Vol.2016-OS-138 No.1
2016/8/8

DBMS 統合環境における トランザクション書き込みの高速化機構

斎藤 直人[1,a)]　山田 浩史[1,b)]

概要：仮想化技術を利用したデータベース管理システム (DBMS) の統合が実運用されている．DBMS はトランザクションをログ先行書き込みで実現しているものが多く，ログ先行書き込みはストレージへシーケンシャルに書き込むことで高速化している．しかしながら，DBMS の統合を行うと，複数の DBMS で発生したログ書き込みが同じディスク上の各 VM イメージ内の DBMS ログファイルを往復しながら書き込まれるためシーケンシャルに書き込みを行えない．本研究では，複数 DBMS で発生したログ書き込みをハイパーバイザで制御し，シーケンシャルに行う手法を提案する．提案手法では，複数の VM の各ディスクイメージに保存されていた DBMS のログファイルを 1 つに統合する．そして，複数 DBMS からのログ書き込みを制御し，統合されたログファイルへシーケンシャルに書き込む．提案手法を Xen 4.2.1，MySQL 5.6.17 に実装し，4 つの DBMS で同時にワークロード TPC-C を実行した．結果，各 DBMS のスループットの平均で 23%向上し，提案手法の有効性が確認できた．

1. Introduction

仮想化技術を利用したデータベース管理システム（DBMS）の統合が実運用されている．仮想化技術とはハードウェアを仮想化し，1 台の物理マシン上で複数の仮想マシン（VM）を動作させる技術である．仮想化技術を利用した DBMS の統合では，1 台の物理マシン上で複数の VM を起動し，各 VM 上で DBMS を動作させる．DBMS の統合を行うことにより，VM 数の増減によってサービスの規模拡張が容易になることや，効率的にリソースを利用することで管理する物理マシン数を減らすことができるなどのメリットがある．実際に，Amazon RDS [1] などのサービスで利用されている．Amazon RDS は DBMS をオンラインで作成，利用できるサービスである．

一般的に利用されている DBMS の多くはログ先行書き込み (WAL) でトランザクション書き込みを実現している．ログ先行書き込みは全てのデータベースの更新内容を事前にログファイルに書き込む．ログファイルからデータベースへの反映は非同期で行われるが，すでにログファイルへの書き込みでディスクに書き込まれているため，突然 DBMS が障害によってダウンした場合でもログファイルからデータベースの状態を復元できる．ログファイルへの書き込みは頻繁に生じるため，全てシーケンシャルに行うことで処理の高速化を行っている．ログ先行書き込みは MySQL [2]，PostgreSQL [3]，SQL Server [4]，Oracle Database [5] など多くの DBMS で採用されている．

しかしながら，ログをシーケンシャルに書き込むことによる DBMS のトランザクション書き込みの高速化は，仮想化環境で複数の DBMS が動作すると無効化されてしまう．複数の VM で発行されたディスクへの書き込みはハイパーバイザでまとめて処理する．複数の DBMS のログファイル間を往復しながら，小さな書き込みを頻繁に処理するため，結果的にランダム書き込みになってしまう．シーケンシャル書き込みが減少すると遅いランダム書き込みが増え，DBMS 統合環境で動作する DBMS の性能が劣化してしまう．その結果，望んだ性能を DBMS に発揮させるためには DBMS の統合数を減らすことになり，効率的にリソースを利用するという DBMS 統合の効果が小さくなってしまう．

本研究では，複数の DBMS からのトランザクション書き込みを制御して，DBMS のスループットを向上させる手法を提案する．提案手法では，複数の VM の各ディスクイメージに保存されていた DBMS のログファイルを 1 つに統合する．そして，複数 DBMS からのログ書き込みを制御し，統合されたログファイルへシーケンシャルに書き込む．提案手法により，ディスクへのシーケンシャルアクセスを増加させ，トランザクション書き込みの高速化が可能となる．

[1] 東京農工大学
[a)] saito@asg.cs.tuat.ac.jp
[b)] hiroshiy@asg.cs.tuat.ac.jp

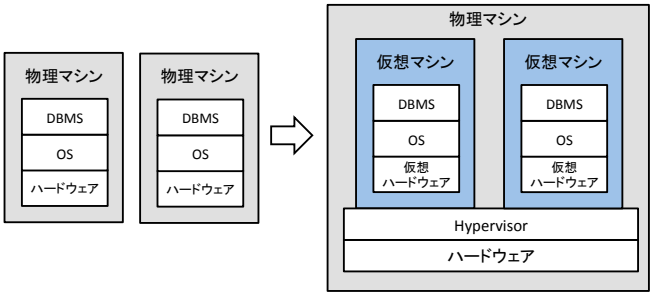

図 1 2 つの DBMS を統合する例

提案手法を Xen 4.2.1，Linux 3.4.103，MySQL 5.6 上に行った．4 つの DBMS が統合された環境で提案手法の評価実験を行ったところ，insert のみを繰り返し行う micro benchmark では 72%性能向上し，TPC-C [6] では 23%スループットが向上した．

2. Background and Motivation

2.1 DBMS Consolidation

仮想化技術を利用したデータベース管理システム（DBMS）の統合は，複数の物理マシン上で一つずつ動作していた DBMS を仮想化技術を利用して 1 台の物理マシン上で複数の DBMS を動作させる．2 つの DBMS を統合する例を図 1 に示す．DBMS を統合することにより，リソースの有効活用が可能になる．DBMS は常にマシンのリソースをすべて使うわけではない．そのため，1 台の物理マシンに一つの DBMS を動作させても，使用していないリソースが発生する．仮想化技術を利用して，1 台の物理マシンに複数の DBMS を動作させることでリソースの共有が可能になり，リソースの無駄を減らすことができる．また，VM 数の増減によりサービスの規模拡張が容易に行える．

