キーバリュー分離下で範囲クエリを効率的に行うデータ格納手法の提案

杉浦直人[1]　藤木大地[1]

概要：キーバリューストアはデータの格納にログ構造化マージツリー（LSM ツリー）を使用しているが、その高い書き込み増幅が問題となっている。LSM ツリーの書き込み増幅を減らす方法として、キーバリュー分離という手法がある。キーバリュー分離では、キーのみを LSM ツリーに格納し、サイズの大きいバリューを別のログに保存する。このログは書き込み順にデータを格納するため、範囲クエリを行う際にランダムアクセスが発生する。そこで本研究では、このログ上のバリューを範囲ごとに分割して配置することで、範囲クエリを高速化する手法を提案する。

キーワード：データベース、キーバリューストア、ストレージシステム、SSD

1. はじめに

キーバリューストアでは、ログ構造化マージツリー[1]（LSM ツリー）を使用し、書き込みを最適化している。LSM ツリーは、ランダムな書き込みをメモリ上にバッファし、大きなシーケンシャルライトを行うことで、高速な書き込みを実現する。その後、効率的な検索を可能するために、古いキーバリューペアを削除し、キー順にソートするコンパクションという処理を行う。大量のデータが書き込まれると、コンパクションが頻繁にトリガーされ、同じデータが何度も書き直しされることになる。このような書き込みの増幅は最大で 50 倍になると報告されていて[2]、問題になっている。

この書き込み増幅を削減する手法として、WiscKey[2]のキーバリュー分離がある。WiscKey では、バリューを LSM ツリーではなく、別の vLog と呼ばれる循環ログに格納し、LSM ツリーにキーとバリューのアドレスを格納する。読み取り時には、LSM ツリーからバリューのアドレスを取得し、vLog からバリューを取得する。バリューのアドレスはバリューに比べてサイズが小さいことが多いため、コンパクションによる書き込み増幅が削減される。また、LSM ツリーのサイズが小さくなりメモリにキャッシュされることで、読み取り性能も向上する。

しかし、WiscKey の vLog は一つの大きなログ構造に全てのバリューを配置するので、古いバリューを削除するガーベジコレクション（GC）のオーバーヘッドが大きいという問題がある。GC では、vLog の末尾から順に LSM ツリーへクエリを行い、バリューが更新されていないか確認を行う。更新されていないバリューのみを vLog の先頭に書き込み、使用していたスペースを開放する。GC の問題点として、LSM ツリーへのクエリは時間がかかるため、更新中心のワークロードでは性能が低下してしまう。

この問題に対処するために、HashKV[3]は、vLog をハッシュベースで分割することで、効率的なガーベジコレクションを実現している。vLog をハッシュベースで segment に分割し、キーのグルーピングを行う。同じハッシュ値を持つものは必ず同じ segment に格納される。segment ごとに GC を行うことで、LSM ツリーへクエリを行わずに、どのバリューが最新かどうか確認できる。GC から時間のかかる LSM ツリーへのクエリを排除することで、HashKV は効率的な更新を実現した。

これらのキーバリュー分離は、効率的な書き込みや更新を実現しているが、vLog 上のバリューがソートされていないため、範囲クエリ時にランダムアクセスが発生し、範囲クエリのパフォーマンスはよくない。ビッグデータの時代において、効率的な範囲クエリは、ますます重要になっている。分析を行うためには、キーバリューストアに保存したデータを範囲クエリで集計する必要がある。大量のデータを扱うキーバリューストアにおいて、範囲クエリの高速化は重要な課題である。

本研究では、範囲クエリの高速化の方法として、HashKV のバリューのグルーピング方法をハッシュベースではなく、範囲ベースで分割することを提案する。これにより、範囲クエリ時のランダムアクセスを緩和できる。

本稿では、HashKV を範囲ベースに変更した場合の範囲クエリの性能を評価し、今後の研究方針についてまとめる。

2. 実装

実装は、公開されている HashKV のコード[4]を元に行なった。HashKV のハッシュ値に変換しグループを決定する処理を、範囲でグループが決まるように変更を加えた。

