不揮発性 DIMM を用いた LSM-tree によるキーバリューストアの性能向上

迫田 賀章[1]　青田 直大[1]　河野 健二[1]

概要：多量のデータを効率的に保持・管理・運用できるストレージシステムとして，キーバリューストア（KVS）が広く一般的に用いられている．本論文では，不揮発性 DIMM（NVDIMM）を用いた KVS のスループット向上手法を提案する．NVDIMM は，通常の DIMM に NAND Flash による待避領域を設けたものであり，不意の電源遮断等に対してもメモリの永続性を提供する．NVDIMM は DIMM と同等のレイテンシでアクセス可能である反面，その容量は DIMM のそれを超えることはできない．KVS は SNS などに用いられることが多く，読出し・書込みの比率が 1:1 となっており，書込みに対しても高いスループットが求められている．本論文では，書込みに対しても高いスループットを達成できる Log-Structured Merge-Tree（LSM-tree）に着目し，NVDIMM を用いて LSM-tree の性能向上を実現する．LSM-tree では，二次記憶上のデータ構造を再構成するコンパクションという処理が頻繁に行われるため，コンパクション時のアクセス遅延が増大する．LSM-tree の管理情報のみを NVDIMM 上に保存することで，コンパクション時の二次記憶へのアクセスを削減し，アクセス遅延の増大を抑える手法を示す．

キーワード：キーバリューストア，NVDIMM

1. はじめに

キーバリューストア（KVS）は Social Networking Service（SNS）や Electronic Commerce（EC）サイトなど，様々な Web サービスで重要な役割を担っている．具体的には，Facebook の RocksDB [1] や Amazon の Dynamo [2]，Google の BigTable [3] などが挙げられる．KVS は任意のデータ（value）に対して一意の key を定めてペアで保存する構造となっており，key や value を指定して Put，Get，Delete などの操作を行うことでデータの保存，管理を行う手法である．関係データベース管理システム（RDBMS）と比較してスケーラビリティに優れており，また高い性能を出しやすい特徴がある [4]．近年のクラウドコンピューティングや SNS の普及に伴い KVS は読込みだけでなく書込みに関しても高いスループットが求められ，ワークロードの書込みの比率が 10〜20 %であったのが 50 %にまで増えている [5]．

本論文では不揮発性 DIMM（NVDIMM）を用いて書込み遅延の小さい KVS を実現する手法を示す．NVDIMM は通常は DRAM として動作し，不意の電源遮断時は NAND Flash にデータを退避させる機能を持った DIMM である．不揮発性でありながら高速かつバイト単位でのアクセスが可能であるという特徴を持つ．近年実用化が始まっており，2015 年 5 月に標準規格の策定が行われ [6]，現在 8GB の NVDIMM が製品化されている [7]．NVDIMM の大きな制約の一つは，その容量が DIMM のそれを超えられないことである．よって KVS 全体を NVDIMM 上に置くことは現実的ではない．また DIMM のスロットを通常の DIMM と共有するため，NVDIMM 上に不揮発性のデータを多く置くと通常のメモリとして利用できるメモリ量が減ることになる．したがって NVDIMM を用いて書込み遅延の小さい KVS を実現するためには，NVDIMM 上に置く不揮発性のデータをできる限り小さくすること，NVDIMM 上に置ききれない場合は二次記憶上に追い出して利用できるデータ構造にすること，通常のメモリ使用量に合わせて動的に NVDIMM に使う領域を増減できることを考慮する必要がある．

本研究では，NVDIMM を想定した書込み遅延の小さい KVS として Log-Structured Merge-Tree（LSM-tree）[8] を題材に設計を行う．LSM-tree は書込みに適したデータ構造であり，多くの KVS で利用されている [1], [3], [9], [10]．key と value を LSM-tree 上で構成することでシーケンシャルにまとめてディスクへ書き込むことができ，書込み性能を向上させる．LSM-tree ではディスク上のファイル

