Vol.2015-OS-134 No.8
2015/8/4

スケールアップマシンにおける DBMS インスタンス高速複製機構

福地 開帆[†1,a)]　山田 浩史[†1,b)]

概要：IaaS クラウドなどの仮想化技術が用いられている環境でデータベースを運用する際は，データベース管理システム (DBMS) を稼働させた仮想マシン (DBMS インスタンスと呼ぶ) を増減させ，負荷に応じた伸縮性を実現することができる．しかし，負荷の状況に応じて DBMS インスタンスを増加させることは困難である．DBMS の複製は DB ファイル等のコピーを要するため，複製した DBMS を起動するのに時間を要する．また，複製した直後の DBMS は DB データのキャッシュミスが頻発するため，複製した DBMS は起動直後に複製元と同等の性能を発揮できない．そのためユーザは，負荷を見積もり冗長な DBMS インスタンス数を決定・起動しロードバランサーを調整するといった無駄なコストや管理作業が必要となる．本研究の提案手法はスケールアップマシンを活用した負荷分散に特化した DBMS インスタンス高速複製手法で，同一マシン上で稼働している複製元の DBMS インスタンスのリソースを利用することで DBMS の効果的な複製を達成する．DBMS の複製を高速に作成し起動するために，最初に全てのデータを複製するのではなく，複製先で必要とされたタイミングで，複製元からオンデマンドに DBMS のデータファイルの複製を行う．さらに，複製直後の DBMS がすぐに性能を発揮できるようにするために，VM 間で DBMS のキャッシュを共有する．Xen 4.4.1 と MySQL 5.6.21，Linux 3.17.2 に対して実装し，実験により，読み込み負荷が高いワークロードでは複製の作成が 18 倍高速化・キャッシュウォーミングが 7.5 倍高速化，書き込み負荷が高いワークロードでは複製の作成が 12 倍高速化されることを確認した．

1. Introduction

IaaS クラウドなどの大規模データセンタ環境では，計算機資源を仮想化技術によって仮想化することで運用することが多い．Amazon EC2 や Google ComputeEngine などが代表的な例である．こうした環境では，ユーザの目的やサービスへの負荷に応じて仮想マシン (VM) を起動することができる．たとえば，ある VM を WEB サーバに，もう一つの VM を DB サーバとして起動できる．また，高負荷時には VM を追加で起動し負荷分散を行い，負荷が収まれば冗長な VM を停止するといったことが可能である．

こうした環境においてデータベースサービスを運用する際には，仮想マシン内でデータベース管理システム (DBMS) を稼働させることで，同様の恩恵を享受することができる．DBMS を稼働させた仮想マシン (DBMS インスタンスと呼ぶ) を増減させることで，負荷に応じた伸縮性を実現することができる．加えて，クラウドサービスプロバイダはデータベースサービスを柔軟に運用できるよう特別な機能を提供していることが多い．たとえば，Amazon RDS や Google CloudSQL などがある．Amazon RDS ではリードレプリカと呼ばれる，DBMS インスタンスを複製し読み込み専用の DBMS を起動させる機能を提供し，読み込みセンシティブなワークロードに対して容易にスケールアウトさせることを可能にしている．DBMS の複製は DBMS 自身が持つ機能によって行われ，一貫性の維持が保証される．

しかしながら，負荷の状況に応じて DBMS インスタンスを増加させることは困難である．DBMS の複製は DB ファイル等のコピーを要するため，複製した DBMS を起動するのに時間を要してしまう．また，複製した直後の DBMS は DB データのキャッシュミスが頻発するため，複製した DBMS が起動開始したとしても，その直後に複製元と同等の性能でリクエストに応答することができない．これらの要因のために，ユーザは，負荷の増加に対応するため冗長な DBMS インスタンスを起動しておき，DBMS へのリクエストを横流しして DBMS のデータキャッシュを準備しておく必要がある．結果として，リクエストをリダイレクトするためのロードバランサーの調整や，冗長な DBMS インスタンスの数を調整といった無駄なコストや

†1 現在，東京農工大学
Presently with Tokyo University of Agriculture and Technology
a) fukuchi@asg.cs.tuat.ac.jp
b) hiroshiy@cc.tuat.ac.jp

管理作業が発生してしまう．

本研究で提案する手法は近年登場しているスケールアップマシンを活用した負荷分散に特化した DBMS インスタンス高速複製手法である．スケールアップマシンとは，数 10 以上のコアや数 100 GB におよぶメモリ，複数ディスクを備えたマシンのことであり，その電力効率の観点から注目を浴びている [1]．実際，HP 社は 80 コア，4 TB，8 台の HDD を備えたマシンを販売しており，Amazon EC2 においても 32 個の仮想 CPU と 244GB のメモリ，644 GB の SSD ストレージを備えるインスタンスが提供されている．

提案手法では，同一マシン上で稼働している複製元の DBMS インスタンスのリソースを利用することで DBMS の効果的な複製を達成する．提案手法では，高速に DBMS インスタンスの複製を作成し起動する．そのために，DBMS の複製を高速に作成し起動するために，最初に全てのデータを複製・転送するのではなく，複製先で必要とされたタイミングで，複製元からオンデマンドに DBMS のデータファイルの複製・転送を行うアプローチをとる．これにより，全ての DBMS のデータファイルの転送が完了するのを待つこと無く高速に複製 DBMS を起動し負荷分散することが可能となる．そのために，初めに DBMS の起動に必要最小限のファイルだけを転送し，その後は OS と VMM が DBMS のディスクへのアクセスを監視し，アクセスされた領域が未転送であれば，VMM が複製元からその領域部分の複製・転送を行う．さらに，提案手法では複製直後の DBMS がすぐに性能を発揮できるようにするために．VM 間で DBMS のキャッシュを共有する．新たに立ち上げた DBMS が，ディスクからのデータ読み込みを行う際，もしも既に他の VM 上にそのデータに対応するキャッシュが存在すれば，ページテーブルを書き換えて物理メモリを共有し，ディスク I/O なしで応答を返せるようにする．これにより，高速なデータのキャッシングが可能となる．

本論文の貢献は以下のとおりである．

- DBMS インスタンスの高速複製手法を示した．提案手法は DBMS が有する複製作成機構を用いることができるという特徴をもつ．たとえば，DBMS の一貫性維持機構であったり，複製先に更新リクエスト内容を伝搬させることでマスター・スレーブによる負荷分散にも利用可能である．
- スケールアップマシン上で DBMS の高速複製を実現するための機構を示した．DBMS の起動開始までの時間を短縮するために DB ファイルの遅延コピーを，DBMS のキャッシュミスを抑制するために複製元の DBMS とのキャッシュ共有を実現する機構を示した．
- 提案手法を Xen 4.4.1 と MySQL 5.6.21，Linux 3.17.2 に対して実装し，実験により，読み込み負荷が高いワークロードでは複製の作成が 18 倍高速化・キャッシュウォーミングが 7.5 倍高速化されること，書き込み負荷が高いワークロードでは複製の作成が 12 倍高速化されることを確認し，提案手法がスケールアップマシンにおける負荷分散を目的とした DBMS インスタンスの高速複製に効果的であることを確認した．

本論文の構成は次のとおりである．2 章で背景について，3 章で提案について，4 章で設計について，5 章で実験について，6 章で関連研究について，7 章でまとめと今後の課題について述べる．

