Node1~3 は既存ノード，Node4 は新規ノードを示している．図 1 から新規ノードの disk I/O がノード追加のボトルネックになっていることが分かる．

3.2 遅延書き込み時間延長によるノード追加時間

新規ノードの disk I/O がボトルネックであることから，新規ノードの遅延書き込み時間を 30 秒から 3000 秒に拡大した．

遅延書き込み時間延長によるノード追加時間の測定結果を図 2 に，新規ノードの disk I/O を図 3，4 に示す．

図 2 から遅延書き込み時間を延長した際にノード追加時間が約 8%の短縮が起こっていることが分かる．図 3，4 から通常のノード追加では HDD address の 2[GB]と 220[GB]周辺に書き込み処理をしていることがわかり，図 4 である遅延書き込み時間延長時の disk I/O では，2[GB]と 220[GB]周辺での書き込み処理をしておらず，またそれによる長距離シークがなくなったため，ノード追加時間が短縮したと考えられる．

3.3 Cassandra における自動同期制御

新規ノードの物理メモリ量が 2，8[GB]のときのノード追加時間の測定結果を図 5 に示す．図 5 からノード追加時間は新規ノードの物理メモリ量と関係していないことが分かる．これは Cassandra が新規ノード追加実行中に，新規ノードの HDD と物理メモリを複数回にわたって同期をしているためである．以後，これを自動同期制御と呼ぶ．本節では Cassandra にある自動的に同期を行う制御を取り除き，遅延書き込み時間延長時のノード追加時間を測定した．結果を図 6 に示す．通常の Cassandra の測定結果を Cassandra，Cassandra から自動同期制御を取り除いたものを Cassandra-sync とした．また last_sync が付加されている結果は新規ノード追加完了後に sync コマンドを実行し sync コマンドが終了するまでの時間をノード追加時間としたものであることを意味している．

