Vol.2019-OS-145 No.14
2019/3/1

Dockerコンテナ向けスナップショット機構のためのOverlayFSの機能拡張

水木 航平[1]　廣津 登志夫[1]

概要：近年，多くの企業でコンテナ型仮想化によるマイクロサービスアーキテクチャが広く使われるようになってきた．コンテナ型仮想化は起動時間が高速でイメージ管理が容易であるという特徴があり，Linux向けのコンテナ仮想化技術としてはDockerが広く使われている．このようなコンテナ型仮想化の利用場面を考えると，誤編集や誤操作によりサービス基盤となる重要な情報を壊してしまう人為的ミスや，攻撃によりコンテナ内のコンテンツが改竄されるリスクに対処する仕組みが必要になってくる．そこで，本研究ではDockerのイメージ管理に使用されるOverlayFSを改良して任意のタイミングでスナップショットを保存する機能を実現する．ここでは，状態の保存と観測というそれぞれの目的に適したスナップショットの方法として，ノーマルモードと強制書き出しモードの2つの方法を用意した．

1. 序論

近年，クラウド環境のサービス基盤や企業等の組織のサービス提供の場面で，コンテナ型仮想化によるマイクロサービスアーキテクチャが広く使われるようになってきている．コンテナ型仮想化は，ハードウェア自体の仮想化を行うハイパーバイザー型仮想化と比較して起動が高速で，イメージを柔軟に管理できるという特徴がある．そのため，マイクロサービスアーキテクチャなどの環境と相性が良く，コンテナ型仮想化を利用するケースが増えている．

コンテナ型仮想化は，インターネット向けのサービス基盤として使われていることから，コンテナ上で稼働するサービスへの攻撃により，ファイルシステムが改竄されたり悪意のあるソフトウェアが実行されたりするリスクへの対処が必要である．また，インターネットに晒されていなくても，組織内部向けの重要なサービスが提供する場合も多いことから，ファイルシステムやハードウェアの障害だけでなく，操作ミスや編集の誤りといった場合にも速やかに回復できることが望ましい．これに対して，コンテナ管理機能の一部としてファイルシステムの状態の保存や状態の改編の観測を行う機能が提供できれば，コンテナ環境を提供しているホスト環境側から定期スナップショットやファイルの改竄監視機能を提供することが可能になる．

Linux向けの代表的なコンテナ型仮想化ソフトウェアであるDocker[1]の場合，コンテナイメージ管理を行う仕組みを利用してファイルの変更内容の検知とファイルシステムの状態の保存を実現する．Dockerのコンテナイメージは，OverlayFS[2]の機能を用いて，コンテナが動作するファイルシステムを複数のイメージをレイヤとして重ね合わせることにより実現されている．レイヤには2種類存在し，コンテナごとに1つ設定されコンテナ内のファイルの変更情報の書き込みが行われる書き込み先レイヤと複数コンテナによって共有して使用される読み込み専用の複数指定可能なベースイメージレイヤである．OverlayFSを使用したコンテナイメージ管理方法の場合，レイヤはホスト環境のディレクトリであり，複数のディレクトリを統合することによってコンテナファイルシステムを実現している．OverlayFSを使用したコンテナイメージ管理方法以外に，ZFS[3]やLVM(Logical Volume Manager)を使用する方法がある．ZFSはスナップショット機能を持つファイルシステムであり，LVM(Logical Volume Manager)は複数の物理ディスクをまとめて論理ボリュームとして使用できる仕組みでスナップショット機能を持つ．しかし，これらは，ブロックデバイスレベルの管理を行っており，ホスト環境からコンテナのファイルシステムの内容を検知することができない．

このDockerのイメージ管理の仕組みを改良して，任意のタイミングでスナップショットを保存する機能を実現する．具体的には，OverlayFSの書き込み先と読み込み専用のレイヤ構成を変更する機能を提供して，remountをトリガーにしてコンテナの動作するファイルシステムを構成するレイヤを新しく追加し，書き込み先レイヤの切り替

[1] 法政大学 情報科学部
Hosei University

えを行う．この機能拡張を行うことにより，構成レイヤを切り替えた時のファイルシステムの状態を切り替え前の書き込み先レイヤに保持することが可能となる．さらに，OverlayFS を使用したコンテナイメージ管理では，レイヤをホスト環境上のディレクトリとして保存するため，ファイルの変更内容などの詳細な情報をホスト上で検知することが可能となる．ここでは，特定の期間のファイルシステムの状態への切り戻しのためのファイルシステムの状態の保存とファイルの改竄検知のためのファイルシステムの状態の観測というそれぞれの目的に適したスナップショットの方法として，open 時のファイルシステムの状態を残すノーマルモードと remount 時の状態を残す強制書き出しモードの 2 つの方法を用意した．