2. Background

2.1 DBMS's elasticity

多くの WEB サービスや，データの一貫性を保つ必要がある証券取引などで DBMS が利用されている．これら環境では DBMS へのリクエスト数が刻々と変化し，負荷上昇が起こる場合がある．例えば，ニューヨーク証券取引所では取引開始直後と終了直前の 10 分間はその他の時間に比べて取引量が増え，DBMS へのリクエスト数が増加する [2]．そのほか Twitter では，人気映画がテレビで放送された際や新年などに，ユーザーからの投稿が集中する．[3][4]．また，サービスを利用するユーザー数の増加によるワークロードの変化により，DBMS へのリクエスト数が増加する場合がある．DBMS の処理能力を超えるリクエストが発生するとサービスの応答性が低下するため，DBMS の処理能力を伸縮させ負荷上昇に対処する必要がある．

これらの負荷上昇に対処するため，DBMS は DBMS 自身がもつレプリケーション機能により自身の複製を作成し，負荷分散を行う．DBMS はデータの整合性を維持するため，複雑なデータ管理機構を持っている．DBMS 自身がもつレプリケーション機能を利用することで，データの整合性を保ちつつ複製を行うことができる．例えば，MySQL のレプリケーション機能は Facebook や Twitter や YouTube などで利用されている [5]

2.2 DBMS replication

2.2.1 DBMS-based replication

DBMS は自身がもつレプリケーション機能により自身の複製を作成し，負荷分散を行う．

DBMS のレプリケーションは主に 5 つのフェーズから構成される．(1) 複製元の DBMS にてスナップショットを作成する．(2) スナップショットを複製先マシンに転送する．(3) スナップショットを元に新たな DBMS を起動する．(4) 手順 2,3 の間に複製元 DBMS に行われた書き込みがあればそれを複製先 DBMS に非同期的に伝え，複製先 DBMS を最新の状態に追い付かせる．(5) ディスク上のデータをメモリ上の DBMS キャッシュに載せるキャッシュウォーミングが徐々に行われる．DBMS への接続が可能となり負荷分散が可能となるのは，手順 3 が完了した段階である．複製元と同じピーク性能の発揮が可能となるのは，手順 5

が完了した段階である．

手順 1 のスナップショット作成では，DBMS のある時点での完全なデータの複製ファイルを作成する．MySQL の場合，スナップショットの作成には MySQL が公式に提供する mysqldump やサードパーティ製の XtraBackup[6] などを用いる．スナップショットの容量は DBMS のデータベースファイルと同等かそれ以上となる．手順 2 のスナップショットの転送では，スナップショットをネットワーク経由で転送する．手順 3 の段階で複製 DBMS が起動し，負荷分散が可能となる．しかし，スナップショット作成中にも DBMS への書き込みが行われる場合があり，それら変更点はスナップショットには反映されないため，それら変更点を手順 4 にて MySQL が非同期的に複製先にも反映する．起動直後の MySQL のキャッシュは空であるが，MySQL がクライアントからの要求を処理するにつれてキャッシュにディスク上のデータが読み込まれていく．

複製作成後は，DBMS の一貫性維持機構により，複製元 DBMS と複製先 DBMS との一貫性が保たれる．複製元 DBMS への書き込みは DBMS の機能により非同期的に複製先 DBMS にも自動的に反映される．そのため，複製元 DBMS に行われた書き込み内容は，複製先 DBMS からも参照可能となる．これにより，更新と参照リクエストを受け付けるマスターと，それの複製であり参照のみを受け付けるスレーブを作成し，負荷分散するといったことが可能となる．マスターに対して更新リクエストが発生するワークロードであっても，一貫性維持機構により複製元 DBMS と複製先 DBMS での一貫性が保たれる．

2.2.1.1 Replication time

5 章の実験にてレプリケーションに要する時間を比較した．16GB のデータファイルをもつ MySQL の場合，読み込みのみが発生するワークロード環境では複製作成に 18 分以上を要した．内訳として，手順 1 のスナップショットの作成に 3 分，手順 2 のスナップショットの転送に 15 分，手順 3 の DBMS の起動に 5 秒を要した．書き込みが発生するワークロードでは，複製作成に 28 分以上を要した．内訳として，手順 1 のスナップショットの作成に 13 分，手順 2 のスナップショットの転送に 15 分，手順 3 の DBMS の起動に 5 秒を要した．

手順 1 に要する時間はデータベースのファイルサイズとワークロードの書き込み量に比例して増加する．また，スナップショットのファイルサイズはもとのデータベースのファイルサイズと同等かそれ以上となるため，手順 2 に要する時間もデータベースのファイルサイズに比例して増加する．大容量のデータを扱う DBMS ほど，レプリケーションに要する時間が増加する．

2.2.1.2 Poor performance of replica DBMSs

起動直後の DBMS はメモリ上のキャッシュが空のため，キャッシュミスが多発しディスクアクセスが必要となり，ピーク性能を発揮することができない．DBMS はディスクから読み込んだデータをメモリ上のキャッシュに保存するキャッシュウォーミングを行うことで，ディスクアクセスの回数を減らし性能向上を行なっている．しかし，起動直後はキャッシュは空であり，メモリよりも低速なディスクから逐次データを読み込む必要がある．大容量のデータを扱う DBMS ほど，キャッシュウォーミングに要する時間が増加する．

5 章の実験にてレプリケーション後にピーク性能を可能になるまでの時間を計測した．MySQL のキャッシュサイズが 15GB の場合，ピーク性能を発揮可能となるまで 30 分以上を要した．

2.2.2 Amazon RDS

Amazon RDS は Database-as-a-Service(DaaS) のひとつであり，DBMS をクラウド環境上で利用できる．クラウド環境上で動作するため，IaaS である Amazon EC2 などと同様に，負荷に応じて柔軟に DBMS インスタンスの追加や削除が行える．

Amazon RDS は DBMS の複製を作成する機能として，リードレプリカと呼ばれるものを提供している．これは，1 つの DBMS から複数の読み込み専用の複製 DBMS を作成する機能で，読み込み負荷の高いワークロードに対して効果的に負荷分散を行うことができる．

リードレプリカでの DBMS の複製も，DBMS 自身のレプリケーション機能により実現されている [7]．通常は 1 日 1 回手順 1 のスナップショット作成が自動的に行われ，それを元に複製が作成される．

2.2.3 VM fork

VM fork[8][9] は VM を高速に複製する手法である．VM fork では，fork システムコールのように VMM が VM 自体を fork する．必要最小限のメモリやレジスタの複製のみで新たな VM を起動し，メモリとディスクはオンデマンドに複製を行う．VM fork を利用することで，高速に DBMS インスタンスの複製を作成し，負荷分散を行うことが可能となる．

しかし，VM fork は VMM レイヤのアプローチであるため，DBMS が備えているレプリケーション機能を利用することができない．そのため，書き込みが発生するワークロードにて負荷分散を行うことができない．複製を作成し負荷分散を行なったあとで DBMS への書き込みが発生した場合，DBMS の機能による複製元 DBMS と複製先 DBMS との間の一貫性維持機構が利用できず一貫性が保たれない．負荷分散後に複製元 DBMS に書き込みが発生したとしても，複製先 DBMS にそれが伝搬されない．DBMS は自身で複雑な一貫性保持の仕組みをもっているため，レプリケーションには DBMS の機能を利用することが必要である．さらに，VM fork は短命で少数の複製 VM を作成することを想定しており，多くの複製 VM を作成すると複製

