OverlayFSを用いたコンテナに対するサービス妨害攻撃の防止

佐藤 寛文[1]　光来 健一[1]

概要：近年，Docker などのコンテナ型仮想化が広く使われるようになってきている．Docker コンテナでは個別のディスクイメージを作成するために，読み込み専用のベースイメージの上に書き込み可能な差分イメージを重ねることができる．そのために，OverlayFS と呼ばれるファイルシステムがよく用いられているが，ファイルを書き換える際に下層にあるファイル全体が上層にコピーされるため，ファイルサイズが大きい場合にはオーバヘッドが大きい．このコピーオンライト機能を悪用されると，コンテナ内で意図的に巨大なファイルを書き換えることによりサービス妨害攻撃を引き起こされる可能性がある．本稿では，このような攻撃を防ぐための新しいファイルシステム TranslayFS を提案する．TranslayFS では，ファイルの書き換え時にスパースファイルと呼ばれる特殊なファイルを作成し，書き換え部分のみを上層に保持することでファイル全体を一括コピーしないようにする．ファイルの読み込みは，上層にデータが保存されていれば上層から，それ以外のデータについては下層から行う．我々は TranslayFS を Linux カーネルに実装し，ファイル書き換え時の遅延を大幅に削減できることを確認した．

キーワード：コンテナ、サービス妨害攻撃、ファイルシステム、スパースファイル

1. はじめに

近年，Docker[1] などのコンテナ型仮想化が広く使われるようになってきている．Docker コンテナのディスクイメージは，読み込み専用のベースイメージの上に書き込み可能な差分イメージを重ねることで作成される．この重ね合わせを実現するために，Docker ではストレージドライバとして OverlayFS[2] と呼ばれるファイルシステムがよく利用されている．OverlayFS は，読み込み時には下層のベースイメージと上層の差分イメージに存在するそれぞれのファイルに対してアクセスを行う一方，ファイルの作成や書き込みは上層の差分イメージに対してだけ行う．

OverlayFS では，下層のファイルを書き換える際にはまず，下層にあるファイル全体を上層にコピーする．そして，上層にコピーしたファイルに対して書き込みを行う．これはコピーオンライトと呼ばれる機能である．そのため，データベースのようにファイルサイズが大きい場合には，ファイルを初めて書き換える際のオーバヘッドが大きくなるという問題がある．この機能を悪用して外部から意図的にコピーオンライトを発生させられると，サービス妨害攻撃を引き起こされる危険性がある．巨大なファイルを書き換えさせることより，コピーが完了するまでコンテナを停止させることができる．また，下層の巨大なファイルを上層にコピーすることにより，システム全体のディスク容量を圧迫させることもできる。

コンテナに対するこのようなサービス妨害攻撃を防ぐために，本稿では OverlayFS をベースとする新しいファイルシステムである TranslayFS を提案する．TranslayFS では，下層にあるファイルの書き換え時にファイル全体のコピーを行う代わりに，書き換えた部分のみを上層に保持することでコピーオンライトのオーバヘッドを削減する．このファイルを読み込む際には，上層にデータが保存されている領域については上層からデータを読み込み，それ以外の領域については下層のファイルからデータを読み込む．上層に保持する書き込みデータを管理するために，TranslayFS は上層にスパースファイルと呼ばれる特殊なファイルを作成する．スパースファイルを用いることにより，断片的なデータを効率よく，かつ，容易に扱うことができる．

我々は TranslayFS を Linux カーネル 4.4.0 に実装し，OverlayFS との間でサイズの大きなファイルの読み書き性能を比較した．実験の結果，TranslayFS ではファイル書き換え時のオーバヘッドが大幅に削減されることが確認できた．一方，読み込み性能は OverlayFS より低下したが，そ

[1] 九州工業大学

© 2018 Information Processing Society of Japan

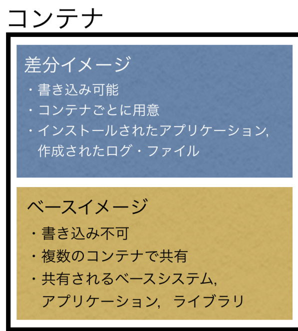

図 1 Docker コンテナ

の性能低下は 1.3%だけであった．また，Docker の他のストレージドライバとも比較を行い，どのドライバよりも性能がよいことを確認した．

以下, 2 章では OverlayFS を用いるコンテナに対するサービス妨害攻撃について述べる．3 章では新たなファイルシステムである TranslayFS を提案する．4 章では TranslayFS の実装について説明する．5 章では TranslayFS と他のストレージドライバの性能を比較した実験について述べる．6 章で関連研究について触れる．最後に 7 章で本稿をまとめる．