2. Docker

Docker では，Linux の cgroup[4] と namespace[5] を用いて隔離された環境のコンテナを作成する．コンテナ内のプロセスはホストのカーネル上で動作し，ホスト環境上では自身のプロセスとして管理できるため，ハードウェア自体の仮想化を行うハイパーバイザー型仮想化と比べてオーバーヘッドが小さい．また，コンテナのイメージは Dockerfile としてファイルに定義することが可能なため，イメージの管理やバージョン管理を柔軟に行うことができる．

Docker は，コンテナが動作するファイルシステムを複数のコンテナイメージをレイヤとして重ね合わせることによって実現する．コンテナイメージは，書き込みレイヤとベースイメージレイヤの 2 種類のレイヤから構成される．書き込みレイヤは，コンテナ内のファイルの変更を書き込むためにコンテナごとに作成されるレイヤである．ベースイメージは，コンテナのファイルシステムの元となるディレクトリ・ファイルが保存されており，読み込み専用で複数指定可能なレイヤである．それぞれのレイヤには，親イメージレイヤからの差分情報として，ファイル・ディレクトリの変更内容が保存される．コンテナファイルシステムを複数のイメージの重ね合わせによって実現しているが，実際に管理を行う方法は独自に実装できるようにストレージ管理のためのインターフェースを提供している．

2.1 OverlayFS

Docker のイメージ管理方法の実装の 1 つとして OverlayFS を使用した方法がある．OverlayFS は，複数のディレクトリを重ね合わせて透過的にファイル・ディレクトリの操作を行うことが可能な単一のファイルシステムを実現する Union Mount[6] ファイルシステムの 1 つである．OverlayFS は，複数ディレクトリを統合的に見せるためのマウントポイント (以下 merged) と，書き込みが可能なレイヤである upperdir，読み取り専用で複数指定可能なレイヤである lowerdir，コピーアップ時の一時ファイル置き場

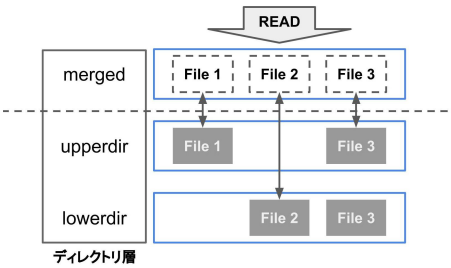


図 1 OverlayFS の Read 処理

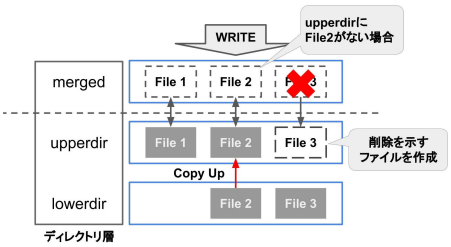


図 2 OverlayFS の Write 処理

などの OverlayFS の処理に使用されるディレクトリである workdir の 4 つの要素によって構成されている．ファイルの検索は，最上位レイヤである upperdir から lowerdir へと順に行われ，ファイルが見つかった場合，それ以降のレイヤに対して検索は行われない．そのため，複数のディレクトリレイヤに同じファイル名がある場合は，より上位のファイルが優先され操作が行われる．

図 1 は，OverlayFS の Read 時の処理内容を示している．File1 を読み込む場合，最初に検索が行われる upperdir にファイルが存在するため，upperdir のファイルに対して処理が行われる．File2 を読み込む場合，upperdir の検索した後に lowerdir を検索でファイルが見つかるため，lowerdir のファイルに対して処理が行われる．File3 を読み込む場合，File1 と同様に最初に upperdir にファイルが存在するため，upperdir のファイルに対して処理が行われ，lowerdir の File3 は無視される．

図 2 は OverlayFS の Write 時の処理方法を示している．File1 に書き込む場合，read の File1 と同様に upperdir のファイルに対して処理が行われる．File2 に書き込む場合，lowerdir の File2 の内容を upperdir にコピーし，完了後 upperdir のファイルに対して処理が行われる．File3 を削除する場合，upperdir に File3 の名前で特殊なキャラクタデバイスファイルを作成する．これによって，lowerdir のファイルを変更せずに削除されたことを表している．

2.2 OverlayFS を使用したコンテナイメージ管理方法

図 3 は，OverlayFS を使用したコンテナイメージ管理の構成である．OverlayFS の merged はコンテナが動作するファイルシステムとなり，upperdir は書き込みを行うレイヤ，lowerdir は複数指定可能な読み込み専用のベースイメージ部分に相当する．このように指定してマウントする