元の VM にディスクの遅延転送要求が集中し，複製元 VM のディスク I/O 性能が下がる場合がある．

2.3 Scale-up machines

近年はスケールアップマシンが登場し，IaaS クラウド環境でもそれらを利用可能となっている．スケールアップマシンとは，数 10 以上のコアや数 100 GB におよぶメモリ，複数ディスクを備えたマシンのことであり，その電力効率の観点から注目を浴びている [1]．実際，HP 社は 80 コア，4 TB，8 台の HDD を備えたマシンを販売しており，Amazon EC2 においても 32 個の仮想 CPU と 244GB のメモリ，644 GB の SSD ストレージを備えるインスタンスが提供されている．

スケールアップマシンを利用することは，マシンを多数活用するすることに比べ，使用電力量やサーバ統合率の点で優れている．Appuswamy らのスケールアップ環境とスケールアウト環境での Hadoop の性能を比較した実験では，スケールアップ環境がスケールアウト環境に比べ，使用電力量あたりの性能が最大で 4 倍高い性能を，ラック数あたりの性能が最大で 10 倍高い性能を示している．[1]

VM 内で DBMS を起動する DMBS インスタンスの場合も同様に，スケールアップマシンを利用することで使用電力量やサーバ統合率の利点を得ることができる．ただし，スケールアップマシン上で DBMS インスタンスを立ち上げる際に，負荷に柔軟に対応可能とするためには，DBMS インスタンス自体をスケールアップさせるのではなく，DBMS インスタンスをマシン内に複数起動する必要がある．VM に多くのメモリと仮想 CPU を割り当て，その上で DBMS を起動したとしても，DBMS への負荷は変動するため常にそれら資源を使い切るわけではなく資源利用の無駄が生じる．さらに，ワークロードを見積もり，どれほどのメモリや CPU を割り当てるのかを予測するといった複雑で困難な管理作業が必要となってしまう．そのため，小さな DBMS インスタンスを複数起動し，負荷に応じて柔軟に DBMS インスタンス数を変更することで無駄な資源利用の回避や複雑で困難なワークロードの見積もり処理の回避を行う必要がある．

3. Proposal

本研究では，スケールアップマシンにおける負荷分散を目的とした DBMS インスタンスの高速複製機構を提案する．本研究の目的は，(1) 高速に DBMS インスタンスの複製を作成・起動し，さらに (2) 複製後の DBMS がすぐにピーク性能を発揮できるようにすることである．本研究では，(1)DBMS の複製を高速に作成し起動するために，最初に全てのデータを複製・転送するのではなく，複製先で本当に必要とされたタイミングで，複製元からオンデマンドに DBMS のデータファイルの複製・転送を行うアプローチをとる．これにより，全ての DBMS のデータファイルの転送が完了するのを待つこと無く高速に複製 DBMS を起動し負荷分散することが可能となる．そのために，初めに DBMS の起動に必要最小限のファイルだけを転送し，その後は OS と VMM が DBMS のディスクへのアクセスを監視し，アクセスされた領域が未転送であれば，VMM が複製元からその領域部分の複製・転送を行う．さらに，(2) 複製直後の DBMS がすぐにピーク性能を発揮できるようにするために．VM 間で DBMS のキャッシュを共有する．新たに立ち上げた DBMS が，ディスクからのデータ読み込みを行う際，もしも既に他の VM 上にそのデータに対応するキャッシュが存在すれば，ページテーブルを書き換えて物理メモリを共有し，ディスク I/O なしで応答を返せるようにする．これにより，高速でキャッシュを暖めることが可能となる．スケールアップマシンの同一マシン上の VM であれば，VM 間でメモリ共有を行うことが可能である．本研究が対象とするワークロードは負荷が増減する realistic なものである．また，負荷分散を行うための DBMS インスタンスを高速に複製する機構であり，高可用性を実現するための DBMS インスタンス作成は対象外とする．

2.2 節で述べた DBMS の各複製手法と，提案手法による複製手法の比較を表 1 に示した．DBMS の機能による複製では，2.2.1.1 節で述べたように複製の作成に時間を要する．さらに，2.2.1.2 で述べたように複製した DBMS がピーク性能を発揮可能になるまでには時間を要する．そのため，負荷を見積もり予め DBMS インスタンスを立ち上げておきロードバランサの設定を行うなど，困難で複雑な管理作業が必要となる．これらは，DBMS の機能をそのまま利用して複製を作成する Amazon RDS も同様である．VM fork では，高速に VM の複製を作成できるため，DBMS インスタンスの複製は高速で行える．また，メモリもオンデマンドに複製されるためピーク性能発揮可能となるまでの時間も短い．そのため，予め負荷を見積もることは不要となり，高負荷になった際に高速に複製を作成すればよい．しかし，VMM レイヤのアプローチのため DBMS の機能を利用したレプリケーションが行えず，負荷分散後に DBMS への書き込みが起きた場合，複製元 DBMS と複製先 DBMS との間の一貫性維持機構が利用できず一貫性が保たれない．そのため，書き込みが発生するワークロードにて負荷分散を行えない．さらに，複数の複製を作成した場合，各複製 VM からの遅延転送要求により，複製元 VM に I/O 要求が集中してしまう．一方で，提案手法では，必要最小限のデータのみを複製し，残りは遅延的に複製するため高速に複製を作成できる．また，キャッシュを VM 間で共有するためピーク性能発揮までの時間も短い．よって，予め DBMS インスタンスを立ち上げておく必要はなくなり，負荷の見積もりなどの複雑で困難な管理作業

は不要となる．さらに，DBMS の機能を利用した複製作成を OS と VMM レイヤにて高速化するため，DBMS の機能を利用した複製が可能であり，書き込みが発生するワークロードでも負荷分散を行える．さらに，全てのデータの遅延転送が終わった場合，複製先 DBMS は複製元 DBMS に遅延転送要求を行うことなしで独立して動作可能となるため，複製元 VM への I/O 負荷の集中を避けられる．

4. Design

4.1 Lazy DBMS replication

4.1.1 Overview

レプリケーションを行う際，まずは必要最小限のデータのみを複製・転送し，新たな DBMS を起動する．その後，残りのデータを複製先で必要とされたタイミングで VMM がオンデマンドに複製する．通常の処理では，最初に全てのデータファイルを複製・転送してから DBMS の起動・負荷分散を行なっていたため，全てのデータファイルの複製・転送が終わるまでは DBMS を起動することが出来ず，負荷分散がすぐには行えなかった．しかし，必要最小限のデータのみを転送して新たな DBMS を起動し，残りは後から転送することで，データファイルが大容量であっても高速に複製の作成と起動を行うことができる．

最初に複製・転送する DBMS が起動するのに必要最小限のデータは，システム用のメタデータなどである．全ての DBMS のデータファイルを複製・転送せずとも，DBMS は起動可能である．

遅延的な複製は OS と VMM レイヤにて，DBMS には透過的に行う．DBMS に透過的に行うことで，複製先との整合性の維持といった DBMS 独自の機能を利用しつつ，高速な複製の作成と起動が可能となる．

ただし，複製先に全てのデータファイルの転送が完了する前に，複製元のデータが書き換えられてしまう場合がある．全ての領域の転送が終わる前に，複製元 DBMS に更新リクエストが行われた場合などである．複製先 DBMS には複製開始時のデータファイルが転送済みであるように見せる必要があり，複製先の DBMS に透過的に遅延的な複製・転送を行うためには，書き込みが起きていない複製開始時のデータを転送する必要がある．そこで，コピーオンライトにより，複製開始時のデータを保持し，データファイルへの書き込みは別ファイルに差分として行わせるようにする．DBMS はコピーオンライトにより作られた差分のファイルに対して読み書きを行い，遅延的な転送処理では元のデータを利用する．

