View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Tottori University Research Result Repository

オブジェクトストレージゲートウェイを用いたオブジェクトストレージの利用に関する評価

An evaluation of the use an object storage using an object storage gateway

本村真一†, 木本雅也†, 川戸聡也†

Shin-ichi Motomura†, Masaya Kimoto†, Toshiya Kawato†

motomura@tottori-u.ac.jp, kimoto@tottori-u.ac.jp, t.kawato@tottori-u.ac.jp

鳥取大学総合メディア基盤センター†
Center for Information Infrastructure & Multimedia, Tottori University†

概要

ビッグデータ、データ爆発という言葉に代表されるように、近年データ量の増加や重要度が増大している。そこで、大容量のデータを安価に保存するとともに、データ損失防止のため遠隔地へのバックアップが可能なクラウドストレージと呼ばれるサービスに注目が集まっている。多くのクラウドストレージはオブジェクトストレージにて構成されているが、オブジェクトストレージには汎用性の欠如、トランザクションの未サポートという問題がある。また、一般的にデータの転送速度は速くない。これらの問題への解決策として、オブジェクトストレージゲートウェイというサーバの利用が挙げられる。ここでは、オブジェクトストレージゲートウェイとオブジェクトストレージを組み合わせた利用について評価・検証を行った。

キーワード

ストレージ，クラウド

1　はじめに

ビッグデータ、データ爆発という言葉に代表されるように、近年データ量の増加が注目されている。大学という教育・研究機関においても、教育用コンテンツの増加や研究データの長期保存など、データ量、重要度ともに増加している。そのため、データを蓄積するストレージにおいては、大容量かつ安価であるとともに、データの損失防止や BCP の観点から、遠隔地へバックアップできることが求められている。このような需要に対応して、クラウドストレージと呼ばれるサービスが出現している。代表的なクラウドストレージとしては、Amazon Simple Storage Service（以下、「Amazon S3」という。）[1] や Windows Azure ストレージ [2] がある。これらのクラウドストレージ [3] の多くは、オブジェクトストレージ [4, 5] と呼ばれるシステムを用いて構築されている。これは、画像や動画などの非構造化データを安価に格納できるとともに、インターネットを介したアクセス、つまりは地理的に分散し遅延が発生する環境での使用に向いたストレージとして選択されているためである。現在主要なストレージである、ブロックストレージやファイルベースストレージは、SCSI や CIFS、NFS などの遅延の少ないイントラネット上での利用を想定しているのに対して、オブジェクトストレージは通常 HTTP プロトコルを利用している。大容量ストレージを低コストで構築できる特徴から注目を集めているオブジェクトストレージであるが、OS のファイルシステムやファイルサーバ等のストレージとして利用するためには 2 つの問題がある。1 つ目の問題は、ストレー

ジへアクセスするためのプロトコルがHTTPベースとなっているため、OSでのサポートが一般的ではなく汎用性に欠ける。アプリケーション側でのサポートが必要となり、CommVault社のバックアップアプリケーションSimpana[6]など、一部のアプリケーションにおいてはクラウドストレージに対応しているものもあるが、一般的とは言えない。２つ目の問題は、排他制御機能を持たないことが多く、トランザクションデータの取り扱いには不向きである。これらの問題への解決策として、オブジェクトストレージゲートウェイと呼ばれるシステムの導入が挙げられる。これは、ext4やNTFSなど、OSのファイルシステムとしてオブジェクトストレージをマウントできるようにするシステムである。OSのファイルシステムとして利用することで、排他制御機能はOSのものを利用し、またキャッシュ機能を提供することでパフォーマンスの向上も期待できる。

ここでは、オブジェクトストレージとオブジェクトストレージゲートウェイを組み合わせた利用について評価・検証を行う。

2 オブジェクトストレージとオブジェクトストレージゲートウェイ

オブジェクトストレージ、オブジェクトストレージゲートウェイともに厳密な定義はないが、一般的にオブジェクトストレージは、ブロックやファイルではなくオブジェクトを管理する。通常、ファイルをオブジェクトとして管理するが、オブジェクトに固有の識別子を付与することで、ファイルを階層的に管理するのとは異なり、フラットなアドレス空間で管理する。また、オブジェクトに様々な属性をメタデータとして付与できることが多い。ストレージへのアクセスのためにHTTPベースのインターフェースとして、REST(Representational State Transfer)インターフェースをサポートしており、PUTリクエストによるオブジェクトの生成、GETリクエストによるオブジェクトの読み込み、DELETEリクエストによるオブジェクトの削除、LISTリクエストによるオブジェクトの一覧表示などのメソッドが提供される。