2. コンテナに対するサービス妨害攻撃

近年，Docker などのコンテナ型仮想化が広く使われるようになっている．従来のハイパーバイザ型の仮想化は OS を含む計算機全体を仮想化した仮想マシンを提供するのに対し，コンテナ型の仮想化は OS を含まないアプリケーションの仮想実行環境だけを提供する．そのため，コンテナは仮想マシンよりも高速に起動することができ、少ないリソースしか必要としないためより軽量に動作する．また，ディスクイメージも小さくすることができ，新しいコンテナの作成を高速に行うことができる．

Docker コンテナのディスクイメージは，図 1 のように読み込み専用のベースイメージの上に書き込み可能な差分イメージを重ねることで作成される，ベースイメージは複数のコンテナで共有されるベースシステムやアプリケーション，ライブラリを含んでおり，コンテナから書き換えることはできない．一方，差分イメージはコンテナごとに用意され，ベースイメージに含まれないアプリケーションやライブラリなどを含む．実行中にインストールされたアプリケーション等や，作成されたログ・ファイルなども差分イメージに含まれる．ベースイメージと差分イメージに対してさらに差分イメージを重ねていくこともできる．

このようなディスクイメージの重ね合わせを実現するために，Docker ではストレージドライバとして OverlayFS と呼ばれるファイルシステムを利用することができる．他にも，AUFS[3] や ZFS[4]，Btrfs[5]，devicemapper[6] などがストレージドライバとして利用できる．その中でも，

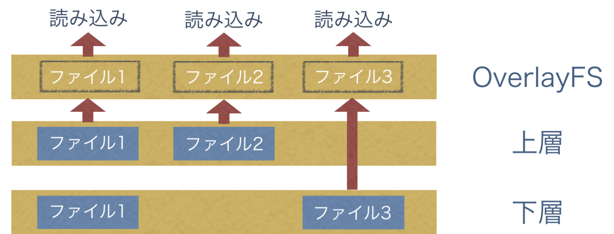

図 2 OverlayFS の読み込み処理

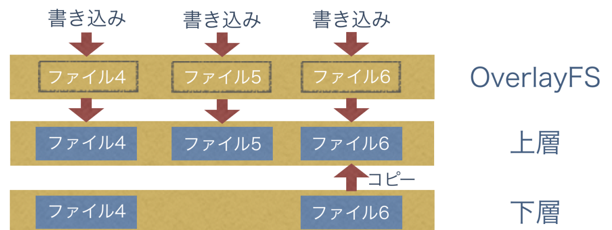

図 3 OverlayFS の書き込み処理

OverlayFS は性能が優れていること，Linux に標準搭載されていること，利用が容易であることなどから広く利用されている．OverlayFS は 2 つのファイルシステムを重ね合わせることができ，下層の読み込み専用のファイルシステムにベースイメージを配置し，上層の書き込み可能なファイルシステムに差分イメージを配置する．

OverlayFS はファイルが上層と下層のどちらに存在するかによって，図 2 のように読み込み処理を行う．ファイル 1 やファイル 2 のように，上層と下層の両方または上層のみに存在するファイルを読み込む時には，上層のファイルを読み込む．ファイル 3 のように，下層にしか存在しないファイルを読み込む時には，下層のファイルにアクセスする．一方，書き込み処理は図 3 のように行う．ファイル 4 やファイル 5 のように，上層と下層の両方または上層のみに存在するファイルに書き込む時には上層のファイルに書き込む．ファイル 6 のように，下層にしかファイルが存在しない場合は，読み込み専用の下層のファイルシステムに書き込むことはできないため，まず，下層にあるファイルを上層にコピーする．次に，上層のファイルに対して書き込みを行う．これはコピーオンライトと呼ばれる機能である．

このように，OverlayFS では下層にのみ存在するファイルに最初に書き込みを行う時に，コピーオンライトのオーバヘッドが発生する．ファイルに書き込む量に関わらず，書き込みを行おうとしたファイル全体がコピーされるため，ファイルサイズが大きいほどオーバヘッドも大きくなる．特に，データベースのように巨大なファイルになるとコピーが完了するまでの間，コンテナの実行が停止する．このファイルのコピーは OS カーネル内で行われるため，CPU，メモリ，ディスクに関するシステム性能にも大きな影響を与える．また，コピーが完了した後は下層と上層