[1] 慶應義塾大学
Keio University

© 2016 Information Processing Society of Japan

が一定数に達すると，それらをマージするコンパクションという処理が発生する．コンパクションの際，複数のファイルを読み書きする必要があるため Write Amplification（WA）に繋がり遅延が生じる．

そこで key と value ではなく key と offset で LSM-tree を構成する手法を提案する．これにより，NVDIMM 上のデータ構造を圧縮できる．さらに LSM-tree の構造を維持することで，NVDIMM とディスクの間でデータを移動するときにデータ構造の変換が不要となる．また LSM-tree の特徴が維持されるため disk I/O が非効率になることもない．コンパクション発生時には NVDIMM 上で処理が完結するため，遅延を抑えることができる．

既存手法を用いた KVS の LevelDB [10] と提案手法の書込み遅延を比較する実験を行った．1KB のデータを 100 万回 Put するワークロードを用いたところ，LevelDB と比較して約 40 %の遅延を削減した．また，Yahoo! Cloud Serving Benchmark（YCSB）[11] を利用して Read と Update の比が 1：1 のワークロードを用いたところ平均約 8.7 %の遅延を削減した．コンパクション発生時に関しては平均約 15 %の遅延を削減した．

本論文の構成を以下に示す．2 章では NVDIMM の特性とその活用法について説明する．3 章では LSM-tree の特徴と構造について説明する．4 章では本研究が提案する手法の概要及び設計を示す．5 章では提案手法と既存の手法の書込み遅延を比較した実験について述べる．6 章では関連研究について紹介する．7 章ではまとめを述べる．

2. NVDIMM

2.1 NVDIMM の特性

NVDIMM は NADN Flash の不揮発性と DRAM の高速かつバイト単位でのアクセスを両立させるデバイスである．通常時は DRAM として動作し，不意の電源遮断時は付随している NAND Flash にデータを退避させることで不揮発性を維持する．2015 年 5 月に NAND Flash をバックアップとして利用する NVDIMM-N という規格の策定が行われ [6]，2016 年 3 月には NVDIMM-N に準拠した NVDIMM が搭載されたサーバが発表されている [7]．Phase Change Memory（PCM）など，不揮発性のメインメモリとして注目を集めている Storage Class Memory（SCM）の前身として実用化が期待されている．また DIMM と同じ速度で読み書きできるため，SCM のように遅延が発生しないことも特性の一つである．

その一方 NVDIMM の大きな制約の一つは，その容量は DIMM のそれを超えられないということである．製品化された NVDIMM の最大サイズは 8GB であり，全てのデータを NVDIMM に載せることは現実的ではない．さらに NVDIMM は通常の DIMM とスロットを共有するため，NVDIMM 上に格納する不揮発性のデータ量を動的に

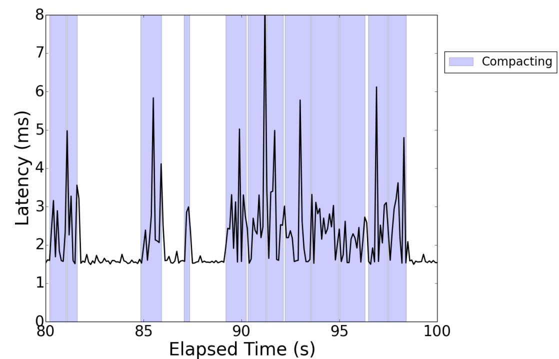

図 1　書込み遅延に関する予備実験結果

変更しなければならない．NVDIMM 上の不揮発性のデータのサイズを大きくしそこに容量を多く割くと，通常のメモリとして使用できる容量が減ることになる．

2.2 NVDIMM の活用

以上のような NVDIMM の特性を踏まえた上で NVDIMM を活用するためには，NVDIMM 上に置くデータに関して工夫する必要がある．まず，前提として NVDIMM 上に置くデータをできる限り小さく抑えるようにする．次に NVDIMM 上に置く不揮発性のデータ量を動的に増減できるようにする．これは通常のメモリ使用量が多い場合は通常のメモリとしての領域に NVDIMM の容量を譲り，通常のメモリの使用量が少ない場合は不揮発性のデータを多く置くことができるようにするためである．そしてそのためには NVDIMM とディスクの間でデータをやりとりする際，データ構造の変更などの大きなオーバヘッドを避けるようにする必要がある．したがって NVDIMM に置くデータ構造はディスク上の構造と近いものにする必要がある．

