1A-04

大容量 Flash を活用したメモリ拡張キャッシュサーバの設計と評価

五木田 駿, 吉田 英司, 三吉 貴史, 小川 淳二
株式会社富士通研究所 コンピュータシステム研究所

1　研究背景

大容量データの需要増加に伴い，多数のサーバの主記憶メモリ上にデータをキャッシングするキャッシュサーバのメモリ容量需要も増大している．一方，現在主記憶として使われている DRAM の容量は，今後大きく増やすことは難しく，サーバ台数を増やせばコストも高くなる．そこで，OS の標準機能であるスワップ機能を使い，デバイスとして SSD を用いてメモリ拡張を行えば，コストを抑えて見かけのメモリ容量を容易に増やすことは可能である．しかし，SSD の記憶デバイスである Flash メモリは DRAM と比較して 3 桁ほどレイテンシが悪く，スワップアウトされたページに対して頻繁にアクセスがあるような状況では大きく性能が低下する．

2　従来のメモリ拡張キャッシュサーバの課題

本章では，図 1 に示す代表的な KVS（Key Value Store）型のキャッシュサーバである memcached のデータ構造を例に，スワップによるメモリ拡張では高速な応答ができない理由を説明する．先ず，Key に対応する Value を取得する Get 処理では，Key を入力としたハッシュ関数を計算し，ハッシュ値をインデックスとした配列であるハッシュテーブルにアクセスを行う．ハッシュテーブルは該当 Key とそれに対応する Value が格納されたアドレスを保持しており，該当 Value を返すことで Get 処理が完了する．ここで，異なる Key が同じハッシュ値を持つ衝突時の対処が必要になるが，memcached では衝突した Key 同士をポインタでつなぎリスト構造化し，リストの先頭から順に Key の一致を確認する連鎖法により衝突時対処が実現されている．このようなデータ構造では，Key のアクセス分布に偏りがあっても，広範囲のメモリ空間に Key が分散して記憶され，かつ探索時には衝突した Key のリストを辿ることになる．この時に，広範囲のメモリ空間に対するランダムアクセスが行われるため，LRU（Least Recently Used）アルゴリズムでページ回収が行われるスワップ方式では，スワップデバイスと DRAM 間でスワップイン/スワップアウトが高頻度で起きてしまい，キャッシュサーバに求められる高速な応答を返すことが困難になる．

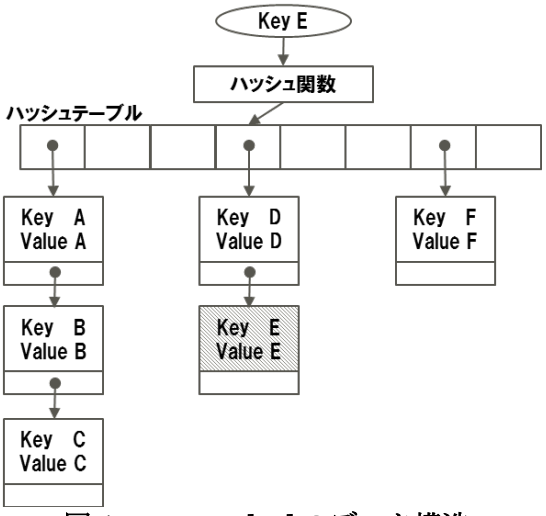

図 1：memcached のデータ構造

3　開発したメモリ拡張キャッシュサーバ

前章で指摘したように，SSD をスワップデバイスとしてメモリ拡張したキャッシュサーバでは通常の DRAM のみのキャッシュサーバと比較して大きく性能が劣る．そこで我々は，キャッシュサーバから DRAM と Flash メモリのデータ配置を制御することで高速化を実現した memcached ベースのキャッシュサーバを開発した．図 2 にその構成を示し，以下の節で詳細を述べる．