© 2018 Information Processing Society of Japan

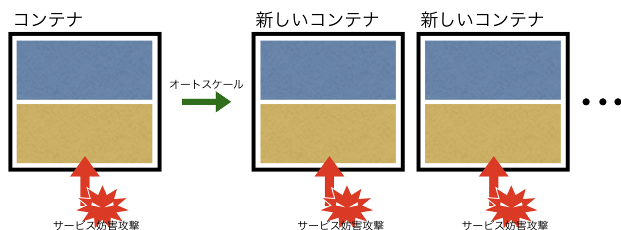

図 4　オートスケールに対するサービス妨害攻撃

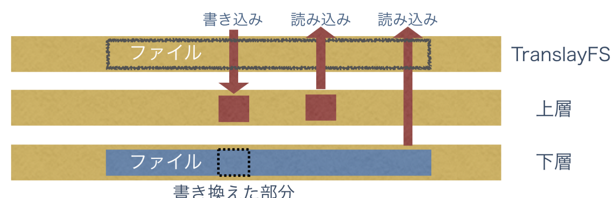

図 5　TranslayFS における下層にファイルの読み書き

図 6　スパースファイル

でほとんど同じファイルを重複して持つことになるため，ディスク容量を圧迫してしまう．

このようなコピーオンライトを外部から意図的に発生させられると，コンテナ上で運用されるサービスに対してサービス妨害攻撃が行われる危険性がある．コピーオンライトはコンテナのディスクイメージの下層にあるファイルそれぞれに対して 1 回ずつ発生させることができる．また，コンテナがオートスケールする設定になっている場合には，図 4 のように大量のリクエストを送信してコンテナをスケールアウトさせることにより，新しいコンテナを作らせることができる．そのコンテナにもまた，下層にあるファイルに対してコピーオンライトを発生させることができる．このようにして，次々にコンテナを一定時間，停止させることができる．また，それぞれのコンテナでファイルのコピーが作成されることになり，システム全体のディスク容量を不足させることもできる．

3.　TranslayFS

OverlayFS を用いたコンテナに対するサービス妨害攻撃を防ぐために，OverlayFS を改良した新たなファイルシステムである TranslayFS を提案する．TranslayFS は下層にあるファイルに対して書き込みが行われた時に，そのデータを上層に書き込むだけで処理を完了する細粒度のコピーオンライトを行う．OverlayFS における粗粒度のコピーオンライトと違い，最初の書き込み時に下層からファイル全体を上層にコピーしてからデータを書き込む必要がない．そのため，サイズが大きいファイルに書き込みを行う際に生じていたオーバヘッドを削減することができる．これにより，ファイルへの書き込みによってコンテナを一時的に停止させるサービス妨害攻撃を防ぐことが可能となる．また，上層にはファイルの変更部分だけを保持することにより，ディスク容量を節約することができる．OverlayFS のように，下層と上層の両方にほとんど同じファイルを保持する必要がなくなる．これにより，システム全体のディスク容量を不足させることによるサービス妨害攻撃も防ぐことができる．

下層にあるファイルが書き換えられた後，TranslayFS へのファイルアクセスは図 5 のように処理される．上層にはファイルの変更部分だけが保持されているため，上層と下層をきめ細かくマージすることで変更後のファイルを読み出す．具体的には，変更部分を読み出す時には上層に保持されたデータを返し，それ以外の部分の読み出しは下層のファイルのデータを返す．これはファイル単位で上層と下層のファイルシステムをマージする OverlayFS と大きく異なる点である．それ以外のファイルアクセスについては OverlayFS と同様である．

下層のファイルの変更部分のみを上層で効率よく管理するにはデータベースなどを用いる必要があり，ファイル全体を上層にコピーしてしまう OverlayFS と比べてはるかに複雑な構造となる．そこで，TranslayFS では，スパースファイルと呼ばれる特殊なファイルを用いてファイルの変更部分を効率よく管理する．スパースファイルは図 6 のようにファイルの一部だけに実データを持たせることのできるファイルである．最初の書き込み時に下層のファイルと同じサイズのスパースファイルを上層に作成し，変更部分にだけデータを書き込む．実データが格納されていない部分はホールと呼ばれ，ディスク上に領域は割り当てられない．そのため，ディスク容量を節約しつつ，OverlayFS と同様にファイルを上層に作成して管理することができる．巨大なファイルであっても作成時には実データをまったく持たないため，高速に作成することができる．