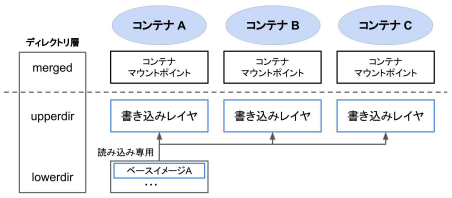


図 3 OverlayFS を使用したコンテナイメージ管理の構成

ことによって，コンテナのファイルシステムのための複数イメージで構成された単一のファイルシステムを実現している．OverlayFS は upperdir, lowerdir がディレクトリのため，コンテナ内からは単一のファイルシステムとして見えるが，ホスト環境からは複数のディレクトリとして見えている．そのため，それぞれのレイヤのファイル・ディレクトリの内容をホスト環境から確認することが可能である特徴を持つ．

3. 設計

本研究は，第 2.2 章の OverlayFS によるコンテナイメージ管理方法をもとに新しい書き込み先レイヤを積み重ねて，コンテナ内のファイルの変更を書き込み先のディレクトリを切り替えることによって，書き込み先切り替え時のファイルシステムの状態を保存する．この機能拡張を用いて，コンテナ内のファイルシステムのスナップショット機能とファイルの改竄検知を実現する．

3.1 保存先切り替え処理

コンテナのファイルシステムにおいて，新しい書き込み先レイヤを積み重ね，コンテナ内のファイルの変更情報の保存先を切り替える処理の実現のため，ホスト環境下での監視，切り替えを行うトリガー，タイミングについて検討を行う．

3.1.1 ホスト環境上からの監視

コンテナにおけるファイルシステムのスナップショットやファイルの改竄検知を行うために，コンテナファイルシステムの状態の保存とコンテナ内のファイルの変更内容の検知が必要となる．これらの機能を実現する時に，ファイルシステムの状態やファイルの変更内容に関して，ホスト環境上から監視可能な方法をとることによって，コンテナをまとめて監視で，柔軟な管理を行うことができる．そのため，コンテナ内の変更情報を透過的にホスト環境から検知できる仕組みが必要である．

3.1.2 切り替えを行うトリガー

保存先切り替え処理は，コンテナ内で動作するプロセスがファイル操作を行う際に切り替えを意識させないために透過的に行う必要がある．そのため，保存先切り替え処理を行うための方法として，remount 時にコンテナファイルシステムを構成するレイヤの変更を行う．その際に，新しい書き込み先レイヤには切り替え前のファイル内容がないため，書き込み処理が行われる際にはコピーアップ処理が行い書き込みを行うレイヤの切り替えを行う必要がある．また，保存先切り替え時にファイルがオープンされていた時の実際の書き込み先レイヤを切り替えるタイミングを考える必要がある．

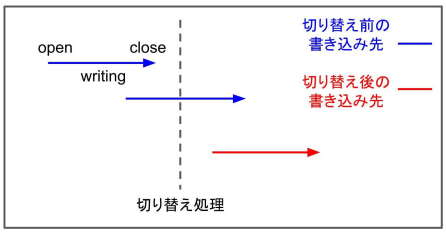


図 4 ノーマルモード

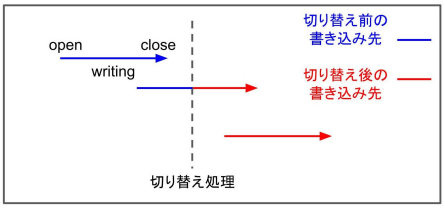


図 5 強制書き出しモード

3.1.3 切り替えを行うタイミング

第 3.1.2 章で述べたトリガーによって保存先の切り替えが行われた際に，ファイルがオープンされていた際に，実際の書き込み先を切り替えるタイミングについて述べる．保存先切り替え処理後，実際の書き込み先レイヤを切り替えるタイミングとして open 処理時に切り替えを行うノーマルモードと保存先切り替え処理時い切り替えを行う強制書き出しモードの 2 種類の方法が考えられる．

ノーマルモードの切り替えのタイミングについて，図 4 に示す．この方法では，保存先の切り替え時にファイルがオープンされている場合，close 処理が行われるまでオープンしているプロセスは，切り替え前の書き込み先に対してファイル操作を行う．そして，保存先切り替え後に新しく open 処理が行われた際に実際の書き込み先レイヤの切り替えが行われ，新しい書き込み先レイヤのファイルに対してファイルの変更処理が行われる．そのため，切り替え前の書き込み先レイヤに書き込み途中のファイルの状態は残らず，切り替え前の書き込み先レイヤのファイルシステムの一貫性を保つことができ，ファイルシステムの状態を切り戻すためのスナップショットとして使用できる．この切り替え方法を用いることにより，コンテナ内のスナップショット機能を実現できるが，この方法を使用する上で問題点が 2 つ存在する．1 つ目が切り替え時にファイルがオープンしていた場合にファイルの不整合が起こる可能性がある点である．保存先切り替え時にファイルがオープンしており，切り替え前の書き込み先レイヤに対して書き込みが行われ続けている際に，別のプロセスがそのファイル

```
mount -t overlay overlay -o lowerdir=/test/lower,
upperdir=/test/upper,workdir=/test/work /test/merged
```