3. Log-Structured Merge-Tree

LSM-tree は書込みに適したデータ構造であり，多くの KVS で利用されている [1], [3], [9], [10]．この章では LSM-tree の特徴及びその構造について説明する．

3.1 LSM-tree の特徴

LSM-tree の特徴を調査するために書込み遅延に関する予備実験を行った．実験には，Intel Xeon E3-1240 3.40GHz，8GB のメインメモリ，1TB の HDD を搭載しているマシンを用いた．KVS として LevelDB を利用した．ベンチマークには YCSB を使用し，1KB のデータが 100 万回挿入された合計 1GB のデータに対し 100 万回のトランザクションが発生するワークロードを用いて，書き込み遅延を測定した．トランザクションの比率は Read と Insert が 1：1 とした．ベンチマーク実行中の一部を抜き出した書込み遅延を図 1 に示す．実験結果より書込み中に大きな遅延が発生する期間があることがわかる．図中の塗りつぶし

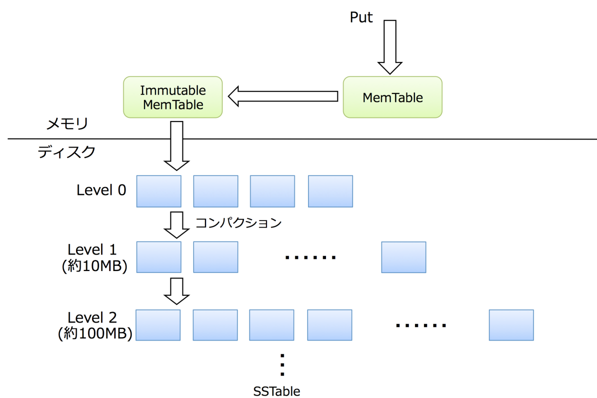

図 2　LevelDB の概要

がされている部分はコンパクションという動作が発生している．コンパクションが発生している時に，書込み遅延が大きくなっていることが示されている．次の節でコンパクションを含め，LSM-tree の構造について説明する．

3.2　LSM-tree の構造

LSM-tree の構造を LevelDB を用いて説明する．LevelDB は BigTable のアーキテクチャをベースにし，LSM-tree を基に実装された KVS である．

LevelDB の概要を図 2 に示す．LevelDB に挿入された key-value（KV）ペアはメモリ上の構造（MemTable）に保存される．MemTable に保存された KV ペアが一定サイズに達すると，MemTable は読込み専用で書き換えられない Immutable MemTable へ変化する．その際に新たな MemTable が作られ，それ以降に挿入された KV ペアはその新たな MemTable に保存される．それと同時にバックグラウンドスレッドで Immutable MemTable の内容を key 順にソートされた KV ペアを保存している Sorted String Table（SSTable）というファイルとしてディスクへ書き出す．SSTable は Level ごとにまとまりとして構成される．Immutable MemTable から書き出された SSTable は Level 0 とされる．それぞれの SSTable はメタデータとして key range を保持しており，Level 0 以外の SSTable はその Level 内で key range が重複しないようにする．KV ペアを削除する際は，削除フラグを立てた KV ペアを挿入することで論理的に削除し，探索しても見つからないようにする．