4.　実装

我々は TranslayFS を Linux カーネル 4.4.0 に実装した．本章では，TranslayFS がベースとしたファイルシステムである OverlayFS の実装について説明した後で，それと比較しながら TranslayFS の実装について述べる．

4.1　OverlayFS

OverlayFS はファイルのオープン時に上層と下層のどちらのファイルにアクセスするかを決定する．読み込み専用でアクセスする場合，ファイルが上層に存在すれば上層のファイルをオープンする．上層にファイルがなければ下層のファイルをオープンする．その後は，オープンしたファイルから読み込みを行う．書き込み可能でアクセスする場

© 2018 Information Processing Society of Japan

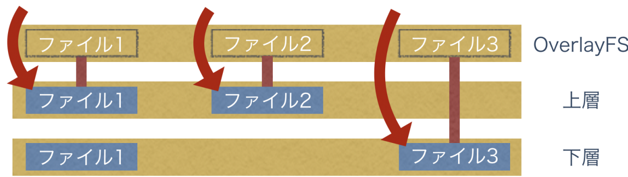

図 7　OverlayFS のファイルアクセス

図 8　TranslayFS のオープン処理

合，ファイルが上層に存在すれば，読み込み専用の場合と同様に上層のファイルをオープンする．一方，上層にファイルがなければ下層のファイルを上層にコピーした後で上層のファイルをオープンする．実際には下層のファイルを書き換えない場合でも，書き込み可能でオープンするだけでファイル全体のコピーが行われる．

図 7 に示すように，上層と下層のいずれかのファイルをオープンした後，アプリケーションは常にそのファイルに対して読み書きを行う．ファイルの読み書きの際には OverlayFS はバイパスされ，上層または下層のファイルシステムに直接アクセスが行われる．そのため，OverlayFS によって 2 つのファイルシステムを重ねることによるオーバヘッドは，ファイルの読み書きに対してはまったく発生しない．

4.2　TranslayFS のオープン処理

TranslayFS は OverlayFS と違い，上層と下層の両方のファイルを扱う必要がある．そのため，図 8 のファイル 1 のように、上層と下層の両方にファイルが存在する場合には，2 つのファイルをオープンする．ファイル 2 やファイル 3 のように一方にだけファイルが存在する場合には，OverlayFS と同様に一方のファイルだけをオープンする．これは読み込み専用でアクセスする場合も書き込み可能でアクセスする場合も同様である．

このように，TranslayFS では書き込み可能でファイルにアクセスする場合も下層のファイルをオープンする必要がある．しかし，下層のファイルシステムは読み込み専用でマウントされているため，書き込み可能でファイルをオープンすることはできない．そのため，下層のファイルをオープンする際には書き込みに関するフラグを削除してオープンする．具体的には，書き込み可能を表す O_WRONLY とファイルサイズを 0 にする O_TRUNC を削除する．

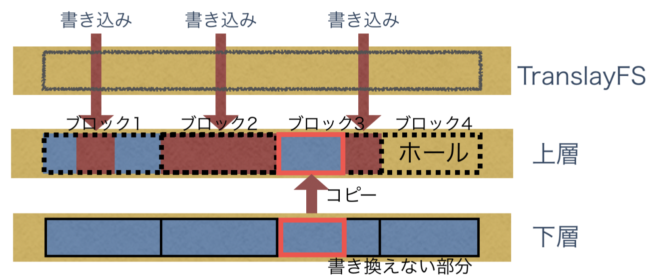

図 9　TranslayFS の書き込み処理

4.3　TranslayFS の書き込み処理

下層にのみ存在するファイルに書き込みを行った時には，TranslayFS は上層にスパースファイルを作成する．その際に，下層のファイルが格納されているディレクトリが上層にまだなければ作成する．そして，下層のファイルとサイズを除いて同じ属性を持つファイルを上層に作成する．実データをまったく持たないスパースファイルを作成するために，まず，サイズが 0 のファイルを作成した後，下層のファイルとサイズが一致するように上層のファイルのサイズを拡張する．そして，作成したスパースファイルをオープンして，既にオープンしている下層のファイルとまとめて管理を行う．