一般的な DBMS としてリレーショナルデータベース管理システム（RDBMS）が利用されている．RDBMS はその多くがトランザクションを提供している．トランザクションは SQL クエリのまとまりをアトミックに処理するための DBMS の機能である．一連のクエリが全てデータベースに反映されるか，全て反映されないかのどちらかの状態になることを保証する．トランザクションの一般的な実現方法としてログ先行書き込みがある．ログ先行書き込みではトランザクションによる変更をデータベースに反映する前にログファイルに書き込み，データベースへの反映は非同期で行う．ログファイルに変更は書き込まれているため，データベースへ反映する前にマシンが障害でダウンしてもログファイルからデータを復元できる．ログファイルへの書き込みは頻繁に発生するため，ディスクへシーケンシャルに書き込みむことで高速化している．ログ先行書き込みは MySQL [2]，PostgreSQL [3]，SQL Server [4]，Oracle Database [5] など多くの DBMS で採用されている．

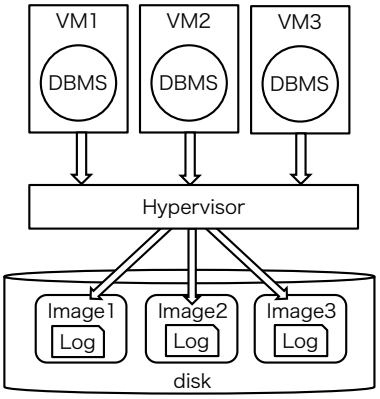

図 2 DBMS 統合環境におけるログファイルへの書き込み

2.2 Perforamnce Degradetion of DBMS Consolidation

ログをシーケンシャルに書き込むことによる DBMS のトランザクション書き込みの高速化は，仮想化環境で複数の DBMS が動作すると無効化されてしまう．仮想化環境における VM の I/O はハイパーバイザに集約され，ハイパーバイザによって実際の I/O を実行する．3 つの DBMS を統合した環境における各 DBMS のログファイルへの書き込みの様子を図 2 に示す．複数の DBMS でログへの書き込みが発生すると，ハイパーバイザで集約して各 VM のディスクイメージへ書き出す．結果として，複数の DBMS のログファイルを往復しながら，小さな書き込みを頻繁に行うランダム書き込みになってしまう．

複数 DBMS のログ書き込みがランダム書き込みになっていることを確認するために実験を行った．同じマシン上でストレージを共有している 2 台の VM 上の DBMS を使用する．DBMS 数を 1 台，2 台と変更し，それぞれでワークロードを実行した際のディスクの書き込み先を時系列順に取得し，ディスクアクセスがどのように行われているか調べる．ワークロードは insert のみを行うトランザクションを 100 回繰り返し実行するものである．実験結果を図 3 に示す．縦軸がディスクの論理ブロックアドレスで，横軸が epoch time となっている．DBMS が 1 台のときは論理ブロックアドレス 7.7×10^8 付近にアクセスが集中している．これは DBMS のログファイルがそこに配置されており，ワークロード実行中，コミット毎にそのログファイルにシーケンシャルに書き込みが行われたためである．DBMS が 2 台のときは論理ブロックアドレス 7.7×10^8 へのアクセスに加えて，5.6×10^8 付近へのアクセスが増えている．これは 2 台目の DBMS のログファイルへの書き込みによるものである．2 つのログファイルを往復しながら書き込まれており，ログをシーケンシャルに書き込めていないことが確認できる．

TPC-C の実行結果ではトランザクションは 1 つの VM で毎分 600 程度処理されており，DBMS を統合すると更

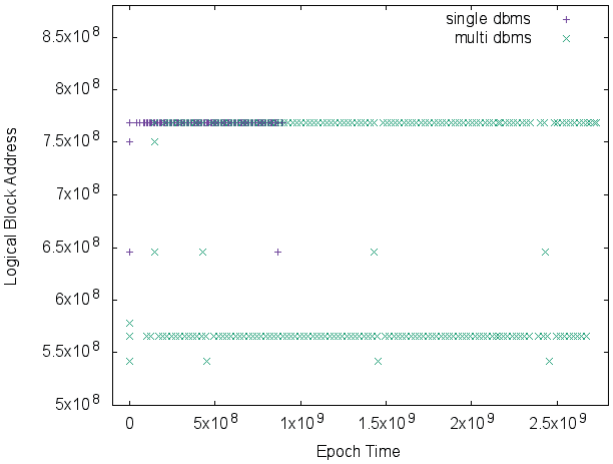

図 3 DBMS ワークロード実行時のディスクアクセスログ
2 台の DBMS が動作すると各ログファイルへアクセスが分散

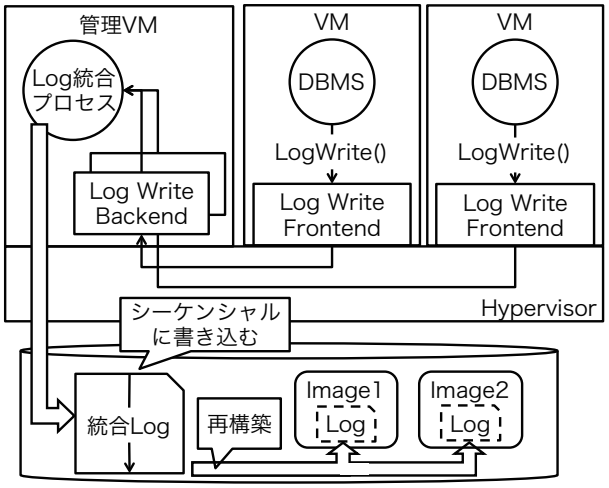

図 4 提案手法の全体像

に処理すべきトランザクションが増える．図 3 で示したように DBMS 統合環境では複数のログファイルにアクセスされるため，最悪の場合 1 つ 1 つのトランザクション書き込みがランダム書き込みになってしまうことになる．ストレージにおいてランダムアクセスはシーケンシャルアクセスに比べて遅いことが知られており，fio ベンチマークによってスループットを測定すると，ハードディスクではシーケンシャル書き込みはランダム書き込みの 70 倍高いスループットで処理ができる．