オブジェクトストレージゲートウェイは、OSからみてブロックストレージもしくはファイルストレージとしてアクセスするためのゲートウェイである。簡易なものとして、FUSE[7]を用いたcloudfuse[8]やs3fs[9]がある。これらは、オブジェクトストレージをUnixのファイルシステムとしてマウントすることができる。また、AWS Storage Gateway[10]においては、クラウドストレージであるAmazon S3へのアクセスをiSCSIとして提供する。

表- 1: Swift 概要

サーバ名	役割
Auth Server	ユーザアクセスに対する認証・認可を担当する
Account Server	アカウントとコンテナの対応を管理する
Container Server	コンテナとオブジェクトの対応を管理する
Object Server	オブジェクトのデータの実体を管理する
Proxy Server	ユーザとAccount Server、Container Server、Object Serverの間に入り、通信処理を中継する

ここでは、オブジェクトストレージのオープンソース実装として定評のあるSwift[11]と、StorSimple社製StorSimple[12]をオブジェクトストレージゲートウェイとして取り上げる。

2.1 オブジェクトストレージ OpenStack-Swift

Swiftは、IaaS（Infrastructure as a Service）を構築するオープンソースのクラウド基盤ソフトウェアOpenStackのオブジェクトストレージを提供するソフトウェアであるが、もともとRackspace社のCloud Files[13]というクラウドストレージで使用されていたコードを基にオープンソース化しているため、安定性が高くSwift単体での採用実績も多い。Swiftの主要なサーバと役割を表1に示す。図1はクライアントおよびサーバ間の通信の流れを示しており、基本的にクライアントとの通信はProxy Serverが担当する。Proxy ServerはRESTインターフェースを提供しており、クライアントからのオブジェクトのアップロードや削除などをPUTやDELETEのリクエストとして受け取る。Swiftのオブジェクトはコンテナに管理されており、コンテナの名前空間はアカウントに管理されている。また、1つのアカウントにはProxy Serverへ接続するためのユーザを複数登録できる。この構造を図2に示す。Proxy Serverはクライアントからのリクエストに基づき、Account Server、Container Server、Object Serverとやり取りを行い、オブジェクトの登録や削除を実行する。

Swiftでは、単一障害点を排除するとともに、高可用性と拡張性の両立を図っている。Proxy Serverをプロキシノード、Account Server、Container Server、Object Serverをストレージノードと呼ぶが、アカウント、コンテナ、オブジェクトのいずれのデータも、複数のノードに自動的に複製することで可用性向上を図っている。ま

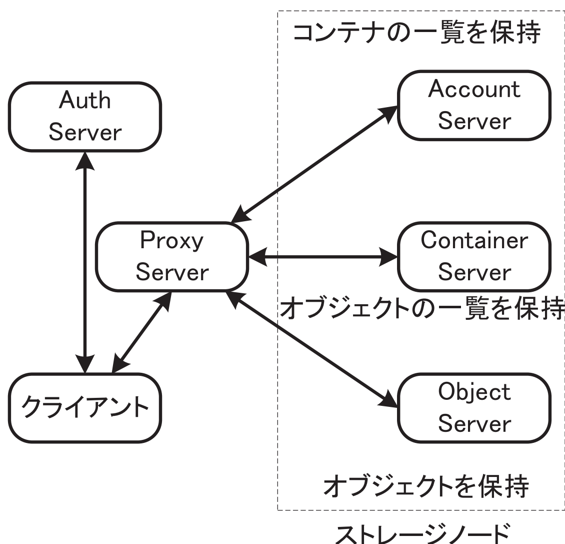

図- 1: Swift のアーキテクチャ

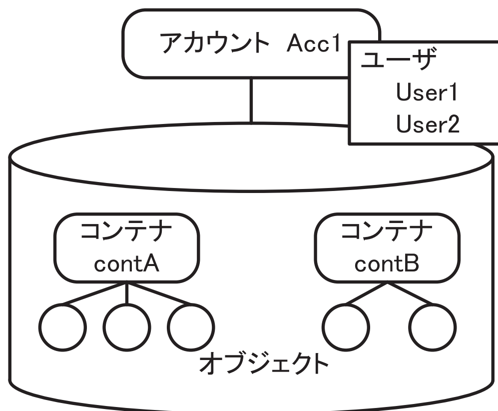

図- 2: Swift の論理構造

た、ストレージノードを追加することで簡単に容量の増加やパフォーマンスの向上が図れる。単一障害点となりうるプロキシノードにおいても、複数のプロキシノードを設置し、クライアントからのアクセスを分散させることができる。一般的には、DNS ラウンドロビンや負荷分散装置を用いて単一障害点の排除とパフォーマンスの向上を図る。