スパースファイルは 4KB のブロック単位でのみ実データを持たないホールを作ることができるため，TranslayFS は図 9 のようにブロック単位で更新されたデータを管理する．そのため，書き込み時にはまず，指定されたバイト単位のファイルオフセットとサイズから，開始ブロックと終了ブロックを計算する．次に，その各ブロックに対してホールかどうかを確認する．ホールの検出方法については 4.5 節で説明する．図 9 のブロック 1 のようにブロックがホールでない場合には，通常通り，スパースファイルにデータの書き込みを行う．ブロックがホールの場合であっても，ブロック 2 のようにブロック全体にデータを書き込む際にはホールでない場合と同様に書き込みを行う．一方，ブロック 3 のようにブロックの一部だけにデータを書き込む場合は，書き込まない部分を下層のファイルからコピーした上でスパースファイルへの書き込みを行う．この場合にだけ下層から上層へのコピーが発生するが，開始ブロックと終了ブロックの一部のみがコピーの対象となり，その間のブロックではコピーは発生しない．この下層からの部分コピーは現在，実装中である．

上層のみにファイルが存在する場合については，通常通り，バイト単位でデータの書き込みを行う．ただし，OverlayFS と違い，TranslayFS の書き込み処理を経由して上層のファイルシステムへの書き込みが行われる．そのため，直接，上層のファイルへの書き込みが行える OverlayFS に比べて性能が若干低下する可能性がある．

© 2018 Information Processing Society of Japan

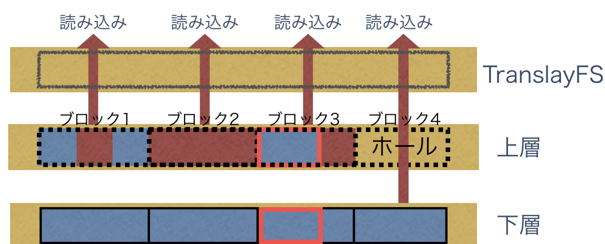

図 10　TranslayFS の読み込み処理

4.4　TranslayFS の読み込み処理

上層と下層の両方にファイルが存在する場合，書き込みの場合と同様にブロック単位で読み込み処理を行う．まず，読み込み時に指定されたバイト単位のファイルオフセットとサイズから，開始ブロックと終了ブロックを計算する．次に，その各ブロックに対してホールかどうかを確認する．図 10 のブロック 1〜3 のようにブロックがホールでなければ上層のスパースファイルからデータの読み込みを行い，ブロック 4 のようにホールであれば下層のファイルからデータの読み込みを行う．上層にはブロック単位で完全なデータが格納されているため，書き込みの場合と違い，上層と下層のどちらかのブロックからのみ読み込めばよい．

下層または上層にのみ存在するファイルを読み込む場合には，通常通り，バイト単位でデータの読み込みを行う．読み込みの場合にも，OverlayFS と違い，TranslayFS の読み込み処理を経由する必要があるため，オーバヘッドが生じる可能性がある．

4.5　ホールの検出

Linux では，アプリケーションがスパースファイル中のホールを探すために 3 つの方法が提供されている．1 つ目は，FIBMAP ioctl を用いて論理ブロック番号に対応する物理ブロック番号を取得する方法である．取得した物理ブロック番号が 0 の場合，実データが格納されていないということを意味し，ホールであることが検出できる．2 つ目は，FIEMAP ioctl を用いてファイルのデータが連続している領域を取得する方法である．データが連続していない領域をホールとして検出できる．3 つ目は，lseek システムコールで SEEK_DATA または SEEK_HOLE を指定して，次の実データまたはホールの位置を探す方法である．

TranslayFS では特定のブロックがホールかどうかを検出する必要があるため，1 つ目の方法をカーネル内で実装して用いた．TranslayFS は上層のファイルシステムのブロック番号を変換する機能を呼び出して，図 11 のように指定された論理ブロック番号を物理ブロック番号に変換する．論理ブロック番号はファイルの先頭ブロックを 0 として，4KB ごとに順に割り振られている．変換した物理ブロック番号が 0 ならホールと判定し，それ以外ならホールでないと判定する．

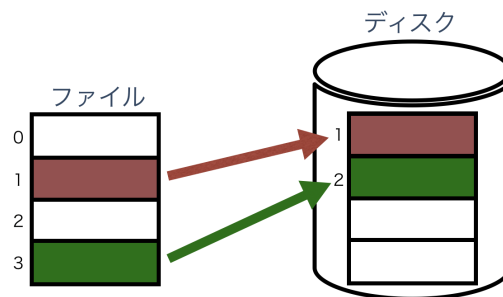

図 11　ホールの検出

表 1　実験環境

CPU	Intel Core i7-3770 3.40GHz
メモリ	8GB
ハードディスク	SATA3 HDD 128GB
OS	Linux 4.4.0
Docker	Docker 1.13.1