さらに，コピーオンライトで元のデータファイルを保持したとしても，DBMS が実行中の場合，データファイルに常に最新のデータが書き込まれているとは限らない．DBMS は自身でキャッシュを持つため，メモリ上にのみ変更が反映され，まだディスク上のデータファイルにはフラッシュされてない場合がある．今回のように DBMS を起動したままで，必要最小限のデータを複製し DBMS を起動する場合，DBMS の起動時にクラッシュログからのクラッシュリカバリによるデータファイルの修復が必要となる場合がある．起動したままの複製であっても，クラッシュリカバリがあるため，メモリ上のデータのフラッシュ無しで必要最小限のデータファイルを複製し DBMS を起動してもデータが失われることはない．

レプリケーション開始時，まずは DBMS のデータファイルに対するコピーオンライトを有効にし，複製開始時の DBMS のデータファイルの状態を保存する．次に，DBMS の起動に必要最小限のデータファイルのみを作成し，複製・転送する．そしてそれを元に，新たな DBMS を起動する．この時点で，新たな DBMS が起動し，負荷分散が可能となる．その後，OS と VMM が複製先 DBMS のディスクアクセスを監視する．DBMS がまだ転送していないデータファイルの領域へのアクセスを試みた際は，VMM が複製元 VM から必要なデータを読み込み，それを利用して DBMS に処理を続けさせる．そのほか，バックグラウンドで複製元 VM から DBMS のデータファイルを複製先 VM に複製・転送し続ける．全てのデータファイルを複製し終えた時点で，複製先 DBMS は通常の DBMS と同じ独立した DBMS として動作可能となる．

概要図を図 1 に示す．複製作成時，まず DBMS の起動に必要最小限のデータファイルのみを複製し，それを元に DBMS を起動する．その後，OS と VMM が複製 DBMS のディスク I/O を監視し，未転送の領域へのアクセスの場合は VMM が複製元 VM のデータファイルの同一領域から読み込みを行い，その結果を返す．また，遅延的な転送は複製先 DBMS には透過的に行う必要があるため，複製元 VM には複製開始時のデータファイルをそのまま保存しておく必要がある．そのため，複製元 DBMS がデータファイルに書き込みを行う際は，コピーオンライト (COW) により元データファイルを保持したまま別ファイルに差分を書き込ませる．COW や遅延転送は DBMS には透過的に行われるため，これら書き込みは DBMS 自身の機能によって複製先 DBMS に非同期的に反映される．

4.1.2 VMM-level mechanism

VM 間で DBMS のデータファイルの各領域を転送するためのインタフェースを追加する．OS が，DBMS のディスクアクセスを監視し，アクセスを試みた領域が未転送だった場合は，OS がこのインタフェースを利用して複製元 VM から必要なデータを取得する．

インタフェースでは，VMM を経由して直接 VM 間のデータの転送を行う．通常の処理では，DBMS はネットワーク経由で VM 間のデータの転送を行う．しかし，今回は同一のスケールアップマシン内にある VM 間のレプリケーションを想定するため，ネットワークを利用せずに

表 1　負荷上昇への対応策の比較

	DBMS の複製に要する時間	複製 DBMS がピーク性能発揮可能となるのに要する時間	DBMS の機能を利用したレプリケーション	複製元 VM への I/O 負荷の集中
DBMS の機能による複製	長い	長い	可能	なし
Amazon RDS での複製	長い	長い	可能	なし
VM fork	短い	短い	不可	あり
提案手法	短い	短い	可能	なし

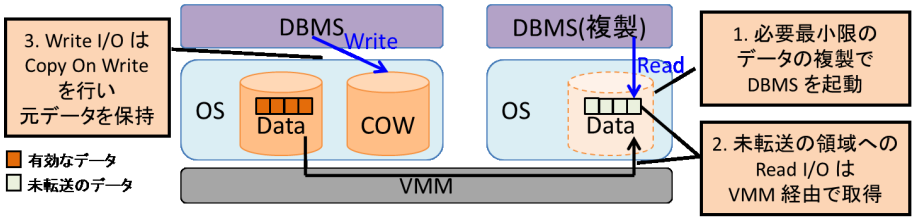

図 1　複製先 DBMS への遅延的なレプリケーション

VMM を経由した直接の VM 間のデータ転送が可能であり，ネットワークのプロトコルスタックによるオーバヘッドなしてデータ転送が可能である．

4.1.3　OS-level mechanism

DBMS 用の新たなディスクアクセスインタフェースと，データベースの各ファイルの各ブロックが転送済みかどうかのフラグを保持するテーブルを追加する．DBMS は，この新たに追加されたディスクアクセスインタフェースを用いてディスクへの読み書きを行う．インタフェースには，読み込んだデータの格納先バッファアドレスや読み書きの長さといった通常のファイルアクセス用のインタフェースと同様の情報を渡す．OS は，このインタフェースを利用して DBMS のディスクアクセスを監視する．OS は，DBMS がアクセスを試みたデータベースの各ブロックが転送済みかのフラグを確認し，もしも未転送であれば 4.1.2 節で追加した VMM のインタフェースを利用して複製元 VM から対応するデータを取得し，ローカルのデータファイルに書き込みを行い，転送済みフラグを立ててから通常の処理を続ける．

4.1.4　DBMS-level mechanism

4.1.3 節で追加したインタフェースを利用してディスクアクセスを行わせるようにする変更と，レプリケーション開始時のデータファイルを保存するために，データファイルへのコピーオンライトを行う機構の追加を行う．

DBMS のディスクアクセスを行う機構に，コピーオンライトの機構を新たに追加する．コピーオンライト有効時，DBMS はデータファイルへの書き込みを別の差分ファイルに対して行い，元のデータファイルには変更を加えないようにする．これにより，レプリケーション開始時のデータファイルが保存され，DBMS に透過的に遅延的な転送を行うことが可能となる．コピーオンライト終了時は，元のデータファイルと書き込みが反映された差分ファイルをマージする．

DBMS 内にコピーオンライト機構を組み込むことで，DBMS のみが持つ情報を利用した実装が可能となり，OS や VMM のレイヤにコピーオンライトを組み込む場合に比べて実装の最適化が容易となる．DBMS は自身でデータファイルへのフラッシュのタイミングなどを巧みに管理し，性能の向上をはかっている．コピーオンライトの機構に対しても，それらと同じ最適化を施すことが可能となる．

4.2　DBMS Cache Sharing

4.2.1　Overview

レプリケーションにより DBMS の複製を作成し起動完了後，VM 間で DBMS のメモリ上のキャッシュを共有し，起動直後にピーク性能を発揮可能とする．通常の処理では，起動直後の DBMS のメモリ上のキャッシュは空であるため，メモリより低速なディスクから逐次読み込む必要がある．しかし，VM 間でメモリ上の DBMS キャッシュを共有することで，複製元 VM 上の DBMS のキャッシュを利用して新たな DBMS のキャッシュを暖めることが可能となり，低速なディスクに対する I/O 処理を行うことなしで高速なキャッシュウォーミングを行える．

