D-039

ディスク I/O を考慮した KVS の動的ノード追加時間の短縮手法の一考察
Disk I/O Aware Dynamic Node Joining Performance Improvement in KVS

御代川 翔平† 徳田 大輝† 藤島 永太† 山口 実靖†
Shohei Miyokawa Taiki Tokuda Eita Fujishima Saneyasu Yamaguchi

1. はじめに

大規模 SNS やクラウドサービスの普及により情報処理システムが処理するデータ量は膨大となり，データベース管理システム(DBMS)のスケーラビリティが重要視されるようになった．そこで高いスケーラビリティを持つ分散型データベースである KVS に注目が集まっている．代表的な KVS の一つである Cassandra[1]は動的なノードの追加や削除による動的性能最適化が可能であり，負荷量の時間変化に合わせてノード数を増減させることができる．ノード数の動的最適化を効率的に行うには，ノード追加処理にかかる時間の短縮が重要であると考えられる．

本稿では動的ノード追加処理に着目し，ディスク I/O を考慮したノード追加処理時間の短縮方法の提案，そしてその評価と考察を行う．

2. KVS

KVS(Key-Value Store)は，Key を指定して Value を取得する仕組みの DBMS である．機能が既存のデータベース管理システム(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 がある．

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，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．よって既存ノードには多くの読み込み負荷が，新規ノードには多くの書き込み負荷が発生する．

3. ノード追加処理の評価

Cassandra は，システム稼働中に動的に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 システムにおけるノード追加処理に要する時間(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，負荷の考察を行う．

評価環境は既存ノードの PC3 台，新規追加ノードの PC1 台，クライアント PC1 台で構成される．データベースの作成と読み込み負荷にはベンチマークソフトの YCSB(Yahoo Cloud Serving Benchmark)[2]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 で行い，読み込み負荷は 120 [ops/sec]で行った．

3.1 ノード追加処理のボトルネック

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(以下，この実験を“Normal”と呼ぶ)．またこの時のノード追加処理時間を図 2 の“Normal”に示す．図 1 の Node1~3 は既存ノード，Node4 は新規ノードを示している．図 1 からノード追加処理中(During Join)は既存ノードの disk I/O 使用率が最も高いことが分かる．このことからノード追加処理のボトルネックは既存ノードの disk I/O であると考えられる．

3.2 ファイルアクセス回数

既存ノードの disk I/O がボトルネックであることから，既存ノードのファイルアクセス回数について調査した．

既存ノードの，ファイルサイズと 1 秒間当たりのアクセス回数の調査結果を図 3 に示す．アクセス回数は OS の SCSI サブシステム実装を改変して観察し，各ファイルのアドレスに対して発行された HDD アクセス命令の数を表している．すなわち，OS ページキャッシュがミスし実際に HDD アクセスが生じた回数を表している．

図 3 からファイルサイズは“data file”が最も大きく，“index file”の約 34 倍の大きさであることが分かる．ファイルアクセス回数は，ノード追加処理前は“index file”と“data file”のアクセス回数がほぼ等しいことが分かる．またノード追加処理中・後は“data file”が“index file”の約 2 倍のアクセス回数があることが分かる．図 3 からボトルネックは“index file”と“data file”へのアクセスが強く関係しており，1 バイトあたりのアクセス回数の側面で見ると“index file”が特に強い影響を与えていると考えられる．

4. 提案手法

3 章 2 節の調査より，ノード追加処理のボトルネック処理は“index file”と“data file”へのアクセスであると考えられる．また“index file”は“data file”に比べ，非常に小さいが頻繁にアクセスされており，かつ頻繁にキャッシュ