5.　実験

TranslayFS を用いたコンテナがサイズの大きなファイルを読み書きする際の性能を調べる実験を行った．比較として，OverlayFS を用いた時の読み書き性能についても測定した．また，Docker で用いられる他のストレージバックエンドとの比較も行った．用いたストレージドライバは AUFS[3] と ZFS[4]，devicemapper[6] である．AUFS は OverlayFS に似たファイルシステム，ZFS はコピーオンライト機能を備えたファイルシステムであり，devicemapper はブロックレベルでイメージ管理を行うストレージドライバである．実験環境を表 1 に示す．

5.1　OverlayFS との書き込み性能の比較

まず，10GB のファイルを含んだベースイメージを用意し，そのベースイメージと差分イメージを用いてコンテナを作成した．次に，そのコンテナ内で 10GB のファイルをオープンして 1 バイトのデータの書き込みを行い，書き込みの完了にかかる時間を測定した．測定結果を図 12 に示す．OverlayFS は，1 バイトのデータを書き込むのに 44 秒の時間を要した．これは下層のファイルを上層にコピーするオーバヘッドによるものである．書き込み完了後には上層に 10GB のファイルがコピーされていた．一方，TranslayFS では，1 バイトの書き込みにかかる時間は 0.15 ミリ秒となり大幅に高速化することができた．書き込みにかかる時間は，スパースファイルの作成にかかる時間と 1 バイトの書き込みにかかる時間のみとなり，ファイル全体のコピーは発生しなかった．実際，上層に 4KB の実データだけを持つスパースファイルが作成されていた．

5.2　OverlayFS との読み込み性能の比較

5.1 節で 1 バイトの書き込みを行ったファイルに対して 10GB のデータの読み込みを行い，TranslayFS と Over-

© 2018 Information Processing Society of Japan

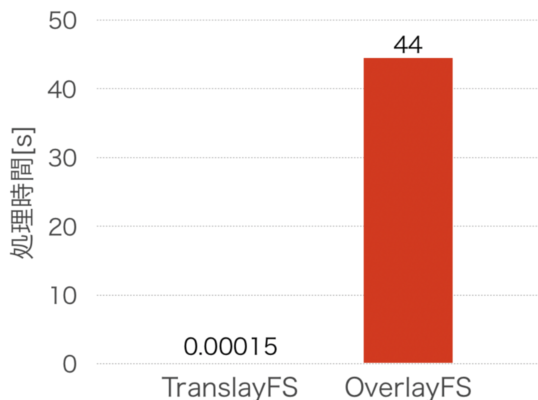

図 12　書き込み性能

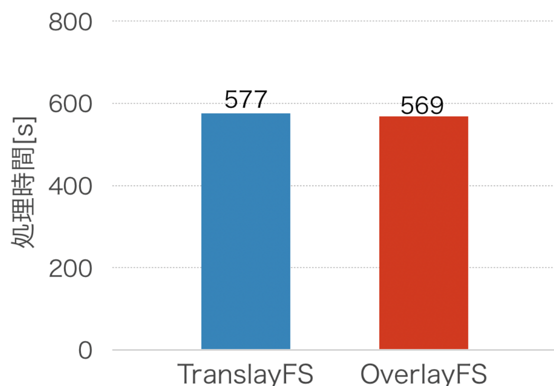

図 13　読み込み性能

layFS を用いた場合の読み込み時間を測定した．測定結果を図 13 に示す．TranslayFS では OverlayFS に比べて 1.3%の性能低下が見られた．これは，TranslayFS の読み込み処理を経由することによるオーバヘッドおよび，ブロック単位でホールの検出を行いながら読み込みを行うオーバヘッドによるものだと考えられる．

5.3　他のストレージドライバとの性能比較

OverlayFS 以外のストレージドライバを用いて作成したコンテナに対して，5.1 節と 5.2 節と同様にして読み書き性能の測定を行った．1 バイトの書き込みにかかる時間の測定結果を図 14 に示す．TranslayFS に比べ，AUFS は書き込み時間がさらに長くなった．これは，TranslayFS では Linux の splice 機能を用いて下層のファイルを上層のファイルに高速にコピーするのに対して，AUFS では下層のファイルを読み込んでから上層のファイルに書き込むためだと考えられる．ZFS は TranslayFS と同様にファイルブロック単位でのコピーオンライトを行うため，書き込み時間が 1 ミリ秒になった．devicemapper はディスクブロック単位のコピーオンライトを行うが，書き込みに 1.1 秒の時間がかかった．