図 6 OverlayFS mount

```
mount -t overlay overlay -o remount,lowerdir=
/test/upper:/test/lower,upperdir=/test/upper2,
workdir=/test/work /test/merged
```

図 7 OverlayFS remount

```
struct ovl_inode {
...
    struct inode vfs_inode;
    struct dentry *__upperdentry;
...
};
```

図 8 ovl_inode 構造体

を open すると切り替え前と切り替え後のレイヤにそれぞれ書き込みが行われる状態となり，ファイルの不整合が起こる可能性がある．2 つ目が close 処理を行わず write 処理を行い続けるプロセスが存在する限り，スナップショットとして使用できない点である．保存先切り替え時にファイルがオープンしており，切り替え前の書き込み先レイヤに対して書き込みが行われ続けている間，切り替え前の書き込み先レイヤに書き込み途中のファイルの状態が存在する可能性があるため，切り替え前の書き込み先レイヤに対して書き込み処理が行われなくなるまでスナップショットとして使用できない．

強制書き出しモードの切り替えのタイミングについて，図 5 に示す．この方法では，保存先切り替え処理時にファイルがオープンされている場合もすべて書き込み先の切り替えが行われ，それ以降はすべてのファイル操作は新しい書き込み先レイヤに対して行われる．そのため，現在の書き込み先レイヤと切り替え前の書き込み先レイヤのファイル・ディレクトリの差分を見ることでファイルの変更内容を検知することが可能となり，コンテナ内のファイルの改竄検知機能の実現が可能となる．しかし，問題点として保存先切り替え時にファイルがオープンされていた場合も書き込み先を切り替えるため，切り替え前の書き込み先レイヤに書き込み途中のファイルの状態が残る可能性があり，ファイルシステムとして不整合が起こる．そのため，この方法の場合スナップショットとして切り戻しを行うことができない．

3.2 OverlayFS の機能拡張

第 3.1 章の保存先切り替えの設計をもとに OverlayFS に機能拡張について述べる．図 6 は，OverlayFS を使用したマウント方法を示している．OverlayFS は，upperdir や lowerdir に関するディレクトリの指定をマウントオプションとして指定し，指定したディレクトリをマウントポイントに重ね合わせて見せる．現在の OverlayFS の実装は，upperdir や lowerdir の設定を変更する場合，unmount を行い再度 mount を行う必要があり，マウントしたままのレイヤ構成の変更ができない．そこで，remount 時に upperdir と lowerdir の設定を変更できるように機能拡張を行う．図 7 は，機能拡張を行い，remount 時にレイヤ構成を変更する方法を示している．この機能拡張によって，第 3.1.2 章で述べた切り替え時にトリガーの設計を満たす．

4. 実装

OverlayFS に対して，remount 処理時に書き込み先と読み込み専用のレイヤ構成を変更する実装について述べる．その際に，保存先切り替えのタイミングとして 2 つの方法で実装を行った．

4.1 OverlayFS の実装

Linux Kernel のファイルシステムは，VFS(Virtual File System) と呼ばれる共通処理を行う層と実際の実装部分に分かれる．また，ファイル・ディレクトリのパスを dentry 構造体として表し，実際のファイル・ディレクトリを inode 構造体として表している．dentry 構造体は親の dentry 構造体と繋がっており，これらをたどることによってファイルパスが表現され，dentry 構造体に設定された inode 構造体の d_inode メンバに対して操作を行うことで実際のファイル操作が行われる．ファイルをオープンする場合は引数として渡すファイルパスを元に dentry 構造体を求める処理として lookup 処理が行われ，create 処理によって dentry 構造体と inode 構造体が作成される．lookup 処理や create 処理は，ファイルシステム固有の処理によって取得するが，一度取得するとキャッシュとしてメモリに保持される．そして，それ以降の lookup 処理では，ファイルの変更処理が行われるまでは VFS 層の共通 lookup 処理時にキャッシュされた dentry が返される．