それぞれの Level には SSTable ファイルの上限数あるいは Level 全体のサイズの上限がある．サイズの上限は Level が高くなるに連れて増えていく．例えば Level 1 の上限が 10MB とすると，Level 2 の上限は 100MB となる．各 Level が上限を超えた際，ファイル数を少なくし，その Level のサイズを小さくする動作が発生する．例えば Level N が上限を超えた際は，Level N の中から 1 つの SSTable を選択し，その key range と重複した key range を持つ Level N+1 の SSTable 全てとマージソートを行うことで新しい SSTable ファイルを生成し，それらを Level N+1 に書き出す．この動作をコンパクションという．コンパクション時に削除フラグが立っているデータがあれば，そのデータは次の Level に書き出さず，物理的に削除される．よって更新されないデータはより高い Level へ移っていくこととなり，より低い Level にはより新しいデータが格納されることになる．

コンパクションはディスク上のデータ構造を再構成することになる．コンパクションの対象となるファイルを全てディスクから読込み，マージソートを行い作成したファイルを再びディスクに書き出す．このように複数のファイルを読み書きする必要があるためディスク I/O が頻繁に発生する．また実際に新しく作られる SSTable のデータサイズ以上に，ディスクに対して物理的な書込みが行われる Write Amplification (WA) という現象が発生する．これらによりコンパクションの処理は非常に重いものとなり，SSTable が多く大きくなると Immutable MemTable の内容をディスクに書き出す処理が遅くなる．すると MemTable が Immutable MemTable に変更できず，新たなデータの挿入に滞りが発生する．したがって前節の実験結果にある通り，コンパクション時は書込み遅延が大きくなる．

KV ペアを探索するときは，まず MemTable，次に Immutable MemTable，そして SSTable を Level 0 から高い Level へ順に探索する．Level 1 以降の SSTable はその Level 内で key range が重複していないため，複数の SSTable を探索する必要はない．したがって新しく挿入されたデータから順に探索することになる．また，複数の Level を何度も探索することを防ぐために Bloom Filters [12] を用いている．

4.　提案

本論文では NVDIMM 上に key を LSM-tree の構造で保存し，ディスク上に value をシーケンシャルに保存することで書込み遅延の小さい KVS を提案する．この章では提案手法とその設計について説明する．

4.1　概要

LSM-tree の書込み遅延を小さくするためにはコンパクションによる WA を防ぐことが大切である．コンパクション動作時にディスク I/O を減らすために， key と value が格納されている LSM-tree 全体を NVDIMM 上に格納しコンパクションを NVDIMM 上で完結するという方法が考えられる．しかし，2 章で述べたように NVDIMM の容量が限られているため全てのデータを NVDIMM 上に格納することは現実的ではない．

そこで NVDIMM 上のデータサイズを小さくするため，NVDIMM 上には key のみを保存することで NVDIMM

© 2016 Information Processing Society of Japan

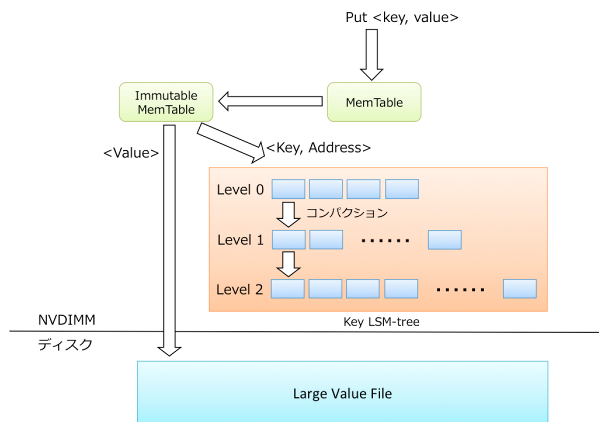

図 3　提案手法の概要

の特性を活かしつつ KVS の性能向上を図る．NVDIMM 上には key とそれに対応する value のディスクアドレスを LSM-tree の構造で保存し，全ての value はディスク上の巨大なファイルにシーケンシャルに保存するように設計する．これにより LSM-tree のコンパクションは NVDIMM 上で完結しディスク I/O が発生しなくなるため，コンパクション時の書込み遅延を抑えることができる．ディスクへの書込みはコンパクションと無関係になり単に value をシーケンシャルに書き出すため，必要以上の書込みを防ぐことができ WA を削減することができる．また既存手法では LSM-tree がディスク上にあったため Get の際にディスク上を探索する必要があったが，本提案では key は NVDIMM 上にあるため key の探索を高速で行うことができる．これにより間接的な効果として読込み遅延についても削減することができる．