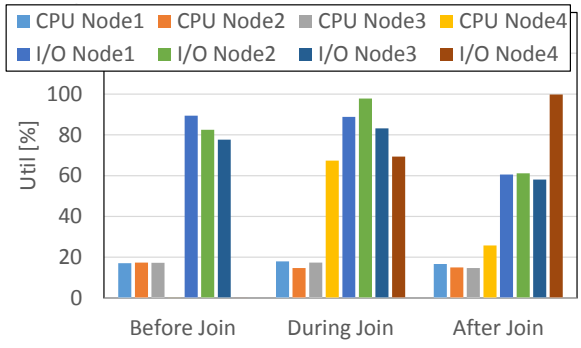

図 1 通常ノード追加時の CPU・I/O 使用率

† 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

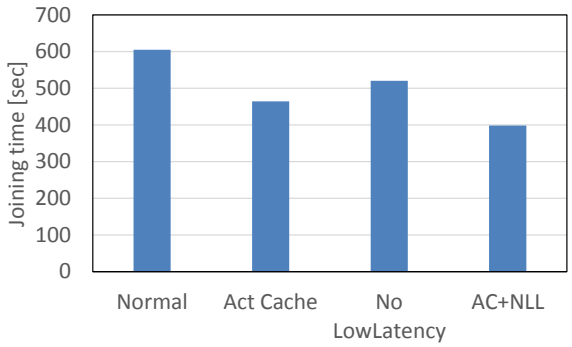

図 2　ノード追加処理時間の評価

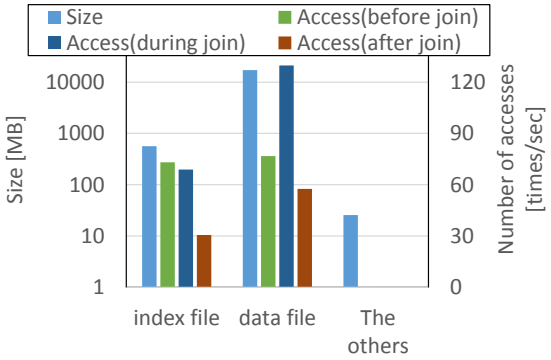

図 3　ファイルサイズとファイルアクセス回数

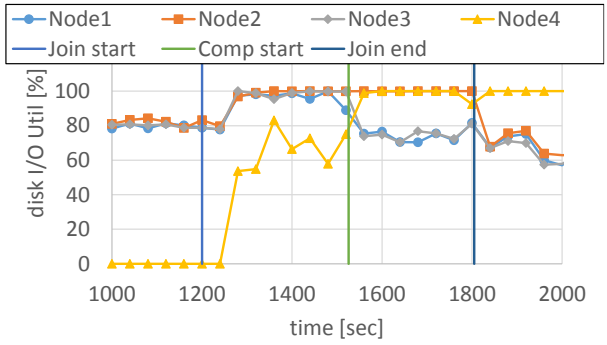

図 4　I/O 使用率(推移)

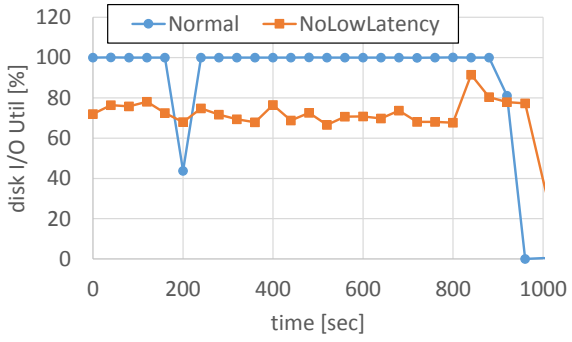

図 5　Autocompaction 処理の I/O 使用率(推移)

ミスしており多くの HDD アクセスを発生させていることが分かる．OS は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．しかし本観察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ており，“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られる．本稿では，“index file”をキャッシュ内に固定的に格納し続ける手法を提案し，これを“Act Cache”と呼ぶ．

また，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． Autocompaction は 4 つの同サイズのデータファイルが作成されたとき，4 つのファイルを 1 つのファイルに新しく作成しなおす処理である．図 4 は join 処理中の各ノードの I/O 使用率の推移を表しており，図の Join start はノード追加処理開始時刻を示し，Join end はノード追加処理完了時刻を示している．また Comp start は Autocompaction が開始した時刻を示す．図 4 から Autocompaction が実行されてから Node4 の disk I/O 使用率が 100%になることが分かる．このことから Autocompaction は disk I/O への負荷が大きいと考えられる．Autocompaction 処理のみの disk I/O 使用率を図 5 に示す．図 5 の NoLowLatency は CFQ スケジューラの LowLatency オプションの無効化した計測を示している．図 5 より LowLatency オプションを無効化することによって disk I/O 使用率が低下していることが分かる．LowLatency を無効化することによってノード追加処理時間を短縮する手法を提案し，これを“NoLowLatency”と呼ぶ．

また上記の両手法を併用した手法を“AC+NLL”と呼ぶ．

5. 性能評価

提案手法を用いたノード追加の処理時間を図 2 の“ActCache”，“NoLowLatency”，“AC+NLL”に示す．それぞれの手法を“Normal”と比較し，“Act Cache”は約 23%，“NoLowLatency”は約 14%，“AC+NLL”は約 36%の短縮を実現していることが分かる．

6. おわりに

本研究では，Cassandra のノード追加処理時間に着目し，評価と短縮手法の提案を行った．評価より，読み込み負荷環境におけるノード追加処理のボトルネックは既存ノードの disk I/O であることが分かった．そして 3 つの提案手法によってノード追加処理時間の短縮が可能であることが分かった．

今後は，さらなるノード追加処理時間の短縮に向けて調査する予定である．

謝辞

本研究は JSPS 科研費 25280022, 26730040, 15H02696 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] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, Russell Sears, “Benchmarking Cloud Serving Systems with YCSB” ACM symposium on Cloud computing, (2010).
[3] 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)