3. 評価

WiscKey、HashKV、範囲ベースの HashKV それぞれの範囲クエリにかかる時間を計測した。HashKV でのグループ数は 100 と設定している。キーバリューペアのサイズは 1 KB で、0 から 999999 の範囲のキーをキーバリューストアに書き込んだ。実世界のワークロードを模すために、書き込んだキーに対し、ランダムに 10 万回の更新を行った。そ

1 慶應義塾大学

の後、範囲 1000 （約 1 MB）の範囲クエリを行い、実行時間を計測した。結果を図 1 に示す。

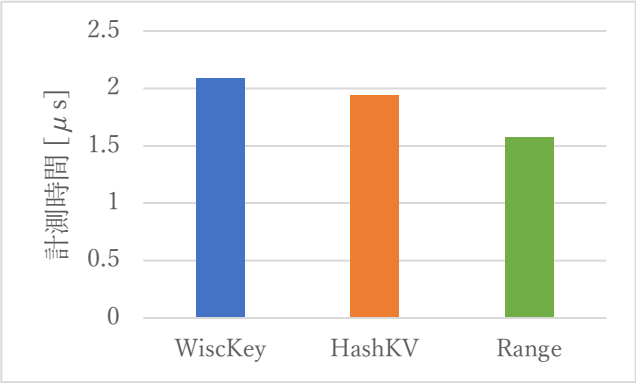

図 1　範囲クエリの計測時間

結果は、WiscKey が 2.09 µs、HashKV が 1.94 µs、範囲ベースの HashKV が 1.57 µs であった。範囲ベースにした場合、WiscKey に対し 1.3 倍、HashKV に対し 1.2 倍の高速化が行えることがわかった。vLog 上のバリューの配置を範囲ベースに変更することで、範囲クエリ時に、同時に読み取られる可能性が高いデータを近くに配置することで、性能を上げることができた。

4. まとめと今後の方針

本稿では、HashKV のグルーピングをハッシュから範囲ベースに変更することで範囲クエリの高速化をすることができた。

今回の評価では、キーの範囲が 0 から 999999 と設定しているが、実際のワークロードではキーの範囲や、分布はわからない。HashKV では、グループ数は固定であり、変えることはできない。また、格納する範囲が固定であると、負荷が偏ることが想定される。そのため、HashKV を範囲ベースに変えるだけでは、実際のワークロードで使用することは難しい。

格納する範囲をワークロードの中で決定する研究として FenceKV[5]がある。FenceKV では、グループが含む最小のキーとしてフェンスを生成し、フェンスベースでグループを決定している。しかし、フェンスは確率的に生成し、一度生成されたら削除されない。そのため、実際の分布を考慮できず、フェンスによっては、分布が偏ってしまう可能性がある。

今後の方針として、我々は、上記問題に対して動的シャーディングを用いて対処する予定である。ワークロードに合わせて、シャードの数やシャードが格納する範囲を動的に変えることで、実際のワークロードにも適応できると考えている。

そのほかの研究の方向として、In SSD processing を用いて、古いデータを削除することで、ホスト側のキャッシュヒット率を上げることなども検討している。キャッシュヒット率の向上により、読み取り性能の向上ができると考えている。

参考文献

[1] O'NEIL, Patrick, et al. The log-structured merge-tree (LSM-tree). Acta Informatica, 1996, 33: 351-385.
[2] LU, Lanyue, et al. Wisckey: Separating keys from values in ssd-conscious storage. ACM Transactions on Storage (TOS), 2017, 13.1: 1-28.
[3] CHAN, Helen HW, et al. {HashKV}: Enabling Efficient Updates in {KV} Storage via Hashing. In: 2018 USENIX Annual Technical Conference (USENIX ATC 18). 2018. p. 1007-1019.
[4] http://adslab.cse.cuhk.edu.hk/software/hashkv/
[5] TANG, Chenlei; WAN, Jiguang; XIE, Changsheng. Fencekv: Enabling efficient range query for key-value separation. IEEE Transactions on Parallel and Distributed Systems, 2022, 33.12: 3375-3386.