図 10 に 10GB のファイルの読み込みにかかる時間の測定結果を示す．AUFS は TranslayFS よりも読み込み時に複雑な処理を行うため，読み込み時間が長くなった．ZFS はデータブロックを読み出す際にブロックポインタをたどる回数が多く，同一の処理に対してより多くの I/O が必要になるため，読み込み時間がどのストレージドライバよりも長くなった．devicemapper は AUFS と同程度の性能であった．

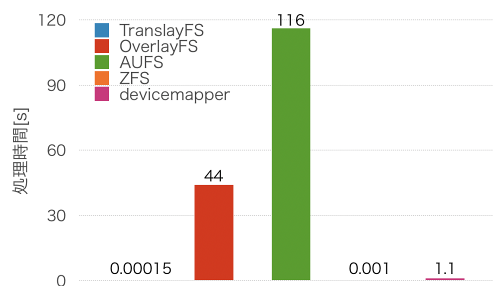

図 14　ストレージドライバごとの書き込み性能

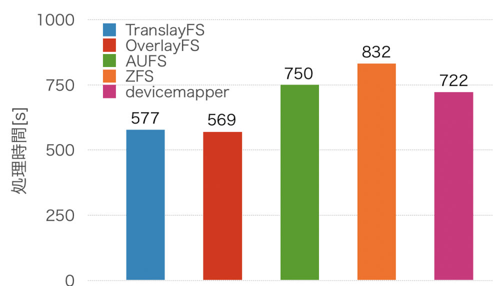

図 15　ストレージドライバごとの読み込み性能

6.　関連研究

2 つのファイルシステムを重ねる OverlayFS は Union ファイルシステムの一実装である．Union ファイルシステムの他の実装には，SunOS の Translucent ファイルシステム [7]，4.4 BSD-Lite の Union マウントファイルシステム [8]，UnionFS [9]，AUFS[3] などがある．Union ファイルシステムはファイル単位のコピーオンライト機能を備えており，下層のファイルに対する書き込みを行うと上層にファイル全体がコピーされる．OverlayFS ではファイルを書き込み可能でオープンした時点でファイルのコピーが行われるが，UnionFS や AUFS ではファイルへの最初の書き込みが行われた時点でファイルのコピーが行われる．AUFS は Docker でサポートされているが，Linux のメインライン・カーネルにマージされていないため，あまり利用されなくなっている．

ZFS は Sun Microsystems 社が Solaris 用に開発したファイルシステムであり，現在では ZFS on Linux プロジェクト [4] で開発されている．ZFS は高い信頼性を持つ高機能なファイルシステムである．ZFS では物理ディスク等からストレージプールを作成し，ストレージプールからファイルシステムを作成する．Docker ではファイルシステム上のベースイメージから読み込み専用のスナップショットを作成し，スナップショットから読み書き可能なクローンを作成する．クローンへの書き込み時にはブロック単位でコ

© 2018 Information Processing Society of Japan

ピーオンライト処理が行われるが，ブロックサイズは最大128KB と比較的大きい．また，ZFS のライセンスであるCDDL は GPL と互換性がないため，Ubuntu でしか標準でサポートされていない．

devicemapper[6] は Red Hat 社が Linux の Device Mapper 機能を用いて開発したストレージドライバである．OverlayFS 等とは違い，ファイルシステムレベルではなくブロックレベルで動作する．devicemapper はデータデバイスとメタデータデバイスからなるシンプールを作成し，シンプールからシンプロビジョニングによりベースデバイスを作成する．Docker ではベースデバイス上のベースイメージから読み書き可能なスナップショットを作成する．スナップショットへの書き込み時にはブロック単位でコピーオンライトが行われるが，ブロックサイズは 64KB と比較的大きい．