OverlayFS において，inode 構造体のアロケーションを ovl_inode 構造体として行う．図 8 は，ovl_inode 構造体の一部のメンバを表す．ovl_inode 構造体には，VFS で処理を行う際の inode 構造体として vfs_inode メンバと実際に書き込み処理を行う dentry 構造体の__upperdentry メンバがある．__upperdentry は，upperdir に設定されたディレクトリからのパスとなり，upperdir のファイルシステムに属する dentry 構造体である．OverlayFS では，実際のファイル操作は行われず，upperdir と lowerdir に設定されたディレクトリをもとにファイル操作を行うレイヤに対して実際のファイル操作を実行する役割を持つ．

OverlayFS のマウント時に root の dentry の作成を行い，その際に ovl_inode 構造体のアロケーションが行われる．ovl_inode 構造体には，upperdir のパスをもとに__upperdentry メンバが設定され，root の dentry に，

ovl_inode 構造体の vfs_inode メンバを設定する．ファイルに対して書き込みなどのファイル内容の変更処理が行われる場合，対象の dentry の ovl_inode 構造体に__upperdentry が設定する処理を行う．__upperdentry の設定は，親の dentry を探していき，__upperdentry が設定されている場所から__upperdentry の作成と ovl_inode 構造体の設定を行う．__upperdentry を作成する際に，lowerdir のディレクトリを検索し，ファイル内容のコピーアップを行う．__upperdentry の設定されている親の dentry 検索処理は，マウント時に root の dentry は設定されているため最悪 OverlayFS の root までたどる．

4.2 保存先切り替えの実装

OverlayFS の remount 処理時に設定した upperdir, lowerdir のディレクトリパスをもとに root の dentry の__upperdentry の設定を書き換える処理を行うことができるように OverlayFS に対して機能拡張によって，レイヤ構成の変更を実現する．これによって，新しく lookup 処理が行われ，root の dentry の__upperdentry をもとに実際の書き込み先の dentry を作成する際に新しい書き込み先にファイル操作が行われるようになる．しかし，保存先切り替え処理前にファイル・ディレクトリの lookup 処理が行われている場合，VFS 層の lookup 処理によってキャッシュが返されてしまい，切り替え前の保存先にファイル操作を続けてしまう問題がある．この問題の対処のために，VFS の invalidate 処理を用いて，保存先切り替え後はキャッシュを使用せず，OverlayFS 固有のルックアップ処理を使用するように変更する．一度ルックアップ処理を行うと，メモリ上に dentry と inode がキャッシュされ，ファイルの変更が行われるまでファイルシステム固有の処理を使用せず，VFS 内のルックアップ処理を使用してルックアップ処理が完結する．dentry と inode に関してキャッシュが見つかった場合でも，ファイルシステム固有のルックアップ処理を行うかの判定する処理が invalidate 処理となる．root の dentry とルックアップ処理を行っている dentry の__upperdentry のパスを比較して判定を行う．これによって，root の dentry と同じ upperdir のパスを持っていない場合は，dentry と inode がキャッシュされていても，OverlayFS 固有の lookup 処理が行われて保存先切り替えが正しく動作する．

4.3 切り替えタイミングの実装

ノーマルモードの切り替えのタイミングの場合，remount 時の処理は root dentry の__upperdentry を切り替える処理のみとなる．保存先切り替え後に，新しく open 処理が行われると，invalidate 処理によってキャッシュを使用せずに OverlayFS のルックアップ処理が行われ，新しい保存先に切り替えが行われる．それに対して，強制書き出しモードの切り替えのタイミングの場合，root dentry の切り替えのほかに，開いているファイルに関する file 構造体に設定された dentry 構造体の書き換え処理を行う．書き換え処理時に，invalidate 処理によって OverlayFS のルックアップ処理が行われて dentry を作成し，切り替えを行う．ルックアップ処理時に新しい保存先にコピーアップ処理が必要となるため，開いているファイルが多いと切り替えにかかる時間は増加する．

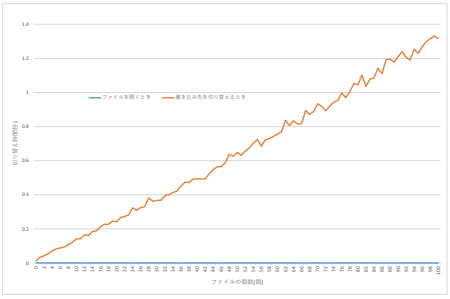


図 9 オーバーヘッド

5. 評価

実際の書き込み先の切り替えタイミングとして 2 種類の方法において切り替えにかかる時間の評価を行った．また，本研究での利用用途として構成するレイヤ数の増加が考えられるため，レイヤ数増加によるオーバーヘッドの評価を行った．そして，ファイルサイズの変化の監視を行うことができ，ファイルの改竄検知に利用できることを示す．