また，SSD においてもシーケンシャル書き込みの方がランダム書き込みに対して約 30%高いスループットを発揮する．SSD ではデータの更新を行う際に更新する部分のデータを 1 度削除した後，データがフラッシュされる．データの削除は 1 ブロック単位で行われるため，書き込むデータのサイズが小さくても 1 ブロック分の削除と書き込み処理が行われる．ランダム書き込みでは小さいサイズのデータの更新が発生することに対して，シーケンシャル書き込みではまとまったより大きなサイズで更新が行われ，無駄な消去と書き込み処理が少なくなり，性能が向上する．

DBMS 統合環境で遅いランダム書き込みが増えると，各 DBMS のスループットが大きく劣化してしまうと考えられる．結果として，DBMS 統合環境を用意する際，書き込み性能を挙げるためにはいくつかのディスクに VM を分散して配置する必要があり，より多くのディスクを用意するための余分なコストが発生してしまう．

3. Approach

本研究では複数の DBMS からのトランザクション書き込みを制御して，DBMS のスループットを向上させる手法を提案する．まず，統合されている DBMS ごと存在していたログファイルを 1 つに統合する．そして，ハイパーバイザで複数 DBMS のログ先行書き込みを捕捉し，シーケンシャルに統合したログファイルへ書き込む．提案手法を実現する上で 2 つの課題点がある．1 つ目，ハイパーバイザで DBMS のログ先行書き込みによる書き込みリクエストを検知できない．ハイパーバイザでは VM のディスクイメージへの書き込み先アドレスなどの抽象化された情報しか得られないため，ログ先行書き込みによるリクエストを判別できない．2 つ目，ログファイルから DBMS がリカバリ可能にしなければならない．ログ先行書き込みは障害時に復旧するためにログファイルにデータを永続化している．そのため，提案手法により統合したログファイルから障害時にリカバリ可能にしなければならない．

前述した課題点を解決するため，DBMS のログ書き込みをフックし，通常の I/O とは異なる独自のパスを通して，ログ書き込みリクエストを管理 VM に渡す．管理 VM で取得したログ書き込みリクエストは統合したログファイルにシーケンシャルに書き込む．また，統合したログファイルからリカバリを行う方法として，統合ログファイルから同じ VM のログのみを集めてオリジナルのログを再構築する．各 DBMS は再構築したログを読み込むことで従来のリカバリ機構を利用してリカバリが可能になる．これらの設計について次章で詳細に述べる．

4. Design Details

提案手法では複数 DBMS のログファイルを 1 つに統合する．提案手法の全体像を図 4 に示す．提案手法を実現するために，DBMS のログ書き込みをフックして統合されたログファイルに書き込むようにログリクエストを制御する．加えて，統合されたログファイルからリカバリ処理を行えるようにログファイルの再構築を行う．本章ではこれらの設計について解説する．

表 1　ログを統合する際に付加する情報

付加する情報	説明
VM 名	ログの発行元 VM の名前
ログファイル識別子	オリジナルのログファイル番号
ログファイルオフセット	オリジナルのログファイル内の位置

4.1 WAL のシーケンシャルアクセス化

4.1.1 ログファイル

複数の DBMS に対してそれぞれ存在していたログファイルを提案手法では 1 つに統合する．ログファイルを統合することで異なるファイルへの書き込みでは実現できなかったシーケンシャルなログ書き込みが実現可能になる．統合されたログファイルは管理 VM のファイルシステム上に作成し，各 DBMS から管理 VM に集められたログリクエストをシーケンシャルに書き込んでいく．統合されたログファイルには複数の DBMS のログが書き込まれるため，各ログにはどの VM の DBMS から書き込まれたか判別するための情報などを付加してファイルに書き込む．ログに付加する情報を表 1 に示す．オリジナルのログファイルというのは DBMS が本来書き込む予定だった VM 内のログファイルのことを示す．表 1 の情報を元に，後述するログファイルの再構築は行われる．

4.1.2 ログリクエストの制御

ハイパーバイザでは VM のディスクイメージへの書き込み先アドレスなどの抽象化された情報しか得られないため，ログ先行書き込みによるリクエストを判別できない．従来の DBMS のログファイルへの書き込みは VM のファイルシステムと I/O の仮想化機構を通してディスクイメージへの書き込みリクエストに変換されて管理 VM に渡され書き込まれる．しかしながら，統合されたログファイルへ DBMS のログを書き込むためには，ログ先行書き込みを判別し，書き込み先を制御する必要がある．

提案手法では DBMS のログ書き込みを通常の I/O とは異なる独自のパスを通して管理 VM へ送信する．具体的には DBMS のログ書き込みを hook し，VM 間通信を利用してログ書き込みを管理 VM へ送信するように DBMS に変更を加える．また，管理 VM では VM 間通信を通して集められた DBMS のログ書き込みを制御し，統合されたログファイルへシーケンシャルにログ書き込みを行う．