分散システムでは排他制御において問題になることが多いが、分散ファイルシステムである Swift においても、データの一貫性は基本的に結果整合性であることに注意する必要がある。例えば、更新直後のオブジェクトを参照する場合、更新前のデータを取得する可能性がある。

2.2 オブジェクトストレージゲートウェイ StorSimple

StorSimple はクラウドストレージへの接続を iSCSI として提供するオブジェクトストレージゲートウェイ

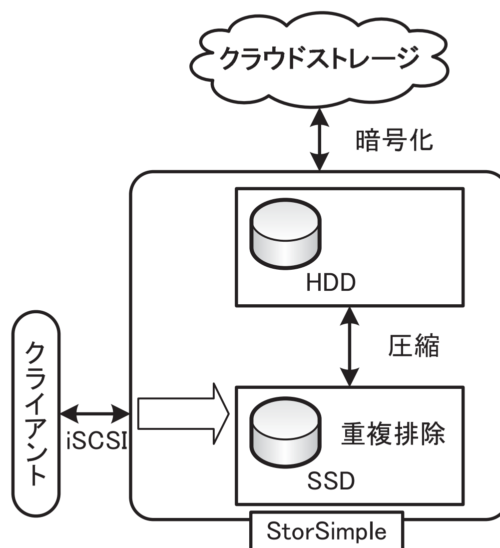

図- 3: StorSimple の動作概略図

である。基本的に Amazon S3 や Windows Azure ストレージなどのクラウドストレージを対象としているが、Swift への接続もサポートしているため、プライベートクラウドとして構築したオブジェクトストレージへの接続も可能である。StorSimple はクラウドストレージに対してのキャッシュの役割を果たすとともに、データの一貫性を提供する。StorSimple では、クライアントへのキャッシュ機能を効果的に発揮するため、ティアリングと呼ばれる自動階層化機能を提供している。図 3 は StorSimple の動作の概略を示しており、クライアントからのデータは最初に SSD へ渡されたあとデータブロック毎に重複排除が行われる。

その後、使用頻度が低いデータブロックは HDD に渡され圧縮される。さらに使用頻度が低いデータブロックはクラウドストレージへと渡される。ローカルの SSD 及び HDD（以下、「ローカルストレージ」という。）に存在しないデータをクライアントが要求した場合は、自動的にクラウドストレージから取得する。なお、クラウドストレージ上のデータの機密性を確保するために、自動的に暗号化及び復号を行う機能を持つ。

StorSimple はバックアップもクラウドストレージへ保存することができる。ローカルストレージは 2TB から 20TB 程度の容量しかないが、クラウドストレージを利用することで、全体の容量としては 100TB から 500TB の容量を扱うことができる。これだけの大容量になると、その保存先のストレージをどう確保するかという問題のみならず、障害や災害時のデータ損失の影響も大きくなる。そのため、遠隔地へ保存できることが望ましいが、StorSimple の場合、差分データや完全データを複数のクラウドストレージへ保存することができるため、データ損失の回避を図ることができる。

3　実験

3.1　Swift のパフォーマンス

実験環境では、バージョン 1.7.4 の Swift を用いた。プロキシノードとして 1 台の Proxy Server を仮想マシンとして構築し、OS には CentOS6 を用いた。仮想マシンは、HP 社製 Proliant BL460c G6（CPU：Xeon E5540 2.53GHz）をホストとし、VMware 社製 vSphere5.0 を用いて CPU を 1 個、メモリを 4GB 割り当てた。ストレージノードは、各サーバに Account Server、Container Server、Object Server を導入し、2 台のサーバを構築した。それぞれのサーバは、HP 社製 Proliant DL320e Gen8（CPU：Xeon E3-1200V2 3.10GHz、メモリ：4GB）を用いた。HDD は 7.2krpm の SATA を 4 台を用いて LVM にてボリュームを作成した。