5.1 切り替えオーバーヘッド

ノーマルモードと強制書き出しモードを用いて保存先切り替え処理にかかるオーバーヘッドを計測する．Docker のコンテナ上で，10MB のファイルを開いたままで保存先切り替え処理を行ったときの切り替えにかかる時間を計測した．保存先切り替えにかかる時間の計測は，OverlayFS のリマウント時の処理を行う関数である ovl_remount の処理の最初と最後にタイムスタンプを設定して測った．

図 9 は切り替えオーバーヘッドの計測結果である．新しく open 処理するときの切り替え方法は，root の dentry に設定された書き込み先の dentry を切り替えのみのため，開いているファイルのファイルサイズや個数による影響はなく，平均 225.04 マイクロ秒であった．remount 時の切り替え方法は，root の dentry 切り替えに加えて，開いているファイルの内容を新しい書き込み先レイヤにコピーアップを行う必要がある．そのため，コピーアップによるローカルコピーの時間がかかり，開いているファイルのファイルサイズの総量に応じて切り替え処理にかかる時間が増加する．10MB のファイルを 100 個開いている状態の切り替え

時は，1GB のコピー処理が発生しており，切り替えにかかる時間は 1.316 秒であった．

5.2 レイヤ数による処理時間のオーバーヘッド

本提案手法を用いたコンテナファイルシステムの監視や保存においては，定期的に切り替え処理を行うことが想定されるため，コンテナファイルシステムを構成するレイヤ数が通常よりも増加する．そこで，レイヤ数によるファイルの read/write 処理のオーバーヘッドの計測した．結果を図 10 に示す．この評価では，ベースイメージの一番下に設定されたレイヤにあるファイルに対してファイルの操作を行う際に，その間のレイヤを増加させてファイルの読み書きを行い，その時の open, read/write, close 処理にかかった時間を計測した．read 処理，write 処理ともにレイヤ数が増加すると，open 処理にかかる時間が増加したが，read/write, close 処理にかかる時間の変化はなかった．

5.3 ファイルサイズ変化の検知

ファイルの改竄が行われるケースとして，コンテナにすでにインストールされている標準コマンドの内容を書き換え，書き換えられた標準コマンドが実行された際に，何らかの悪意のある挙動を起こす場合がある．標準コマンドのように一度インストールされた後にほとんどファイル内容が変更されないものの場合，ファイルサイズやチェックサムなどを用いてファイルの変化を検出し，異常な変更が発生している場合に悪意のある書き換えかどうかを検査するという監視方法が考えられる．

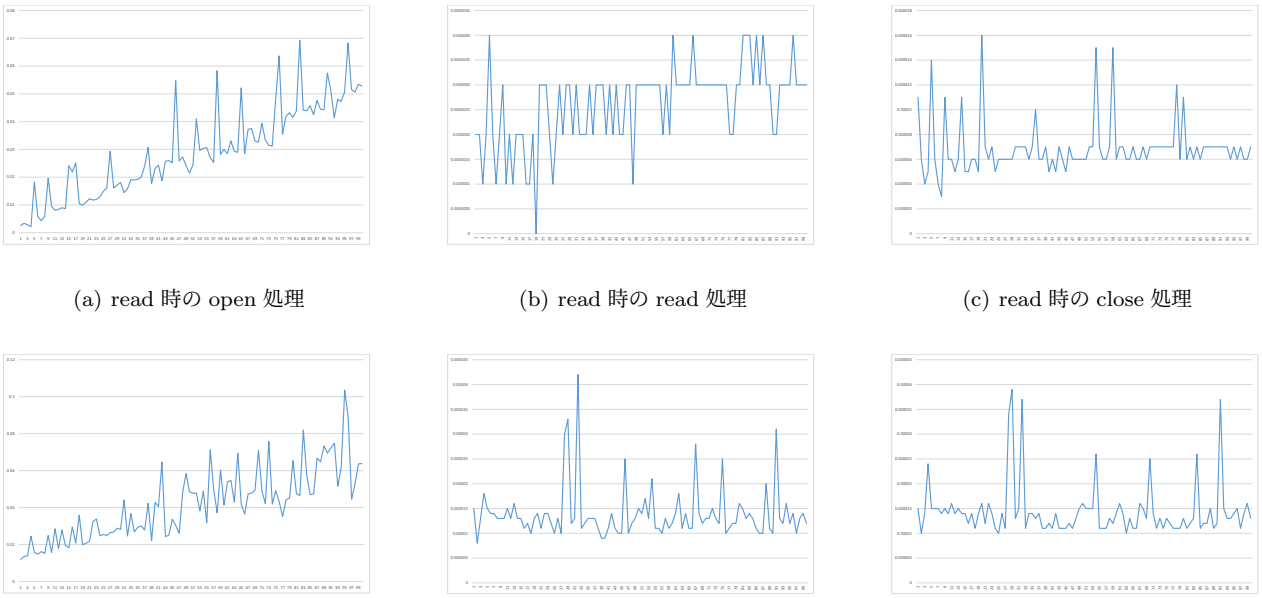
そこで，コンテナ内の書き込みレイヤを切り替える機能によって，コンテナ内のファイルの変更によるファイルサイズの変化をホスト環境上から監視できることを示す．1 秒ごとに 1〜1000 バイトをランダムで追加または削除する処理を行い，5 秒ごとに書き込み先の切り替え処理を行い，これを 150 秒間実行した．図 11 は切り替え処理ごとのファイルのサイズを計測した結果である．保存先切り替え処理ごとに切り替え前の書き込み先レイヤに切り替え時のファイルシステムの状態が残る．コンテナ内からは単一のファイルシステムとして見えるが，ホスト環境からは，複数のディレクトリとしてレイヤに分かれて保存される．このコンテナイメージ管理の特徴を活かして，ホスト環境上でレイヤをそれぞれ調べることによってファイルサイズの変化をホスト環境から監視することが可能となる．