VM 間の通信を行うために VM と管理 VM に Frontend-Backend driver を導入する．Frontend-Backend driver は VM ごとに作成され，VM 側に Frontend driver を配置し，管理 VM 側に Backend driver を設置して VM 間の通信を行う．Frontend driver は DBMS からログ書き込みリクエストを受け取り，Bakcend driver へ送信する．

管理 VM に集められたログ書き込みリクエストを制御するためにログ統合プロセスを動作させる．ログ統合プロセスはマシンに 1 つ動作し，各 VM の Backend driver からログリクエストを集めて，統合されたログファイルへシーケンシャルに書き込む．統合ログファイルにログを書き込む際には表 1 の情報をログと一緒に書き込んでいく．

提案手法におけるログ書き込みは以下の手順で行われる．VM 上の DBMS でトランザクションがコミットされるとログ書き込み関数が hook され，追加したシステムコールを通して Frontend driver へログリクエストが渡される．Frontend driver は Backend driver へリクエストを送信する．管理 VM 上のログ統合プロセスは Backend driver からリクエストを集約し，統合されたログファイルへシーケンシャルに書き込む．

4.1.3 最適化

ログ統合プロセスには処理を効率化するためにリクエストの Polling と Delayed-Write の 2 つの機構を導入する．リクエストの Polling ではログ統合プロセスが Backend driver からリクエストを受け取るキューを Polling し，リクエストを受け取ったらすぐに処理を行う．デフォルトの動作ではログ統合プロセスは sleep で待機し，Backend driver からリクエストを受け取ったら sleep から復帰する．しかしながら，複数の DBMS が頻繁にログ書き込みを発生させるため，統合する DBMS が増えるほどログ統合プロセスの sleep-wakeup が頻繁に発生することが考えられる．結果として sleep-wakeup のオーバヘッドが大きくなってしまう．リクエストを Polling することでログ統合プロセスが処理をすぐに開始することが可能になる．

Delayed-Write では統合プロセスにおいてリクエストの処理を開始する前に一定時間待機することで，ログ統合プロセスが Backend からリクエストを受け取るキューにリクエストを溜めて，複数のリクエストをまとめて統合ログファイルに書き込むことでディスク書き込みを効率化する．ログ統合プロセスでは Backend から受け取ったリクエストを統合ログファイルに書き込んでいく．このとき write()+fsync() でファイルへの書き込みを行う．なるべく多くのリクエストをまとめて fsync() した方が書き込みが効率的に行えるため，キュー内のリクエストを全て write() した後，一度に fsync() している．しかしながら，キュー内にリクエストが近いタイミングに到着することは難しく，結果として，1 つのリクエストを write() しただけで fsync() して場合もある．そこで，1 つめのリクエストを受け取った後に待機する時間を作り，リクエストの到着を待つ．待機時間を長めに取ればより多くのリクエストをまとめて処理できるが，latency が劣化する可能性があるため待機時間は調整する必要がある．

DBMS のログ先行書き込みにはログの書き込みの他にチェックポイントという処理がある．チェックポイント処理は DBMS のログに書き込まれている変更を DB データファイルに反映する処理である．チェックポイント済みのデータはログファイルからリカバリする必要がなくなる．

提案手法では統合ログファイルとは別のチェックポイントファイルを VM ごとに作成し，ログ書き込みリクエストと同様に Frontend と Backend を通してログ統合プロセスによって書き込みが行われる．

4.2 ログファイルの再構築

提案手法では複数のログファイルを統合するため，リカバリ処理を工夫する必要がある．方法としては，統合されたログファイルから VM ごとにログを抽出してオリジナルのログファイルを再構築し，再構築したログファイルから DBMS のリカバリ処理を行う．再構築したログファイルは提案手法を施す前のログファイルと同じデータを持つため，従来の DBMS のリカバリ機構を再利用できる．ログの再構築処理では統合されたログファイルを先頭から読んでいき，どの VM の DBMS のリクエストかを判別し，VM ごとに新しいログファイルに書き出していく．

統合されたログファイルに書き込まれているログにはログを再構築するための情報として表 1 の情報が付加されているため，その情報を元にログを再構築していく．また，チェックポイントの情報もチェックポイントファイルから読み取り，再構築されたログファイルに反映する．管理 VM で再構築したログファイルを各 VM に配布することで DBMS はリカバリが可能になる．

5. Implementation

仮想化環境に Xen 4.2.1，Linux Kernel 3.4，DBMS に MySQL 5.6.17 を対象に提案手法のプロトタイプを実装した．管理 VM と VM の通信には Xen [7] の共有ページを利用したリング通信を使用している．ログ統合プロセスは管理 VM の Kernel スレッドとして実装している．管理 VM の linux kernel に 3529 行，VM linux kernel に 1525 行，MySQL に 220 行のコード変更及び追加を行っている．

6. Experiments

6.1 Experimental setup

2 台の同じスペックの物理マシンを使用し，一方を DBMS 統合環境サーバ，もう一方をワークロードを発行するクライアントとして使用する．使用した物理マシンは，CPU が Intel Xeon E3-1270 V2（4 コア，ハイパースレッディングオフ），メモリ 16GB である．ストレージはハードディスクまたは SSD を使用する．各 VM は VCPU 1 コア，仮想メモリ 1GB に設定し，VM 上で動作する MySQL のバッファプールは 700MB，その他設定はでデフォルトのままにしている．