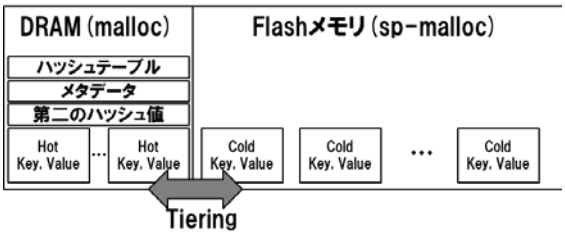

図 2：開発したキャッシュサーバの構成

3.1　２種類のメモリアロケータ

前章で述べたように Flash メモリに高頻度にアクセスされるデータが置かれると大幅に性能が低下する．そのため仮想メモリ空間に DRAM を割り当てる通常の malloc と，Flash メモリを仮想メモリ空間に割り当てる特殊なメモリアロケータ（以下 sp-malloc）の２種類のアロケータを用い [1]，それぞれの割り当てをキャッシュサーバから制御することで高速化を図った．高頻度にアクセスされるハッシュテーブルなどのデータ構造および人気の高い Key, Value ペアには malloc，逆にアクセス頻度が低いデータ構造および人気の低い Key, Value ペアには sp-malloc でリソースを割り当てる．そして，Key, Value ペアに関しては，DRAM と Flash メモリ間で LRU ベースの Tiering を行うことで，DRAM 側に人気度の高い Key, Value ペアを集めることで高速化を図っている．

3.2　省メモリかつ高速な Key 探索手法

memcached の Key 探索アルゴリズムでは，Key に対するアクセス頻度が Value に対して高いため，

Design and Evaluation of Memory Expansion Cache Server Using Large Capacity Flash Memory
Shun Gokita, Eiji Yoshida, Takashi Miyoshi, and Junji Ogawa
Computer Systems Laboratories, Fujitsu Laboratories LTD.

Copyright ©2017 Information Processing Society of Japan.
All Rights Reserved.

高速化のために Key を全て DRAM に割り当てることで，低速な Flash メモリにアクセスさせない方法が先ず考えられる．しかし，memcached では Key のサイズは最大 250 B まで許容しており，かつ比較的小さな 1 KB 前後のデータをキャッシュさせるような用途が多い．そのために例えば，平均 Key サイズが 64 B，平均 Value サイズ 1 KB，DRAM サイズ:Flash メモリサイズ=1:16 の場合には，全ての Key を DRAM に格納した場合にそれだけで DRAM を使い切ることになる．すると，DRAM に Value を全く格納できず，人気のある Value にアクセスする度に Flash メモリにアクセスする必要が出てくる．

上記のような状況を避けるために，従来のハッシュテーブルのインデックスを計算するハッシュ関数 H_X とは別に，H_X が衝突した Key 同士を区別するための第二のハッシュ関数 H_Y を用意する手法である．この関数 H_Y の値を DRAM に格納し，人気のない Key そのものを Flash メモリに記憶することで，省メモリ化を実現する．先ず，H_X が衝突した Key の探索時は DRAM に格納された H_Y の値を参照し，H_Y が一致していればさらに Key 自体の一致を確認して最終的に値を返すことで，H_Y が衝突した場合に対応する．なお，Key, Value が Flash メモリに記憶されていた場合，sp-malloc の仕様上 4KB 単位で DRAM にキャッシュされるために，Key と Value が 4KB ページ境界を跨いで格納されていなければ Flash メモリへのアクセスは一度で済む．図 3 に Key C を探索したときの本提案手法の Key 探索手順を示す．この例では，Key C に対応する Key, Value ペアは DRAM に格納され，Key A, B に対応する Key, Value ペアは sp-malloc により Flash メモリに記憶保持される．ここで，Key C に対応する Key, Value ペアを探索するまでに，DRAM に格納されたハッシュテーブル，H_Y(A），H_Y(B）のみにアクセスが発生するために，高速なアクセスが可能である．省メモリの効果としては，H_Y のサイズを 64 bit とすると，DRAM に全ての Key を格納した場合と比べて DRAM の 87.5%が開放されるために，人気の高い Key, Value ペアを DRAM に格納することができる．なお，例えば H_Y 同士の衝突確率が 1e-15 程度と十分低くなるように H_Y のサイズを大きく取れば，Key 探索中に Flash メモリへのアクセスは発生しない．