Btrfs[5] は Oracle 社，Red Hat 社などが共同で開発したファイルシステムである．ZFS を基に作られており，コピーオンライト機能やスナップショット機能を備え，耐障害性が高い点が特徴としてあげられる．ZFS と同様にストレージプールを作成し，ストレージプールからファイルシステムの一部であるサブボリュームを作成する．Docker ではサブボリューム上のベースイメージから読み書き可能なスナップショットを作成する．スナップショットへの書き込み時にはブロック単位でコピーオンライトが行われ，ブロックサイズは 16KB と比較的小さい．しかし，Btrfs はまだ十分に安定しているとは言えず，RHEL8 以降ではサポートされないことが決定している．

qcow2 は QEMU[10] などで仮想マシンのディスクイメージとして用いられているファイルフォーマットである．コピーオンライト機能を用いて読み込み専用のベースイメージに対する書き込みを差分として別のファイルに保存することができる．また，qcow2 はスパースファイルと同様に書き込まれた分だけディスク領域を確保し，書き込みが行われるたびにディスクイメージの拡張を行うことでディスク容量を節約している．OS が標準的にサポートするフォーマットではないため，マウントを行うために qemu-nbd などのツールが必要となる。

重複除外の技術を用いることで，ディスク上の重複するデータを一つにまとめてディスク容量を節約することができる．重複除外にはディスクに書き込む際に処理を行うインライン方式と，書き込んだ後で処理行うポストプロセス方式がある．ZFS，Btrfs，macOS の APFS などのファイルシステムではこの機能を用いることで，ファイルのコピー時に実データを複製しないようにすることができる．OverlayFS が用いる上層と下層にこれらのファイルシステムを用いることで，下層のファイルを上層にコピーするオーバヘッドを削減することができる．TranslayFS は OverlayFS に重複除外の機能を追加したものと考えることもできる．

7. まとめ

本稿では，OverlayFS を用いたコンテナに対するサービス妨害攻撃を防ぐために，新たなファイルシステムである TranslayFS を提案した．TranslayFS では，下層のファイルへの書き込み時にファイル全体が上層にコピーされるのを防ぐために，上層にスパースファイルを作成してファイルの変更部分だけを保持する．そのファイルの読み込み時には，変更部分は上層のスパースファイルから読み込み，それ以外の部分は下層のファイルから読み込む．これにより，サイズの大きなファイルへの書き込みによるコンテナの一時停止や，上層にファイル全体のコピーが作られることによるディスク容量の圧迫を防ぐことができる．我々は TranslayFS を Linux カーネルに実装し，他のストレージドライバとの性能比較を行った．その結果，サイズが大きいファイルに対する書き込み性能は OverlayFS に比べて大幅に向上し，読み込み性能については 1.3%の性能低下に抑えることができた．

今後の課題として，ブロックの一部にだけ書き込まれた際に，下層のブロックの一部を上層のスパースファイルにコピーする機能を実装する必要がある．また，さまざまなファイルの読み書き性能を測定したり，読み書き以外の性能を測定したりして，TranslayFS の性能を明らかにする．TranslayFS ではファイルを一括コピーしないことによりデータのフラグメントが発生するため，性能が低下する可能性がある．また，ファイルサイズが小さい場合などに OverlayFS のほうが性能がよくなる可能性があるため，ファイルサイズなどによって TranslayFS と OverlayFS の実装を使い分ける仕組みを必要に応じて検討する．

参考文献

[1] Docker, Inc., Docker: Build, Ship, and Run Any App, Anywhere, https://www.docker.com/.

[2] M. Szeredi, Overlay Filesystem, https://www.kernel.org/doc/Documentation/filesystems/overlayfs.txt.

[3] J. R. Okajima, "Aufs4 - Advanced Multi Layered Unification Filesystem Version 4.x," https://aufs.sourceforge.net/.

[4] Lawrence Livermore National Laboratory, ZFS on Linux, https://zfsonlinux.org/.

[5] Oracle Corporation, Btrfs File System, https://btrfs.wiki.kernel.org/index.php/Main_Page.

[6] Red Hat, Inc., dm-thin: Thin Provisioning, https://www.kernel.org/doc/Documentation/device-mapper/thin-provisioning.txt.

[7] D. Hendricks, "A Filesystem for Software Development," USENIX Summer 1990 Conf. Proc., pp.333-340, 1990.

[8] J. Pendry, M. K. McKusick, "Union Mounts in 4.4BSD-Lite," USENIX 1995 Technical Conf. Proc., pp.25-33, 1995.

[9] D. P. Quigley and J. Sipek and C. P. Wright and E.

© 2018 Information Processing Society of Japan

Zadok, "UnionFS: User- and Community-oriented Development of a Unification Filesystem," Proc. of Linux Symposium, pp.349-362, 2006.
[10] Bellard: QEMU, https://www.qemu.org/

© 2018 Information Processing Society of Japan