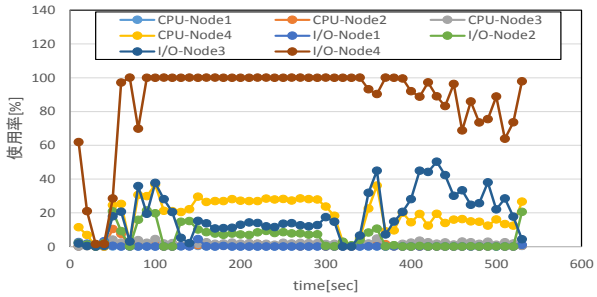

図 1　ノード追加時の CPU・I/O 使用率

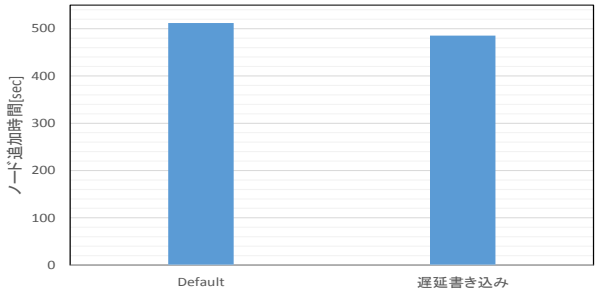

図 2　遅延書き込み時間延長による追加時間

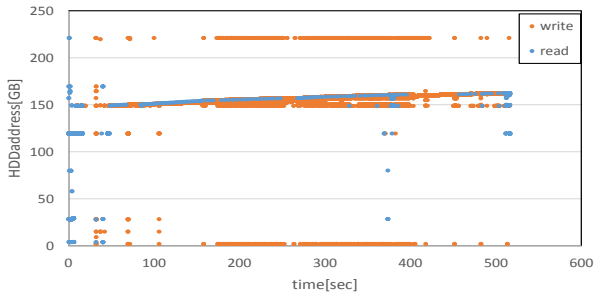

図 3　通常ノード追加時の disk I/O

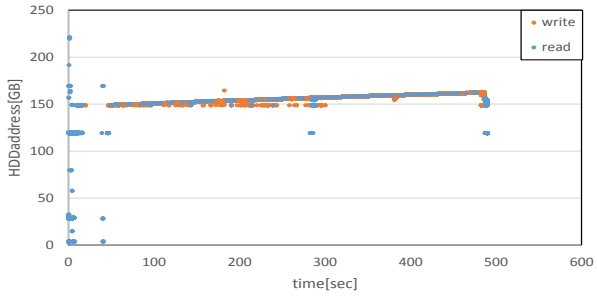

図 4　遅延書き込み時間延長時の disk I/O

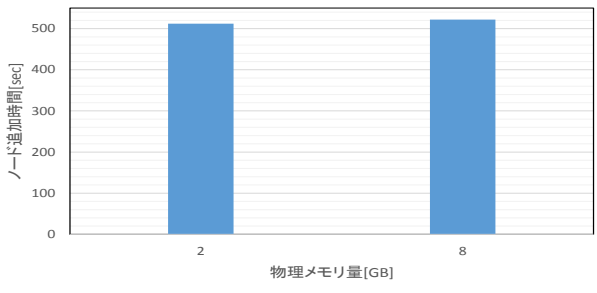

図 5　物理メモリ量(2，8[GB])による追加時間

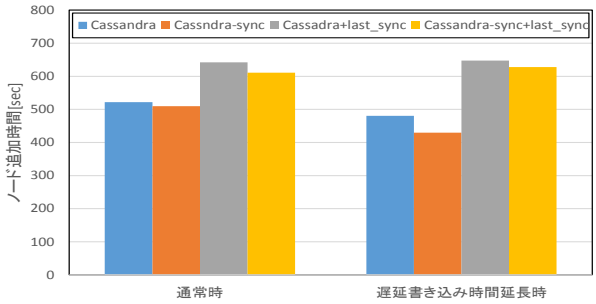

図 6　非自動同期制御による追加時間

図 6 より Cassandra から自動同期制御を除き遅延書き込み時間を延長させることによってノード追加時間は約 20%の短縮が行われていることが分かる．Sync コマンドを行った場合では，自動同期制御を除いた Cassandra で行った方が，約 4%の短縮があることが分かる．

3.4 考察

自動同期制御を除いた際のノード追加時間について考察する．自動同期制御を除いたことにより，ノード追加時間の短縮が行われた．これは HDD における長距離シークの削減とノード追加完了時に取得しているデータベースの一部データが dirty page として物理メモリのみに保存されているためだと考えられる．よって，Cassandra システムの性能拡張されるまでにかかる時間は短縮されるが，データの保存性は低下していると考えられる．

4. おわりに

本研究では，Cassandra のノード追加時間に着目し，評価と考察を行った．評価より，遅延書き込み時間の延長によってノード追加時間の改善がわずかながら可能であることがわかった．また，Cassandra にある自動同期制御を取り除くことによって遅延書き込み時間延長によるノード追加時間の更なる短縮が可能であることがわかった．

今後は，高負荷環境でのノード追加性能について調査する予定である．

謝辞

本研究は JSPS 科研費 24300034，25280022，26730040 の助成を受けたものである．

参考文献

[1] Avinash Lakshman and Prashant Malik, “Cassandra- A Decentralized Structured Storage System”, LADIS 09, (2009).
[2] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels, “Dynamo: Amazon’s Highly Available Key-value Store”, SOSP’07, (2007).
[3] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes and Robert E. Gruber,“Bigtable: A Distributed Storage System for Structured Data”, IOSDI’06, (2006).
[4] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, Russell Sears, “Benchmarking Cloud Serving Systems with YCSB” ACM symposium on Cloud computing, (2010).
[5] 堀内 浩基,山口 実靖,“KVS における動的性能伸張性の向上”研究報告マルチメディア通信と分散処理（DPS）,Vol.2013-DPS-154(39),No.10 (2013)