異なる VM 上の DBMS キャッシュであっても，同一の領域のキャッシュかどうかを判断する指標として，DBMS が自身のデータファイルを管理する際に用いるデータ ID を利用する．データ ID とは DBMS が自身でデータファイルの各領域を管理する際に用いる一意な値である．データファイルの各ブロックごとにデータ ID が一意につけられている．DBMS は，各データファイルを指定したサイズごとのブロックに分割し，ファイルごとにつけた ID とファイル内部のオフセットを組み合わせてデータ ID として利用している．OS がディスクを管理する際に用いるデバイス番号とセクタ番号のようなものであり，MySQL の場合，space と offset である．複製により同じデータファイルを扱う DBMS 同士の場合，異なる VM 間であっても，

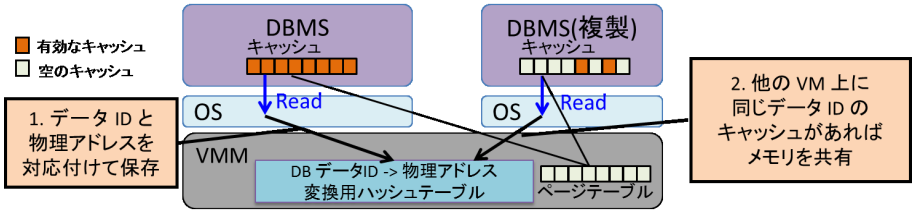

図 2　VM 間での DBMS キャッシュ共有

DBMS にとってのデータ ID が同一であれば，ファイルとファイル内のオフセットは同一となる．

DBMS がキャッシュにデータを読み込む際に，VMM 内のハッシュテーブルに，DBMS のデータ ID をキーとして，そのデータ ID に対応するキャッシュ領域のメモリの物理アドレスを要素として保存する．OS は，このハッシュテーブルとそれに対するインタフェースを利用して，各データ ID に対応するキャッシュが他 VM 上に存在するかを確認する．

OS が DBMS のディスクアクセスを監視し，DBMS が既に他の VM のメモリ上に存在する DBMS キャッシュのデータをディスクから読み込もうとした際に，VM 間でそのキャッシュが載っているメモリ領域を共有する．DBMS がデータ ID の情報とともにディスクアクセス要求を OS に行い，OS はそのデータ ID に対応するキャッシュが他 VM に存在するか確認し，存在すればメモリの共有を行う．共有中のメモリ領域は書き込み禁止とし，書き込みが発生した場合はページフォルトで検知しコピーオンライトにより共有を解除する．

概要図を図 2 に示す．まず，VMM と OS が DBMS のディスク I/O を監視する．Read I/O が発生した場合は，DBMS から渡されたデータ ID と Read I/O の結果を格納する DBMS キャッシュの物理メモリアドレスとを対応付けて VMM 内のハッシュテーブルに保存する．ハッシュテーブルはキーがデータ ID，値がキャッシュの物理メモリアドレスであり，キャッシュにあるデータ ID の要素が存在することはある VM 上の DBMS がそのデータ ID に対応するキャッシュを持っていることを示す．その後，複製先 DBMS インスタンスの DBMS が Read I/O を発行した際に，VMM は DBMS が読み込もうとしているデータ ID が他 DBMS のキャッシュに存在するかをハッシュテーブルを元に確認し，存在すればページテーブルを書き換えることで物理メモリを共有する．これにより，ディスクアクセスなしでキャッシュウォーミングを行える．

4.2.2　VMM-level mechanism

DBMS のデータ ID とそのキャッシュが乗っているメモリの物理アドレスを対応付けて保存するためのハッシュテーブルと，そのハッシュテーブルへの要素の追加・取得要求を受け付けるインタフェースと，指定した物理アドレス同士のメモリ共有開始要求を受け付けるインタフェースを追加する．

ハッシュテーブルには，DBMS のデータ ID をキーとして，そのキャッシュが載っているメモリの物理アドレスを要素として登録する．データ ID に対応する要素がハッシュテーブルに存在する場合，ある VM 上にデータ ID に対応するキャッシュが既に存在することになる．

ハッシュテーブルに要素を追加するインタフェースでは，DBMS のデータ ID とそれに対応する物理メモリアドレスを受け取る．そしてデータ ID をキー，物理メモリアドレスを要素としてハッシュテーブルに追加する．さらに，その物理メモリアドレスを書き込み禁止とする．書き込みが発生した場合，VMM がページフォルトを検知し，そのメモリアドレスを要素として持つハッシュテーブルのエントリを削除することで，キャッシュが破棄され別の用途として使われ始めた場合などの，メモリ内容が変化し物理メモリアドレスとデータ ID の対応付けが変化したことに対処する．

指定した物理アドレス同士のメモリ共有開始要求を受け付けるインタフェースでは，2 つの物理メモリアドレスを受け取る．1 つが共有元のキャッシュデータが入った物理メモリアドレス，もう 1 つの物理メモリアドレスが共有のために書き換えるべきページテーブルエントリの物理メモリアドレスである．同一のマシン内では，ページテーブルエントリを書き換えることで，ある物理アドレスが同一のメモリ (マシンアドレス) を指すようにすることができる．VMM がインタフェースを経由して OS からメモリの共有依頼を受け取ると，指定されたページテーブルエントリを書き換えてメモリの共有を行う．さらに，ページテーブルエントリのフラグを書き換えて該当アドレスを書き込み禁止とし，メモリに書き込みが起きた際はページフォルトハンドラ内で共有を解除する．

4.2.3　OS-level mechanism

DBMS からのディスクアクセスを受け付けるインタフェース内に，VM 間のメモリ共有を試みる処理を追加する．そのほか，4.1.3 節にて追加した DBMS 用のディスクアクセスインタフェースにて，通常のファイルアクセス用の情報だけでなく DBMS のデータ ID も受け取るようにする．インタフェース内部にて，DBMS が読み込もうとする領域が既に他の VM 上の DBMS キャッシュに存在するかをデータ ID を元に確認し，存在すれば VMM にメモリの

共有要求を行う．

OS がインタフェースで DBMS のディスクアクセス要求をデータ ID と共に受け取ると，4.2.1 節のハッシュテーブルからの要素の取得インタフェースを利用して他の VM に該当する領域のキャッシュが他 VM に存在するかを確認する．データ ID が同一であれば，同じ領域のデータをキャッシュしていることは 4.2.1 節で述べた通りである．他 VM 上に存在すれば，メモリ共有要求のインタフェースを利用して，ディスクから読み込んだデータの格納先のメモリと他 VM 上のメモリを共有する．存在しない場合，通常のディスク I/O 処理を行う．

4.2.4 DBMS-level mechanism

4.1.3 節で追加したインタフェースを利用してディスクアクセスを行わせるようにする変更を行う．DBMS はインタフェースを利用して，ディスクアクセス時に通常のファイルアクセス用の情報だけでなく，読み込もうとしている領域のデータ ID を OS に渡す．これにより，OS や VMM はそのデータ ID に対応する領域がすでに他 VM 上にキャッシュされているかを確認し，VM 間のメモリ共有を行うことができる．

5. Experiments

5.1 Experimental setup

プロトタイプの実装を Xen4.4.1，Linux 3.17.1，MySQL5.6.21 に対して行い実験を行った．

実験は，VM と DBMS を起動するサーバマシンとしてメモリ 256GB， AMD Opteron 6282 SE(3.0 GHz, 16 コア) プロセッサ，1TB 7200RPM HDD を 8 つ搭載したマシンを利用した．ベンチマークを動作させるクライアントマシンとして，メモリ 32GB，Intel Xeon E3-1286 v3 (3.70GHz, 4 コア) プロセッサを搭載したマシンを利用した．2 つのマシン間は 40GB イーサネットで接続した．Xen 4.4.1 を使用し，ホスト OS，ゲスト OS 共にカーネルは Linux 3.17.2 の 64bit 版を用いた．domU には 1 コアを割り当てた．