提案手法の全体のデータ構造を図 3 に示す．挿入された KV ペアは NVDIMM 上の MemTable に保存され，MemTable が一定のサイズに達した際 Immutable MemTable に変更する．Immutable MemTable の内容をディスクへ書き出すとき，key とそれに対応する value のディスクアドレスを NVDIMM 上に LSM-tree の構造（Key LSM-tree）で保存し，value はまとめてディスクにファイル（Large Value File）としてシーケンシャルに書き出す．KV ペアを探索するときは，まず MemTable，Immutable MemTable の順に探索し，次に Key LSM-tree を探索する．Key LSM-tree で key を見つけた後，そのペアとして保存されているディスクアドレスを基に Large Value File から value を読み込む．KV ペアを削除するときは，既存の LSM-tree と同様に削除フラグを立てた KV ペアを挿入し，コンパクション時にその key を持つエントリを Key LSM-tree より削除することで，探索しても見つからないようにする．なお，この際 Large Value File から value は削除されないため，ディスクを圧迫することになる．したがって不要になった value は将来的にディスクから削除する必要がある．これに関しては WiscKey [13] の手法を参考にし，手動でガーベジコレクションを行うことで定期的にディスクの不要な value を削除することで解決する．

4.2　コンパクション時の処理

コンパクションによる Key LSM-tree の更新は全て NVDIMM 上で行われる．既存手法ではコンパクションに関するファイルを全て一度メモリ上に読み込む必要があったが，本提案では既に NVDIMM 上に存在するため読み込む必要がない．マージソートを行った後も再びディスクに書き出すことはなく，そのまま NVDIMM 上に残しておけば良い．LSM-tree に格納されている value に対してのディスクアドレスは変わらないため，ディスク上の Large Value File にアクセスをする必要はない．したがって，コンパクション時にはディスク I/O が発生しないことになる．よってコンパクションの処理に時間がかからなくなり，書込みが滞ることなく行われるため，書込み遅延を抑えることができる．

4.3　NVDIMM 上の構造

LSM-tree はシーケンシャルにディスクへ書込むことで書込み性能を向上させる構造であり，本来低速な I/O を行うディスクを前提として設計されている．よって NVDIMM はバイト単位でのアクセスができるため必ずしもディスクに適した構造でなくても良いが，提案手法では NVDIMM 上のデータは LSM-tree に準拠した構造にする．なぜなら NVDIMM は容量に制約があり，NVDIMM とディスク間でデータの移動を発生させる事態を想定する必要があるためである．万が一 NVDIMM 上のデータがディスクに追い出されることになった際，NVDIMM 上のデータ構造がディスクのそれと同じであれば，NVDIMM とディスクの間でデータを移動する際にデータ構造を変更する必要がなくなりオーバヘッドの発生を防ぐことができる．したがって，key が大量に挿入されたり通常のメモリ使用量が多くなったりし NVDIMM の容量が足りなくなった際は，NVDIMM 上のデータをそのままディスクに書き出すことができ，NVDIMM の特性を活用することができる．

本提案の Key LSM-tree は LevelDB の LSM-tree と同様に階層的に SSTable を保存する．それぞれの SSTable は key とそれに対応する value のディスクアドレスを保持し value 自体は保持しないため，既存手法と比較し LSM-tree のサイズが小さくなる．Key LSM-tree をディスクに書き出す場合，いくつかの SSTable をそのままファイルとして書き出すことになる．このとき，高い Level の SSTable を優先的に書き出す．それにより比較的新しく挿入されたデータの key を NVDIMM に残すことができ，新しく挿入された key を素早く探索することができる．