6.2 MicroBenchmark

提案手法における DBMS の書き込みスループットを評価するために，insert のみを行うワークロードを実行し完了時間を計測し比較する．ワークロードには 2 つの integer のデータを insert して commit する処理を 10000 回繰り返すものを使用する．ワークロードを同時実行する DBMS のインスタンス数を変更しつつ，提案手法の有無で平均完了時間を比較する．比較対象は提案手法を適用しない vanilla，提案手法のログ統合処理を適用した proposal，提案手法のログ統合プロセスにてリクエストの pollig を行う proposal+polling，提案手法のログ統合プロセスで delayed-write を行う proposal+delay，polling と delayed-write の両方を適用した proposal+polling+delay として比較を行う．delayed-write における delay の挿入時間は事前の実験にて性能が向上しやすい時間を調査しており，それを使用している．ストレージに HDD と SSD のそれぞれを利用した際の実験を行っており，HDD を使用した環境では DBMS インスタンス数 (VM 数) を 1〜4，SSD を使用した環境では DBMS インスタンス数を 1〜8 に変更して実験する．HDD を使用した環境では DBMS インスタンス数が多いとディスクの競合で vanilla の性能が大きく下がってしまうため 4 インスタンスまでとしている．

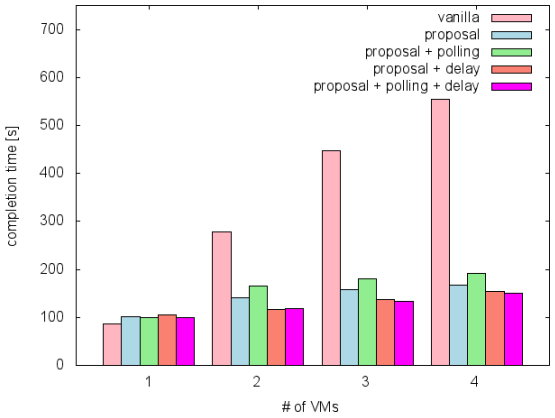

図 5 HDD 環境における DBMS 統合時の書き込み性能 (Lower is better)

HDD を使用した環境の実験結果を図 5 に示す．実験結果のグラフは横軸に DBMS 統合数，縦軸にワークロード完了時間を示しており，実行時間が短いほうがよい結果となっている．HDD を使用した環境では DBMS 統合数が増えるほど提案手法と vanilla の性能差が広がり，4 つの DBMS で並列実行した場合には最大 72%実行時間を削減できている．vanilla に比べて，提案手法では DBMS 統合数が増えた際の実行時間の変動が小さいのはシーケンシャルにログを処理できているためである．vanilla では DBMS 統合数が増えるとアクセスするログファイルが増えるためよりランダム書き込みが発生しやすくなり実行時間が伸びている．提案手法の効率化の工夫の中でも polling はあまり効果を発揮しなかった．対照的に delayed-write の効果があり，proposal に対して最大で 8%実行時間を削減した．DBMS 統合数が 1 のときは提案手法のオーバヘッドが出

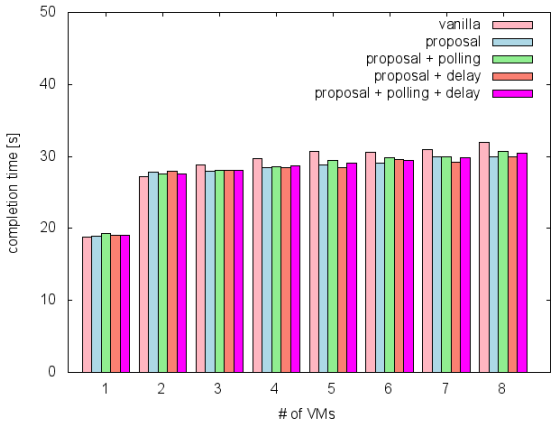

図 6 SSD 環境における DBMS 統合時の書き込み性能 (Lower is better)

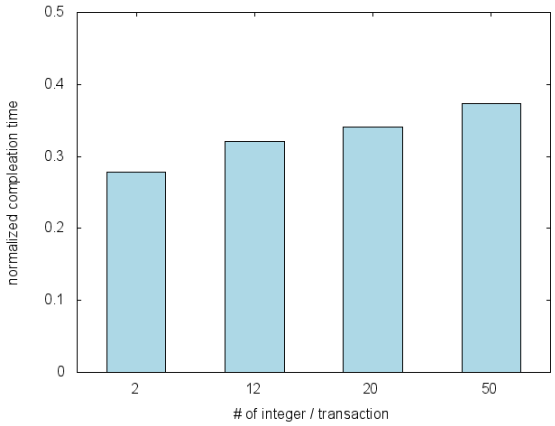

図 7 HDD 環境におけるトランザクションサイズに対する提案手法と vanilla の書き込み性能差 (Lower is better)

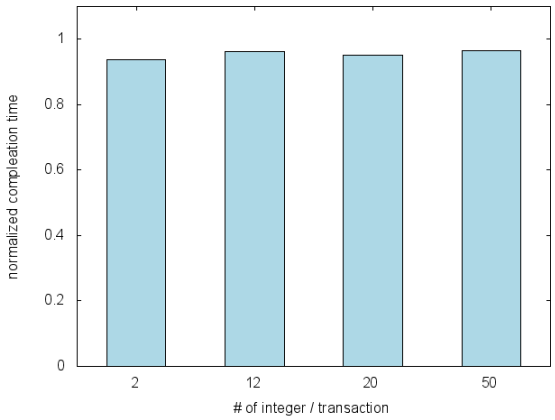

図 8 SSD 環境におけるトランザクションサイズに対する提案手法と vanilla の書き込み性能差 (Lower is better)

ており，vanilla に対して proposal は 15%実行時間が長くなっている．これは提案手法ではログ統合プロセスが I/O 実行パスに含まれるため，vanilla に対して 1 ステップ多く I/O 実行までかかってしまうことや，vanilla では Xen の qemu backend が効率的に I/O を処理できたことによる実行時間の差と考えている．