ベンチマークは SysBench[10] と TPC-C[11] を使用した．SysBench は継続的に負荷をかけ続け，毎秒のスループットを算出可能なものである．今回は読み込みのみのワークロードを用いた．TPC-C は倉庫の在庫管理のを模したオンライントランザクション処理ベンチマークであり，ディスクへの書き込み負荷が高いものである．DBMS には MySQL 5.6.21 を用いた．いずれの実験でも，VM のメモリサイズは 16GB，MySQL のバッファプールのサイズは 15GB，MySQL が扱うデータサイズは 17GB とした．MySQL のスナップショットの作成には XtraBackup を利用した．

5.2 Methodology

提案手法がワークロードの変化に柔軟に対応できることを確認するため実験を行った．ワークロードの負荷を変化させ，それに伴って MySQL の複製を作成し負荷分散を行なった際のベンチマークのスループットと VM のメモリ使用量を計測した．提案手法により，MySQL の複製を高速に作成できること，高速にキャッシュウォーミングを行えること，マシン全体での VM のメモリ使用量が少なくなることが期待される．また，提案手法が DBMS のレプリケーション機能を利用できていることを確認するため，書き込みが発生する書き込み負荷が高いワークロードでも実験を行った．

5.2.1 Read-intensive workload

まず 1 つの VM 上で MySQL を起動し，SysBench による負荷をかける．その後，3500 秒ごとに新たな VM の起動と MySQL の複製の作成・起動を行い，それらに対しても SysBench で新たに負荷をかける．最終的に合計 8 個の新たな VM と複製の MySQL を起動し，その間の SysBench のスループットを計測した．Amazon RDS のリードレプリカのように，読み込み負荷が大きな環境にて，複数の複製を作成し負荷を分散する環境を想定した．

5.2.2 Write-intensive workload

まず 1 つの VM 上で MySQL を起動し，TPC-C と SysBench による負荷をかける．その後，3500 秒ごとに新たな VM の起動と MySQL の複製の作成・起動を行い，それらに対しても SysBench で新たに負荷をかける．SysBench は TPC-C のデータセットに対して読み込みを行うようにした．最終的に合計 8 個の新たな VM と複製の MySQL を起動し，その間の SysBench のスループットを計測した．Amazon RDS のように，1 つの DBMS のみが書き込みを受け付け，その他は読み込みのみを受け付ける環境を想定した．

5.3 Results

5.3.1 Read-intensive workload

図 3 は複製元 MySQL に TPC-C で書き込み負荷をかけつつ，3500 秒ごとに，MySQL の複製を作成し負荷分散を行なった際の各 VM に対する SysBench のスループットの合計値である．default は提案手法なしの場合，lazy は遅延的な DBMS のレプリケーションを有効にした場合，share は DBMS キャッシュの VM 間共有を有効にした場合，lazy_share は遅延転送とキャッシュ共有の両者を有効にした場合である．

提案手法なしの default の場合，複製の作成と起動までに 18 分以上を要している．内訳として，完全なスナップショットの作成に 3 分，スナップショットの転送に 15 分以上を要した．これは，通常のレプリケーションでは完全なスナップショットを作成するため，データベースが扱うデータサイズに応じた大容量のスナップショットを作成・転送する必要があったためである．また，起動後に安定し

てピーク性能を発揮可能となるまでの時間は 4 分程度であった．短い時間で完了したのは，スナップショットの転送時にスナップショットが OS のキャッシュに載ったため，高速にキャッシュウォーミングを行えたためである．参考として，OS のページキャッシュが空の環境である最初に起動した MySQL のキャッシュウォーミングには 30 分以上を要した．スナップショットサイズが VM のメモリサイズを超過し，スナップショットが OS のページキャッシュに十分に載らない環境では，時間を要すると考えられる．

遅延的な複製を有効にした lazy の場合，複製の作成と起動までは 72 秒で終わり，default に比べ 15 倍高速化された．これは，必要最小限のデータのみを転送したことで，大容量の完全なスナップショットの転送を行うこと無く複製を作成し起動を行えたためである．また，時間の内訳として，VM 自体の起動に 54 秒，必要最小限のデータの作成と転送に 12 秒，MySQL の起動に 6 秒であった．一方で，起動後にピーク性能を発揮可能となるまでの時間は，30 分以上を要した．default と違い，完全なスナップショットを転送しないため，OS のページキャッシュも空であり，キャッシュウォーミングに時間を要した．

VM 間でのキャッシュ共有を有効にした share の場合，起動後にピーク性能を発揮可能となるまでの時間は 4 分程度であった．また起動直後からピーク性能の 67 %の性能を発揮可能であった．

遅延的な複製と VM 間キャッシュ共有を有効にした lazy_share では，複製の作成と起動までは 60 秒であった．内訳として，VM 自体の起動に 37 秒，必要最小限のデータの作成と転送に 16 秒，MySQL の起動に 7 秒であった．さらに安定してピーク性能を発揮可能となるまでの時間も 4 分程度であった．よって，提案手法により複製の作成と起動は 18 倍高速化され，提案手法の遅延的な複製作成によりスナップショットの作成・転送に要する時間を効果的に削減できたと言える．また，完全なスナップショットを作成・転送しないため複製先 VM のページキャッシュが空であり，本来は DBMS のキャッシュウォーミングに 30 分以上要するが，提案手法の VM 間 DBMS キャッシュ共有により 4 分以内で完了し，ピーク性能発揮可能までの時間は 7.5 倍高速化された．よって，提案手法による VM 間の DBMS キャッシュ共有は高速なキャッシュウォーミングに効果的であるといえる．

図 4 は実験中の VM へのメモリ割り当て量の合計の推移である．detalut と lazy では，VM 間のメモリ共有を行わないため DBMS インスタンスを起動するごとに 16GB ずつメモリ消費量が増えている．一方で，メモリ共有を行う share と lazy_share では，DBMS のキャッシュが複製元と複製先で共有されたことにより，新たに DBMS インウタンスを起動しても僅かなメモリ消費量で済んでいる．各 DBMS インスタンスでは 13.3GB が毎回共有され，VM の

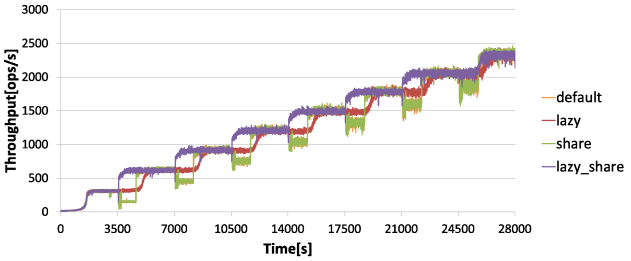

図 3　読み込み負荷の高いワークロードでのスループット

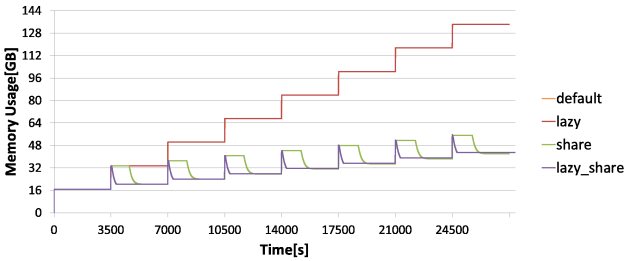

図 4　読み込み負荷の高いワークロードでの VM のメモリ使用量の合計の推移

メモリ使用量は 0.16 倍となった．よって，提案手法の VM 間での DBMS キャッシュの共有はメモリ資源利用の節約にも効果的であるといえる．

5.3.2 Write-intensive workload

図 5 は複製元 MySQL に TPC-C で書き込み負荷をかけつつ，3500 秒ごとに，MySQL の複製を作成し負荷分散を行なった際の各 VM に対する SysBench のスループットの合計値である．native は提案手法なしの場合，cow は遅延的な DBMS のレプリケーションを有効にした場合，ps は DBMS キャッシュの VM 間共有を有効にした場合，ps_cow は遅延転送とキャッシュ共有の両者を有効にした場合である．