Vol.2016-OS-138 No.17
2016/8/9

表 1 簡易的なベンチマークを使用した実験結果

	LevelDB	提案手法
平均書込み遅延 [μs]	57.887	34.264
スループット [kops/s]	17.145	28.804

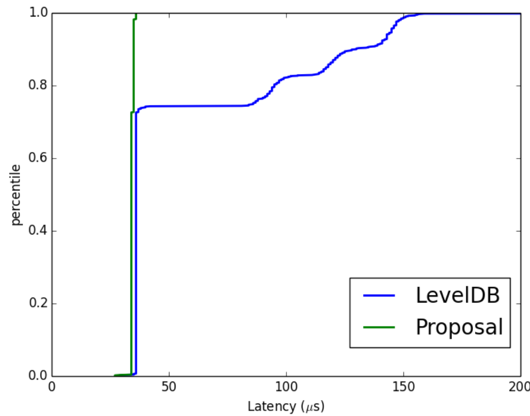

図 4 書込み遅延の累積分布図

5. 書込み遅延の比較

既存手法を利用している LevelDB と提案手法の書込み遅延を 2 種類のベンチマークで比較した．1 つは単に書込みのみを行う簡易的なベンチマークを使用し，もう 1 つは YCSB を使用し読み書きどちらも行うワークロードを用いた．いずれの実験も，Intel Xeon E5620 2.40GHz，16GB のメインメモリ，500GB の SSD を搭載しているマシンを用いた．

5.1 書込みベンチマーク

1KB のデータを 100 万回 Put するワークロードを用いて，書込みの性能を比較した．この簡易的な書込みベンチマークを用いた実験結果を表 1 に示す．提案手法を用いた場合，LevelDB と比較して平均で書込み遅延を 40.8 %削減し，スループットが 1.68 倍に向上した．これより LevelDB と比較し，提案手法では書込み遅延を抑え，性能を向上させていることが示された．

また，書込み遅延に関する累積分布図を図 4 に示す．約 70 %以上の遅延時間は LevelDB と本提案でさほど差がないことがわかる一方，LevelDB では 100μs より大きな遅延が発生しているのに対し，本提案では多くの遅延が 50μs に収まっている．これより LevelDB ではコンパクションが発生する際に平均と比較して大きな書込み遅延が発生し，それが全体の書込み遅延を増加させる原因となっていることがわかり，一方提案手法では大きな遅延が発生することがなく，コンパクションによる書込み遅延を削減できていることが示された．

表 2 YCSB を使用した実験結果

	LevelDB	提案手法
平均書込み遅延 [ms]	3.9588	3.6140
コンパクション時の書込み遅延 [ms]	4.2550	3.6304
平均読込み遅延 [ms]	1.9007	1.7103
コンパクション時の読込み遅延 [ms]	1.9687	1.6957
平均スループット [kops/s]	5.4008	5.9673
コンパクション発生回数 [回]	145	120
コンパクション発生時間 [s]	140.65	4.6398

5.2 YCSB

YCSB を使用し，1KB のデータが 5 万回挿入された合計 50MB のデータに対し 80 万回のトランザクションが発生するワークロードを用いて，読み書きそれぞれの性能を比較した．トランザクションにおける Read と Update の比率は 1 : 1 とし，16 のクライアントが並行に処理を行うようにした．Read は単に Get を行うトランザクションであり，Update は Get したあとに同じ key で別の value のデータを Put するトランザクションである，