SSD を使用した環境の実験結果を図 6 に示す．SSD を使用した環境でも DBMS 統合数が増えるほど提案手法の有効性が出ており，8 つの DBMS を統合した環境では最大で 7%実行時間を削減している．また，統合数が 1 のときのオーバヘッドは最大で 2%であった．

トランザクションのサイズによる性能差を調べるための実験を行う．1 つのトランザクションのサイズが大きくなるとログ先行書き込みで書き込むデータ量も多くなる．DBMS 統合環境ではログのデータ量が増えるとログの書き込みにかかる時間が増え，複数のログファイルをシークするためにかかっていた時間の割合が相対的に小さくなる．逆にログのデータ量が小さいと，1 つのログの書き込み時間に対して別のログファイルに書き込むためのシーク時間の割合が増え，結果として DBMS 統合環境の各 DBMS のトランザクション書き込み性能が劣化しやすくなる．実験方法としてはワークロードのトランザクションサイズを変えて，提案手法と vanilla の性能差を比較する．ワークロードは 1 トランザクションで書き込む integer のデータを 2/12/20/50 と変更して，それぞれ 10000 回繰り返す．提案手法と vanilla での各トランザクションサイズにおける実行時間の差を調査して比較する．提案手法では polling と delayed-write のどちらも有効にしたものを使用する．ストレージに HDD と SSD をそれぞれ使用して実験を行い，DBMS 統合数は HDD 環境では 4，SSD 環境では 8 としている．

HDD 環境での実験結果を図 7 に示す．実験結果は横軸にトランザクションサイズ，縦軸に提案手法と vanilla の性能差を示している．実行時間で比較しているため小さい値の方が良い結果である．HDD を使用した環境ではトランザクションサイズが小さい方が提案手法の効果が大きい．vanilla はトランザクションサイズが小さい方が DBMS 統合時に性能が劣化しやすく，それに対して提案手法はシーケンシャルにログを書き込むことで改善できるためである．

SSD 環境での実験結果を図 8 に示す．SSD を使用した環境では HDD 環境よりも顕著では無いがトランザクションサイズが最も小さいときに性能差が最大となっている．

6.3 MacroBenchmark

実環境に近いベンチマークとして TPC-C [6] と TPC-H [8] を使用して評価実験を行った．TPC-C は電子商取引を模したベンチマークで DB の read と write がミックスされたトランザクションを発生させる．一定時間繰り返しクエリを発行し，1 分間あたりのトランザクションスループットを性能の指標とする．TPC-C を実行する DBMS の統合数を変更しつつ，提案手法の有無による平均スループットを確認する．比較する実験設定は microbenchmark と同様である．ストレージに HDD または SSD を使用して，それぞれ実験を行う．HDD 環境では DBMS 統合数を 1〜4,

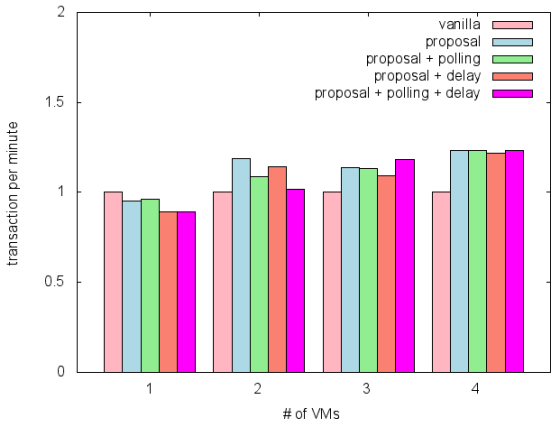

図 9 HDD 環境における TPC-C のスループット (Higher is better)

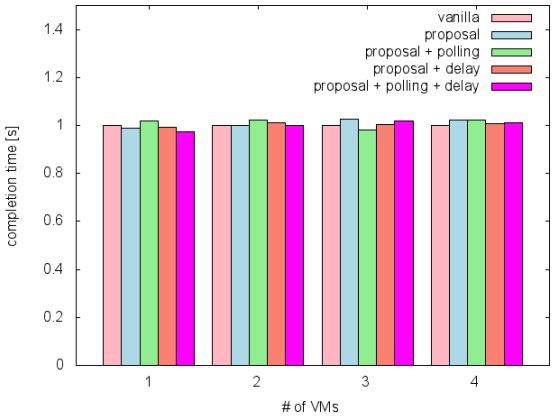

図 11 HDD 環境における TPC-H クエリ 2 の完了時間 (Lower is better)

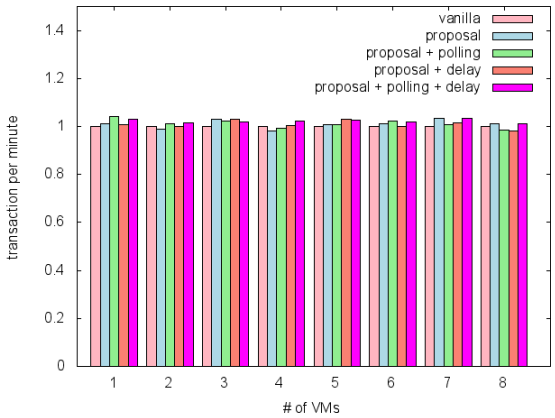

図 10 SSD 環境における TPC-C のスループット (Higher is better)

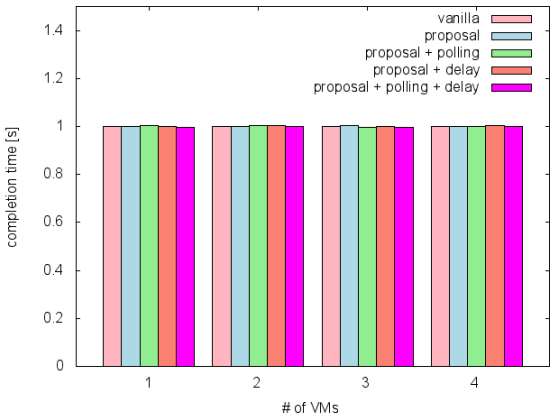

図 12 HDD 環境における TPC-H クエリ 20 の完了時間 (Lower is better)