6. 考察

第 5.1 章に示す実験結果によると，ノーマルモードのオーバーヘッドは，切り替え時に開いているファイルのサイズ，個数に関わらず 200 マイクロ秒程度であり，ほとんど影響はなかった．これは，新しくファイルを開いた時の切り替え処理が，root dentry の__upperdentry の切り替えのみで済むためである．それに対して，強制書き出しモードの場合，開いているファイルの個数が多くなるほど切り替えにかかる時間は増加した．この原因は，切り替え時に書き込み先をすべて切り替える処理は，開いているファイルがあった場合に新しい書き込み先にコピーアップ処理を行う必要があるためと考えられる．しかし，コピーアップ処理はローカルのコピー時間のため，ファイルサイズが 10MB のファイルを 100 個開いている場合の切り替え時間は 1.3 秒程度で実際に使用する際に大きな問題はないと考える．

本研究では，定期的な保存先の切り替え処理を行うことによって，ファイルの改竄検知とスナップショット機能を実現する．そのため，通常のコンテナの場合複数のベースイメージで構成されていても数個のレイヤ数によって構成されるが，この機能を用いる際は構成するレイヤ数が数百になることが考えられる．図 10 に示す実験より，レイヤ数によるオーバーヘッドに関する評価を行った．レイヤ数が増加した場合，一番下に設定されたベースイメージレイヤのファイルに対する操作を行う際に，open 処理にかかる時間が増加した．これは，open 処理時に行われる lookup 処理においてレイヤ数が増加するとたどるディレクトリの個数が増加するために処理に時間がかかるためである．しかし，レイヤ数が 100 個に増えた場合で 0.06 秒のため大きなオーバーヘッドではない．また，レイヤ数増加によるオーバーヘッドはあるファイルをそのレイヤにおいてはじめて open する時にのみ発生することから大きな問題にはならない．write 処理の場合，コピーアップが行われ，それ以降のファイルの操作は一番上に設定されている upperdir のレイヤに対して処理が行われ，read 処理の場合についても 2 度目以降の lookup 処理はキャッシュが使用される．

ファイルの改竄検知を実現するためには，ファイルサイズの変化を検知する方法や，ファイルのハッシュの比較を行う方法などがある．標準コマンドのようにコマンドを起動してからコマンドのバージョンアップするまでファイル内容が変わらないケースでは，ファイルサイズの変化を検知し，標準コマンドの改竄が行われているかを調べることができる．書き込み先を切り替える時にすべての書き込み先を切り替える方法を使用すると，切り替えを行った際のファイルシステムの状態を残して，切り替え以降とのファイルの変更内容を知ることが可能となる．図 11 に示した実験においては，ランダムでファイルの変更を行ったときに，定期的に書き込み先の切り替え処理を行っておくことによって，ファイルサイズの変更内容を知ることができることを示した．コンテナイメージ管理方法として，OverlayFS のほかに ZFS や LVM を使用した管理方法があり，これらは機能の一部としてスナップショットを提供している．しかし，これらはブロックデバイスレベルの管理を行っているため，ホスト環境上からコンテナファイルシステムの内容を知ることができない．OverlayFS を使用し