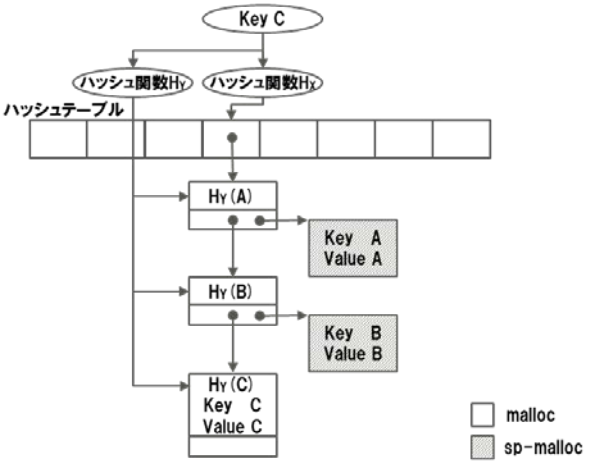

図 3：本提案手法による Key 探索手法

4 評価

本手法の評価は，クライアントから最大負荷をかけたときの memcached サーバの実効スループットを，スワップによるメモリ拡張を行った場合と比較した．ベンチマークは，memcached ベンチマークツールである mutilate に，Facebook データセンタで観測された Key の人気度偏りの分布を実装したものを用いた [2, 3]．Flash デバイスについては，提案手法に対してはソフトウェア定義型 SSD [4]を用い，スワップデバイスに対しては NVMe-SSD である Intel DC P3700 を用いた．図 4 に DRAM と Flash メモリの容量比を変えたときの評価結果を示す．評価結果では，提案手法はスワップによるメモリ拡張に比較して， DRAM と Flash メモリの容量比約 1:10 の場合に約 1.8 倍性能が向上した．また全て DRAM の場合と，提案手法での DRAM と Flash メモリの容量比約 1:10 の点で比較してスループットの性能低下を約-14%に抑えうることを確認した．

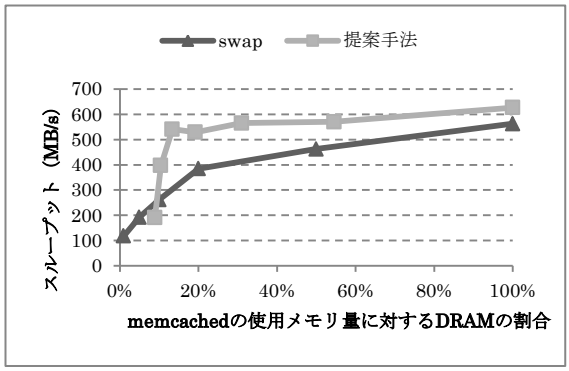

図 4：評価結果

5 まとめ

本提案手法により，キャッシュサーバのメモリ拡張に DRAM の約 10 倍の Flash メモリを用いることで，同容量の DRAM キャッシュサーバと比較して性能を約-14%に抑えつつコストを大幅に低減可能なことを示した．

参考文献

[1] H. Bernhard, et al., “An approach for hybrid-memory scaling columnar in-memory databases”, ADMS’14, 2014.

[2] B. Atikoglu, et al., “Workload analysis of a large-scale key-value store”, SIGMETRICS, 2012.

[3] https://github.com/leverich/mutilate.

[4] 風間哲ほか, “高並列アクセスを可能にする高性能ソフトウェア定義型 SSD の開発”, 2015, 研究報告計算機アーキテクチャ (ARC).

Copyright ©2017 Information Processing Society of Japan.
All Rights Reserved.