SSD 環境では DBMS 統合数を 1〜8 で変化させ実験する．

HDD 環境での実験結果を図 9 に示す．実験結果は横軸が DBMS 統合数で縦軸が vanilla のスループットで標準化したスループットである．スループットなので高い値の方が良い結果である．HDD 環境の結果では提案手法の方が DBMS 統合数が 4 のときに最大で 23%スループットが向上している．TPC-C は DBMS が read と write をミックスして発行するワークロードであり，VM ディスクイメージにある DB データにも I/O が発行する．よって，提案手法でログ書き込みをシーケンシャルに処理しようとしても DB read が発生するとシーケンシャルに処理できない．全体の read と write の I/O 量は 4：6 程度であるが提案手法の効果が出ている．read と write がミックスされたワークロードでも提案手法の有効性が確認できた．

SSD 環境での実験結果を図 10 に示す．SSD 環境では DBMS 統合数が増えると提案手法の有効性が確認でき，最大で 3%スループットが向上している．

TPC-H は集計処理などの読み込みのみを発生させるトランザクションを実行する．いくつかのクエリが用意されており，今回の実験では CPU intensive なクエリ 2 と I/O intensive なクエリ 20 を使用する．DBMS で write が発生しない限りログ先行書き込みも発生しないため，提案手法は read のみのトランザクション性能に直接関与しない．提案手法によるオーバヘッドが発生していないかを調査する．実験方法としては，TPC-H クエリ 2，20 をそれぞれ DBMS の統合数を変更しつつ実行し平均実行完了時間を調べ，提案手法の有無による性能変化があるか確認する．比較する実験設定は microbenchmark と同様である．ストレージに HDD または SSD を使用して，それぞれ実験を行う．HDD 環境では DBMS 統合数を 1〜4，SSD 環境では DBMS 統合数を 1〜8 で変化させ実験する．

実験結果を図 11，図 12，図 13，図 14 に示す．実験結果は横軸が DBMS 統合数であり，縦軸が vanilla のクエリ完了時間で標準化した完了時間である．図 13 の SSD 環境でのクエリ 2 の実験結果では DBMS 統合数が増えると polling の性能劣化が大きくなっている．これは提案手法の polling のために 1 つの CPU の実行時間を消費してしまっていることに起因する．その他の実験結果には大きな変化は確認できないため，提案手法が read トランザクションに及ぼすオーバヘッドは問題にならないと考えられる．

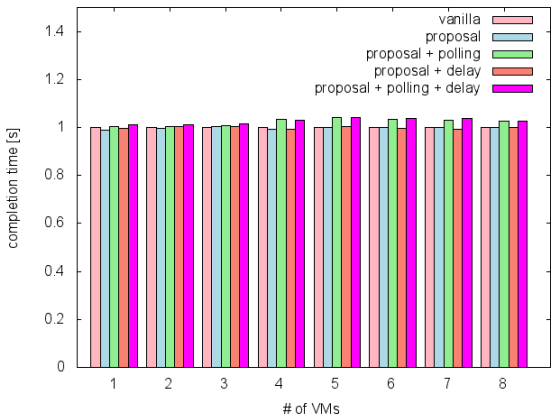

図 13 SSD 環境における TPC-H クエリ 2 の完了時間 (Lower is better)

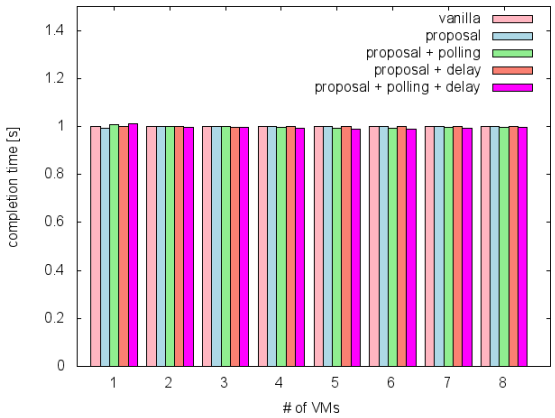

図 14 SSD 環境における TPC-H クエリ 20 の完了時間 (Lower is better)

7. Related work

本章では，関連する研究について紹介する．

DBMS 統合を実現する方法としては，本研究の対象としている VM を利用した方法以外にもマシンの全リソースを共有して 1 つの DBMS インスタンスで行う方法などがある．それらの方法を比較した研究 [9] やリソースを共有した環境における DBMS 統合手法を提案している研究 [10] [11] があり，新しい IaaS の実現方法を提案している．しかしながら，実際には多くの IaaS は VM を利用して実現されており，例として Amazon RDS [1] では VM 上で動作する DBMS をサービスとして提供している．本研究はより広く利用されている VM を用いた DBMS 統合環境を対象にしている．

Virtualization Design Advisor [12]，*SmartSLA* [13] では仮想化技術を利用した DBMS 統合環境でのリソース分配の自動設定手法を提案している．これらの研究は必要最小限のリソースを VM に設定することで DBMS の統合数を増やそうとしている．本研究は DBMS 統合環境のスループットを向上する研究であり競合する部分がない，よって協調できる立ち位置にある．