提案手法なしの default の場合，複製の作成と起動までに 28 分以上を要した．内訳として，スナップショットの作成に約 13 分，転送に約 15 分を要した．また，複製作成直後，複製元の性能が不安定となっている．これは，スナップショット作成時には多量のディスク I/O が発行され，複製元 MySQL とそれが動作する VM の性能に影響をおよぼしたためである．

一方で，提案手法ありの lazy_share の場合，複製開始から 136 秒で複製 MySQL の起動が完了している．結果として，複製 MySQL の作成・起動が約 12 倍高速化された．なお，default・lazy・share・lazy_share の全ケースにて，OS のページキャッシュが空の環境である最初に起動した MySQL のキャッシュウォーミングに比べ，複製である 2 つめ以降の MySQL の方がキャッシュウォーミングが高速化されているが，これは 2 つ目以降の複製 MySQL には 1 つ目の MySQL と違い，書き込み負荷が全くかかっていないためである．

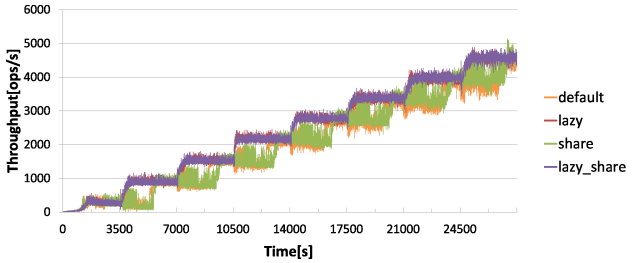

図 5　書き込み負荷の高いワークロードでのスループット

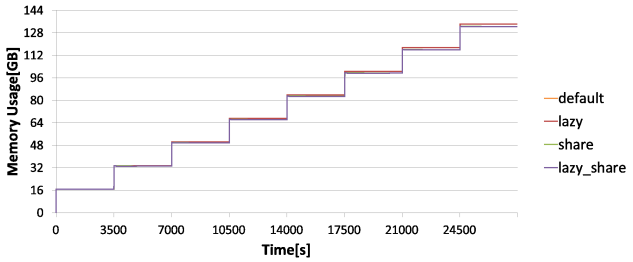

図 6　書き込み込み負荷の高いワークロードでの VM のメモリ使用量の合計の推移

図 6 は実験中の VM へのメモリ割り当て量の合計の推移である．提案手法の lazy_share では最終的に 1.4 %メモリ使用量が削減された．書き込み負荷が高いワークロードの場合，ページ共有の解除が発生する機会が多い．

また，書き込みが発生するワークロードであっても，提案手法を利用できている．これは，複製元 DBMS と複製先 DBMS との一貫性維持や，書き込み内容の伝搬といった，DBMS がもつレプリケーション機能を提案手法が利用できるためである．

6. Related Work

SnowFlock[8] や Kaleidoscope[9] は VM を高速に複製する VM fork を行う．VM fork では，fork システムコールのように VMM が VM 自体を fork する．必要最小限のメモリやレジスタの複製のみで新たな VM を起動し，メモリとディスクはオンデマンドに複製を行う．VM fork を利用することで，高速に VM と DBMS の複製を作成し，負荷分散を行うことが可能となる．しかし，DBMS の高速な複製を行う際，VMM レイヤのアプローチである VM Frok では，DBMS 自身がもつレプリケーション機能を利用することができない．一貫性維持などの複雑な DBMS の機能を利用するためには，DBMS 自身がもつレプリケーション機能を利用して複製を作ることが望ましい．一方で，本研究では DBMS のレプリケーション機能を利用しつつ，それを VMM が DBMS には透過的に高速化するため，DBMS 自身の機能を利用することができる．さらに，VM fork は短命で少数の複製 VM を作成することを想定しており，多くの複製 VM を作成すると複製元の VM にディスクの遅延転送要求が集中し，複製元 VM のディスク I/O 性能が下がる恐れがある．一方で，本研究ではバックグラウンドで DBMS の全テータを転送するため，転送完了後は DBMS は複製元とは独立して動作することが可能となる．そのため，多くの複製を作成しても，複製元に遅延転送要求が集中し続けることはない．

VM substrate[12] は VM の高速な起動を目的として，VM のスナップショットの複製による VM の起動と，ディスクの遅延転送を行う．新たに起動した VM からの遅延転送要求のみに基づいてディスクの遅延転送を行うと，多数の VM を起動した際に VM fork と同様に I/O 負荷の集中が発生するため，VM substrate ではバックグラウンドで全データの遅延転送を行い，I/O 負荷の集中が長期化することを避けている．VM substrate と本研究では目的が異なるが，本研究でも複製元 DBMS インスタンスへの I/O 負荷の集中の長期化を防ぐため VM substrate と同様にバックグラウンドで全データの転送を行った．

Satori[13] や XLH[14] は VM 間で OS のページキャッシュを共有する．しかし，DBMS は自身でキャッシュを管理するため，VM 上の多くのメモリを DBMS が用いている場合，OS のページキャッシュの共有のみでは高速なキャッシュウォーミングと資源利用の節約を実現できない．本研究では，DBMS のみがもつ情報を利用して，DBMS のキャッシュを VM 間で共有するため，高速なキャッシュウォーミングと資源利用の節約を実現できる．

CMD[15] も VM 間でメモリページ共有を行うが，特殊なハードウェアを必要とする．また，Difference Engine[16] も VM 間で同一内容のメモリを共有し，さらに圧縮することでメモリ資源利用を節約するが，DBMS は自身でキャッシュを管理するため，単にメモリを VM 間で共有するだけでは同一の DBMS キャッシュを共有することができず，高速なキャッシュウォーミングによる複製 DBMS のピーク性能発揮までの時間短縮を実現できない．

ShuttleDB[17] は，テナント環境の DaaS において，DBMS の資源利用量に応じて VMM による VM 自体の複製と DBMS の機能による DBMS の複製とを使い分けることで，高速に DBMS の複製を作成し DBMS テナントの柔軟性を提供する．しかし，複製後の DBMS がピーク性能発揮可能になるまでの時間を考慮していない．

RemusDB[18] は，Xen[19] の Remus[20] をベースとして DBMS の高可用性を実現する．DBMS インスタンスの複製を別のマシンに作成し予め待機させておき，障害発生時はすぐにその待機中の DBMS インスタンスを利用してサービスを継続できるようにする．しかし，複製の作成目的が高可用性の実現であり，Elasticity が目的ではない．DBMS の複製による高可用性を実現したい場合は RemusDB を，高速な複製作成による Elasticity を実現したい場合は本研究を用いればよい．