YCSB を用いた実験結果を表 2 に示す．提案手法を用いた場合，LevelDB より平均で書込み遅延を 8.7 %削減した．特にコンパクション時は書込み遅延を 14.7 %削減している．提案手法において，コンパクション時の書込み遅延と平均書き込み遅延にほとんど差がないことがわかる．またコンパクションの発生回数が LevelDB と提案手法であまり変わらないのに対し，提案手法のコンパクション発生時間は LevelDB の 3.30 %と非常に小さくなっている．すなわち，1 回のコンパクションの発生時間が提案手法では大きく削減されている．これらより，提案手法でのコンパクション時は書込み遅延をほとんどを発生させず，さらにコンパクションの処理自体を高速に処理できるため，全体の書込み性能を向上させていることが示された．さらに，読み書きどちらも行うワークロードにおいても提案手法の書き込み性能が LevelDB と比較して向上することが示された．

また読込みに関しても平均で 10.0 %の遅延を削減した．これにより提案手法では key を全て NVDIMM に置くことで探索時間が短縮されており，LevelDB と比較して読込みに関しても性能を向上させられることがわかった．読込み遅延に対してコンパクションは大きな影響を与えないことも実験結果より示された．

次に，経過時間ごとの書込み遅延の変化を図 5 に示す．ベンチマークを動かした直後は初期の処理が実行されるためオーバヘッドが発生し，書込み遅延が大きくなっているが，その後は基本的にはある程度一定の書込み遅延を保っていることがわかる．しかし，LevelDB はコンパクションの影響により大きな書込み遅延が発生している箇所があることを確認できる．それに対し提案手法では，特別書込み遅延が大きくなることはなく，コンパクションによる書込

© 2016 Information Processing Society of Japan

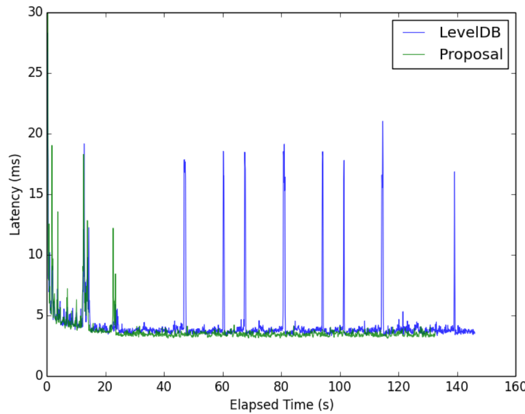

図 5　経過時間ごとの書込み遅延

みへの影響がないことが示された．

6. 関連研究

NVDIMM を利用した研究がなされている．WSP [14] は全てのメモリが NVDIMM と想定し，failure 発生時に Flash に書き出すことでアプリケーションの状態を保持し，永続性を保証する手法を提案している．本論文では現実的なサイズの NVDIMM を想定している．

近年，KVS の性能向上に関する研究が多くなされている．SILT [15] は 3 つのデータ構造を組み合わせることで読込み性能向上を図る KVS を提案している．データ構造を変化させる処理のオーバヘッドがボトルネックとなり，書込み性能が低下する．LOCS [16] は Open-Channel SSD を使用し，SSD の並行性を活用することでコンパクションの遅延を低減し，性能向上を図る KVS を提案している．Open-Channel SSD を前提としており，SSD へのディスクアクセスが多く発生する．LSM-trie [17] は key をハッシュ key として保存しメタデータのサイズを抑えることによって，読み書きどちらの性能も向上させる KVS を提案している．key をハッシュするため key 順にソートされないため，範囲探索を効率よく行うことができない．WiscKey [13] は key と value を分けることで SSD の特性を活かすことで性能向上を図る KVS を提案している．本論文と考え方は似ているが，WiscKey が SSD を想定しているのに対し，本研究では NVDIMM を想定した KVS を設計しているという部分で異なる．

7. まとめ

近年，KVS は様々な Web サービスで活用されており，書込みに関しても高い性能が求められる．本論文では NVDIMM を利用することで KVS の性能向上を図る手法を提案した．NVDIMM は DIMM と同様のアクセス速度を実現することができる一方，容量に関して制約がある．そこで LSM-tree というデータ構造に注目し，NVDIMM の特性を活かす KVS を設計した．key を NVDIMM に格納することで LSM-tree のボトルネックであるコンパクション時の書込み遅延の削減を図った．実験により，書込みのみを行うワークロードでは書込み遅延を約 40 %削減する結果を得た．また，読込みに関しても遅延を削減できることを確認した．