高い Availability は多くのサービスで求められることだが，性能とのトレードオフの関係にある．仮想化環境における性能を維持しつつ Availability を高めることを目標にした研究として，*PipeCloud* [14] や *RemusDB* [15] が存在する．本来のサービスへ与える影響を少なくして，Availability を高めるための取り組みが行われている．これらの研究は VM を利用したクラウドサービスを想定しているため，DBMS 統合環境を実現する場合は，本研究と組み合わせることで信頼性と性能を同時に向上できる．

WAL の scalability に関する研究に *Aether* [16] がある．マルチスレッドで動作させた際の WAL のボトルネックを分析し，scalability を向上させるための変更を加えている．DBMS 単体が対象で主に WAL のメモリ周りの処理に手を加えているため，本研究とは協調することが可能である．

高速なストレージデバイスとして Non-Volatile Memory(NVRAM) が注目されている．その NVRAM 上で DBMS の WAL を行う研究として NVWAL [17] がある．NVRAM 上で高速な WAL の書き込みを行い，DBMS の性能を向上させることができる．本研究と同じく WAL を関連しているが本研究は HDD，SSD を対象にしているため競合しない．

8. Conclusion

本研究では DBMS 統合環境においてトランザクション書き込みの高速化が効化されてしまう問題に取り組んだ．複数の DBMS からのトランザクション書き込みを制御して，DBMS のスループットを向上させる手法を提案する．ハイパーバイザで複数 DBMS のログ先行書き込みを捕捉し，シーケンシャルにディスクへ書き込む．DBMS ごとに存在していたログファイルをマシンに 1 つに統合してシーケンシャルな書き込みを実現可能にし，DBMS のログ書き込み関数を hook することでログ書き込みの検出を行い，統合されたログファイルにシーケンシャルに書き込むように制御した．プロトタイプを実装し，評価実験を行った．結果，insert のみを繰り返し行う micro benchmark では 72%性能向上し，TPC-C では 23%スループットが向上しており，提案手法の有効性が確認できた．

9. Future Work

提案手法におけるリカバリ処理の実装が完了していないため，ログの再構築によるリカバリ処理のオーバヘッドが評価できていない．統合ログファイルからの DBMS ごとのログ再構築機構の実装後，ログの再構築時間を含めた DBMS のリカバリかかる時間を評価する．今回は TPC-C，TPC-H をベンチマークとして使用したが，他のトランザクションの内容における提案手法の有効性を確認するために異なるベンチマークでの性能評価を行う．

参考文献

[1] Amazon Web Services, Inc: Amazon RDS, http://aws.amazon.com/jp/rds/.
[2] Oracle Corporation: MySQL, http://www.mysql.com/.
[3] PostgreSQL Global Development Group: PostgreSQL, https://www.postgresql.org/.
[4] Microsoft: SQL Server, https://www.microsoft.com/en-us/cloud-platform/sql-server.
[5] Oracle: Oracle Database, https://www.oracle.com/database/index.html.
[6] Transaction Processing Performance Council: TPC-C, http://www.tpc.org/tpcc/.
[7] Barham, P., Dragovic, B., Fraser, K., Steven Hand, T. H., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A.: Xen and Art of Virtualization, *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, pp. 164–177 (2003).
[8] Transaction Processing Performance Council: TPC-H, http://www.tpc.org/tpch/.
[9] Zhang, N., Tatemura, J., Patel, J. and Hacigumus, H.: Rethingking Designs for Managing Multi-Tenant OLTP Workloads on SSD-based I/O Subsystems, *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD '14)*, pp. 1383–1394 (2014).
[10] Curino, C., Jones, E. P., Madden, S. and Balakrishnan, H.: Workload-aware database monitoring and consolidation, *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data (SIGMOD '11)*, pp. 313–324 (2011).
[11] Narasayya, V., Das, S., Syamala, M., Chandramouli, B. and Chaudhuri, S.: SQLVM: Performance Isolation in Multi-Tenant Relational Database-as-a-Service, *Proceedings of the 6th Biennial Conference on Innovative Data Systems Research (CIDR '13)*, pp. 1–9 (2013).
[12] Soror, A. A., Minhas, U. F., Aboulnaga, A., Salem, K., Kokosielis, P. and Kamath, S.: Automatic Virtual Machine Configuration for Database Workloads, *Proceedings of the 2008 ACM SIGMOD International Conference on Management of data (SIGMOD '08)*, pp. 953–966 (2008).
[13] Xiong, P., Chi, Y., Zhu, S., Moon, H. J., Pu, C. and Hacigumus, H.: Intelligent Management of Virtualized Resources for Database Systems in Cloud Environment, *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering (ICDE '11)*, pp. 87–98 (2011).
[14] Wood, T., Lagar-Cavilla, H. A., Ramakrishnan, K. K., Shenoy, P. and der Merwe, J. V.: PipeCloud: Using Causality to Overcome Speed-of-Light Delays in Cloud-Based Disaster Recovery, *Proceedings of the 2nd ACM Symposium on Cloud Computing (SoCC '11)*, pp. 17:1–17:13 (2011).
[15] Minhas, U., Rajagopalan, S., Cully, B., Aboulnaga, A., Salem, K. and Warfield, A.: RemusDB: transparent high availability for database systems, *The VLDB Journal*, Vol. 22, No. 1, pp. 29–45 (2013).
[16] Johnson, R., Pandis, I., Stoica, R., Athanassoulis, M. and Ailamaki, A.: Aether: A Scalable Approach to Logging, *Proc. VLDB Endow.*, Vol. 3, No. 1-2, pp. 681–692 (2010).
[17] Kim, W.-H., Kim, J., Baek, W., Nam, B. and Won, Y.: NVWAL: Exploiting NVRAM in Write-Ahead Logging, *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '16)*, pp. 385–398 (2016).