(a) read 時の open 処理　(b) read 時の read 処理　(c) read 時の close 処理

(d) write 時の open 処理　(e) write 時の read 処理　(f) write 時の close 処理

図 10　レイヤ数による処理時間の変化

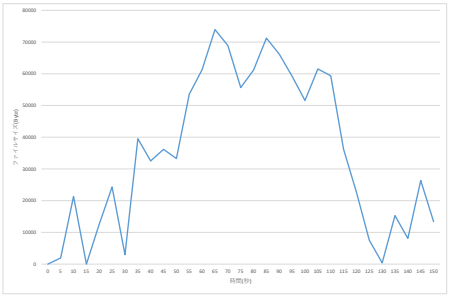


図 11　ファイルサイズの観測

てスナップショット機能の実現することによって，ホスト環境上からファイルの変更情報の検知を行うことが可能となり，ファイルの改ざん検知機能を実現した．また，書き込み先の切り替え処理を行うことによって，新しいディレクトリに書き込みが行われていくため，ホスト環境からは，特定レイヤに相当するディレクトリ内を見るだけで，ファイルの変更内容を検知することが可能である．ファイル改竄検知の方法として，Rootkit Hunter[7] のように事前に正常なファイルのハッシュ値をとっておき，現在のファイルのハッシュ値と比較することによってファイルが改竄されているかを比較する方法がある．ここでは，ハッシュ値をファイルシステム全体のファイルに対して行っていくことによってルートキットの検出を行うが，毎回ファイルシステム全体に実行すると実行時間がかかってしまうことや，ファイルシステム全体に対して実行すると誤検知がいくつか検出されてしまうという問題がある．これらの問題に対して，強制書き出しモードの場合，新しく設定された書き込み先レイヤにファイルの変更情報が書き込まれていく．そのため，ハッシュ値比較を行う必要のあるファイルが新しくファイルが変更されたものだけで済み実行時間の短縮を行うことが可能になる．また，新しくファイルの変更が行われたものだけで済むため誤検出が少なくなる点によって改善されると考えられる．

OverlayFS の切り替え後に新たに open された変更のみを新しいレイヤに記録し，それより前の更新は元のレイヤに残すと，ある切替時点の操作が正しく完了した状態を残すことになる．これにより，OverlayFS の構成を切り替え前に戻すことで，切り替え時点の安定したファイルシステムの状態に回復させることが可能である．その一方で，close されない限りはそのレイヤに変更が記録されないため，ファイルの改竄等をいち早く検出することは困難になる．これに対しては，切り替え時点の更新状態を全てレイヤに記録してしまう手法を提供した．この手法は，変更が即時に記録される利点はあるが，あるファイルに対してアプリケーションの操作をしている途中の状態が，切替を行う度に保存されてしまう．そのため，そのレイヤを見ると正常な更新が終了された時点なのかというのがわかりにくいという欠点もある．正常な更新完了かどうかがわかるようにファイルシステムレベルでの拡張が必要となる．

7. 結論

本研究では OverlayFS の remount 時の挙動を拡張し，書き込みレイヤを随時重ねることができるようにすることによって，スナップショット機能を実現した．この機能を Docker のコンテナイメージの保存において使用するこ

とで，Docker コンテナ内の編集ミス等からの回復や改竄の検出を容易に実現することが可能になった．ここでは，OverlayFS の切り替えにおいて，ノーマルモードと強制書き出しモードを実現した．これらの手法は，それぞれファイルシステムの状態復元とファイルシステムの改ざん検知に利用できるものである．提案手法に対する評価の結果として，切り替えによるオーバヘッドはほとんどなく，ホスト環境からコンテナ内のファイルの変更内容を検知できることを示した．

謝辞 本研究の一部は JSPS 科研費 JP18K11247 の助成を受けたものである．

参考文献

[1] Docker Inc: Docker - Build, Ship, and Run Any App, Anywhere, 入手先 ⟨https://www.docker.com/⟩ (accessed 2019-02-03).
[2] Neil Brown: Overlay Filesystem 入手先 ⟨https://www.kernel.org/doc/Documentation /filesystems/overlayfs.txt⟩ (accessed 2019-02-03).
[3] ZFS, 入手先 ⟨https://zfsonlinux.org/⟩ (accessed 2019-02-03).
[4] Menage, P.: CGROUPS, 入手先 ⟨https://www.kernel.org/doc/Documentation/cgroup-v1/⟩ (accessed 2019-02-03).
[5] Biederman, E. W. and Networx, L.: Multiple instances of the global linux namespaces, Proceedings of the Linux Symposium, Vol 1, Citeseer, pp. 101-112, (2006).
[6] Jan-Simon and Pendry Marshall and Kirk McKusick: Union mounts in 4.4BSD-lite, Proceedings of the USENIX 1995 Technical Conference Proceedings (TCON'95), (1995).
[7] M. Boelen: The Rootkit Hunter project, 入手先 ⟨http://rkhunter.sourceforge.net/⟩ (accessed 2019-02-03).