参考文献

[1] Facebook: RocksDB, http://rocksdb.org/ (2013).
[2] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P. and Vogels, W.: Dynamo: Amazon's Highly Available Key-value Store, *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, New York, NY, USA, ACM, pp. 205–220 (2007).
[3] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A. and Gruber, R. E.: Bigtable: A Distributed Storage System for Structured Data, *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, OSDI '06, Berkeley, CA, USA, USENIX Association, pp. 15–15 (2006).
[4] Stonebraker, M.: SQL Databases v. NoSQL Databases, *Commun. ACM*, Vol. 53, No. 4, pp. 10–11 (2010).
[5] Sears, R. and Ramakrishnan, R.: bLSM: A General Purpose Log Structured Merge Tree, *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, New York, NY, USA, ACM, pp. 217–228 (2012).
[6] JEDEC: JEDEC Announces Support for NVDIMM Hybrid Memory Modules, https://www.jedec.org/news/pressreleases/jedec-announces-support-nvdimm-hybrid-memory-modules (2015).
[7] HewlettPackard: Hewlett Packard Enterprise Expands Server Portfolio with New Innovations in Compute, http://www8.hp.com/us/en/hp-news/press-release.html?id=2203338 (2016).
[8] O'Neil, P., Cheng, E., Gawlick, D. and O'Neil, E.: The Log-structured Merge-tree (LSM-tree), *Acta Inf.*, Vol. 33, No. 4, pp. 351–385 (1996).
[9] Lakshman, A. and Malik, P.: Cassandra: A Decentralized Structured Storage System, *SIGOPS Oper. Syst. Rev.*, Vol. 44, No. 2, pp. 35–40 (2010).
[10] Dean, J. and Ghemawat, S.: LevelDB, http://leveldb.org/ (2011).
[11] Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R. and Sears, R.: Benchmarking Cloud Serving Systems with YCSB, *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, New York, NY, USA, ACM, pp. 143–154 (2010).
[12] Bloom, B. H.: Space/Time Trade-offs in Hash Coding with Allowable Errors, *Commun. ACM*, Vol. 13, No. 7, pp. 422–426 (1970).
[13] Lu, L., Pillai, T. S., Arpaci-Dusseau, A. C. and Arpaci-Dusseau, R. H.: WiscKey: Separating Keys from Values in SSD-conscious Storage, *14th USENIX Conference on File and Storage Technologies*, FAST '16, Santa Clara, CA, USENIX Association, pp. 133–148 (2016).
[14] Narayanan, D. and Hodson, O.: Whole-system Persistence, *Proceedings of the Seventeenth International*

© 2016 Information Processing Society of Japan

Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS XVII, New York, NY, USA, ACM, pp. 401–410 (2012).

[15] Lim, H., Fan, B., Andersen, D. G. and Kaminsky, M.: SILT: A Memory-efficient, High-performance Key-value Store, *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, New York, NY, USA, ACM, pp. 1–13 (2011).

[16] Wang, P., Sun, G., Jiang, S., Ouyang, J., Lin, S., Zhang, C. and Cong, J.: An Efficient Design and Implementation of LSM-tree Based Key-value Store on Open-channel SSD, *Proceedings of the Ninth European Conference on Computer Systems*, EuroSys '14, New York, NY, USA, ACM, pp. 16:1–16:14 (2014).

[17] Wu, X., Xu, Y., Shao, Z. and Jiang, S.: LSM-trie: An LSM-tree-based Ultra-large Key-value Store for Small Data, *Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC '15, Berkeley, CA, USA, USENIX Association, pp. 71–82 (2015).

© 2016 Information Processing Society of Japan