7. Conclusion

本研究では，スケールアップマシンにおける DBMS インスタンス高速複製機構を提案した．高速に DBMS の複製を作成し起動するために，DBMS の起動に必要最小限のファイルを複製し，残りは VMM が DBMS には透過的に遅延的に転送するアプローチをとった．さらに，複製直後の DBMS が即座にピーク性能を発揮可能とするため，VM 間でメモリを共有し，新たに起動した DBMS が他 VM 上の DBMS のキャッシュを利用して高速にキャッシュウォーミングを行えるようにするアプローチをとった．実験により，読み込み負荷が高いワークロードでは複製の作成が 18 倍高速化・キャッシュウォーミングが 7.5 倍高速化されること，書き込み負荷が高いワークロードでは複製の作成が 12 倍高速化されることを確認し，提案手法がスケールアップマシンにおける負荷分散を目的とした DBMS インスタンスの高速複製に効果的であることを確認した．

今後の課題として，さらなるメモリ使用量の削減を目的として，XLH[14] のようにコンテンツベースのメモリ共有を取り入れるかの検討があげられる．メモリの各ページ内容をスキャン・比較したのち同一のものがあればそれらを共有するコンテンツベースのメモリページ共有に対して，本研究ではその他の情報から確かに同一であると判断したメモリページ同士を共有するセマンティクスベースのメモリページ共有を行っている．書き込み量が多いワークロードの場合，たとえ DBMS のレプリケーション機能により後で同一内容のメモリページになったとしても，現在のセマンティクスベースの実装では書き込みが行われた際に共有を解除してしまい，その後ページ内容をスキャンすることはしないため，共有可能なページであるのに見逃してしまう．そのほか，KVS といった異なるアーキテクチャの DBMS に対してのプロトタイプ実装が挙げられる．

参考文献

[1] Appuswamy, R., Gkantsidis, C., Narayanan, D., Hodson, O. and Rowstron, A.: Scale-up vs Scale-out for Hadoop: Time to Rethink?, *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, New York, NY, USA, ACM, pp. 20:1–20:13 (online), DOI: 10.1145/2523616.2523629 (2013).

[2] Taft, R., Mansour, E., Serafini, M., Duggan, J., Elmore, A. J., Aboulnaga, A., Pavlo, A. and Stonebraker, M.: E-store: Fine-grained Elastic Partitioning for Distributed Transaction Processing Systems, *Proc. VLDB Endow.*, Vol. 8, No. 3, pp. 245–256 (online), DOI: 10.14778/2735508.2735514 (2014).

[3] Twitter: Celebrating a New Year with a New Tweet Record — Twitter Blogs, Twitter (online), available from ⟨https://blog.twitter.com/2011/celebrating-new-year-new-tweet-record⟩ (accessed 2015-07-03).

[4] Twitter: New Tweets per second record, and how! — Twitter Blogs, Twitter (online), available from ⟨https://blog.twitter.com/2013/new-tweets-per-second-record-and-how⟩ (accessed 2015-07-03).

[5] MySQL: MySQL Replication, MySQL (online), available from ⟨https://www.mysql.com/products/enterprise/replication.html⟩ (accessed 2015-07-03).

[6] Percona: XtraBackup, Percona (online), available from ⟨https://www.percona.com/software/percona-xtrabackup⟩ (accessed 2015-07-03).

[7] Amazon: Working with PostgreSQL and MySQL Read Replicas Amazon Relational Database Service, Amazon (online), available from ⟨http://docs.amazonaws.cn/en_us/AmazonRDS/latest/UserGuide/USER_ReadRepl.html⟩ (accessed 2015-07-03).

[8] Lagar-Cavilla, H. A., Whitney, J. A., Scannell, A. M., Patchin, P., Rumble, S. M., de Lara, E., Brudno, M. and Satyanarayanan, M.: SnowFlock: Rapid Virtual Machine Cloning for Cloud Computing, *Proceedings of the 4th ACM European Conference on Computer Systems*, EuroSys '09, New York, NY, USA, ACM, pp. 1–12 (online), DOI: 10.1145/1519065.1519067 (2009).

[9] Bryant, R., Tumanov, A., Irzak, O., Scannell, A., Joshi, K., Hiltunen, M., Lagar-Cavilla, A. and de Lara, E.: Kaleidoscope: Cloud Micro-elasticity via VM State Coloring, *Proceedings of the Sixth Conference on Computer Systems*, EuroSys '11, New York, NY, USA, ACM, pp. 273–286 (online), DOI: 10.1145/1966445.1966471 (2011).

[10] sysbench: SysBench, SysBench (online), available from ⟨https://github.com/akopytov/sysbench⟩ (accessed 2015-07-03).

[11] TPC: TPC-C, TPC (online), available from ⟨http://www.tpc.org/tpcc/⟩ (accessed 2015-07-03).

[12] Wang, K., Rao, J. and Xu, C.-Z.: Rethink the Virtual Machine Template, *Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '11, New York, NY, USA, ACM, pp. 39–50 (online), DOI: 10.1145/1952682.1952690 (2011).

[13] Miłós, G., Murray, D. G., Hand, S. and Fetterman, M. A.: Satori: Enlightened Page Sharing, *Proceedings of the 2009 Conference on USENIX Annual Technical Conference*, USENIX'09, Berkeley, CA, USA, USENIX Association, pp. 1–1 (online), available from ⟨http://dl.acm.org/citation.cfm?id=1855807.1855808⟩ (2009).

[14] Miller, K., Franz, F., Rittinghaus, M., Hillenbrand, M. and Bellosa, F.: XLH: More Effective Memory Deduplication Scanners Through Cross-layer Hints, *Proceedings of the 2013 USENIX Conference on Annual Technical Conference*, USENIX ATC'13, Berkeley, CA, USA, USENIX Association, pp. 279–290 (online), available from ⟨http://dl.acm.org/citation.cfm?id=2535461.2535495⟩ (2013).

[15] Chen, L., Wei, Z., Cui, Z., Chen, M., Pan, H. and Bao, Y.: CMD: Classification-based Memory Deduplication Through Page Access Characteristics, *Proceedings of the 10th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '14, New York, NY, USA, ACM, pp. 65–76 (online), DOI: 10.1145/2576195.2576204 (2014).

[16] Gupta, D., Lee, S., Vrable, M., Savage, S., Snoeren, A. C., Varghese, G., Voelker, G. M. and Vahdat, A.: Difference Engine: Harnessing Memory Redundancy in Virtual Machines, *Commun. ACM*, Vol. 53, No. 10, pp. 85–93 (online), DOI: 10.1145/1831407.1831429 (2010).

[17] Barker, S., Chi, Y., Hacigümüs, H., Shenoy, P. and Cecchet, E.: ShuttleDB: Database-Aware Elasticity in the Cloud, *11th International Conference on Autonomic Computing (ICAC 14)*, Philadelphia, PA, USENIX Association, pp. 33–43 (online), available from ⟨https://www.usenix.org/conference/icac14/technical-sessions/presentation/barker⟩ (2014).

[18] Minhas, U. F., 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 (online), DOI: 10.1007/s00778-012-0294-6 (2013).

[19] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A.: Xen and the Art of Virtualization, *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, New York, NY, USA, ACM, pp. 164–177 (online), DOI: 10.1145/945445.945462 (2003).

[20] Cully, B., Lefebvre, G., Meyer, D., Feeley, M., Hutchinson, N. and Warfield, A.: Remus: High Availability via Asynchronous Virtual Machine Replication, *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'08, Berkeley, CA, USA, USENIX Association, pp. 161–174 (online), available from ⟨http://dl.acm.org/citation.cfm?id=1387589.1387601⟩ (2008).