実験環境における Swift のパフォーマンスを、Swift のパッケージに含まれる swift-bench というツールを用いて測定した。swift-bench は、オブジェクトサイズ、クライアントからの同時実行数、PUT 及び DELETE するオブジェクトの数、GET するオブジェクトの数をパラメータとして与え、1 秒間に処理できるオブジェクト数を計測する。ここでは、PUT 及び DELETE するオブジェクトの数を 1,000、GET するオブジェクトの数を 100 として実行した。図 4 は、オブジェクトサイズを 64KB に固定し、同時実行数を変化させたときのパフォーマンスの違いを示している。同時実行数を増やしても PUT 及び DELETE できるオブジェクトの数は大きく変化していない。これは、オブジェクトの書き込みに関するパフォーマンスがストレージノードに大きく依存しているためである。実験環境ではストレージノードを 2 台のサーバで運用しているため、全てのオブジェクトの書き込みは 2 台のサーバに対して行われる。Rackspace 社の経験によれば、データの消失を避けるためにオブジェクトの複製は 3 台以上のストレージノードに対して行うことが推奨されている [14]。ストレージノードの数よりサーバ台数が少なくなると、オブジェクト更新時に 1 台のサーバに複数のオブジェクトの更新要求が発生するため、書き込みのパフォーマンスを向上させるためにはストレージノードの数以上の台数でサーバを運用する必要がある。

GET リクエストにおいては、同時実行数の増加に合わせて減少している。これは、オブジェクトの読み込みに関するパフォーマンスがプロキシノードに大きく依存しているためであると考えられる。Swift は python で記述されており、python インタープリタの制限により各デーモンは 1CPU コアしか利用できないため、CPU コアを追加しない限り、起動するスレッド数を増やしてもプロキシノードのパフォーマンスは向上しない。パフォーマンス向上のためには、CPU コアの追加もしくはプロキシノードの追加を行う必要がある。この確認のために、プロキシノードの仮想マシンに CPU を 2 個割り当てた時の実験結果を図 5 に示す。PUT 及び DELETE リクエストにおいてはあまり変化がないが、GET リクエストにおいては 2 倍程度変化している。

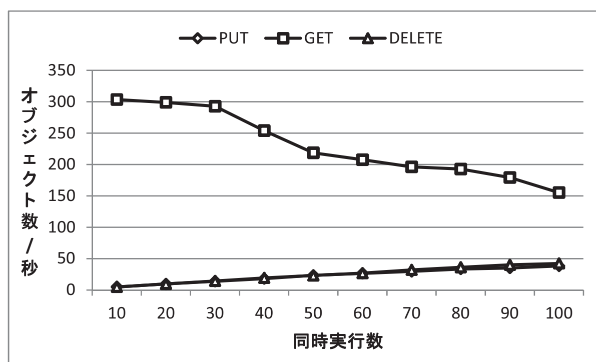

図- 4: 同時実行数による Swift のパフォーマンスの違い

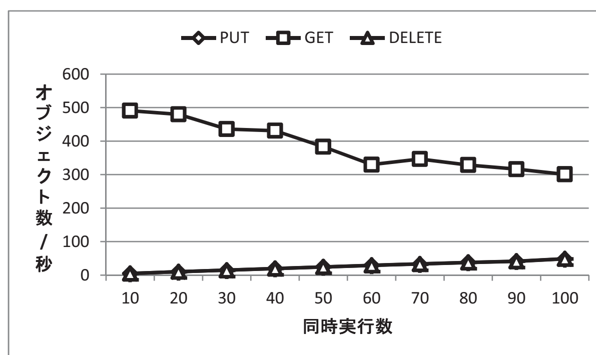

図- 5: 同時実行数による Swift のパフォーマンスの違い (2CPU の場合)

図 6 は、同時実行数を 100 とし、オブジェクトサイズを変化させたときのパフォーマンスの違いを示している。PUT 及び DELETE においてはほとんど変化がないが、GET においてはオブジェクトのサイズによって転送できるオブジェクトの数に大きな影響がある。このデータを転送速度として表したものが図 7 である。基本的にオブジェクトのサイズが大きいほど転送速度は速くなるが、オブジェクトのサイズはファイルサーバやメールストレージ等の利用するアプリケーションに依存しており、各アプリケーション毎に適切なオブジェクトサイズが異なる。そのため、アプリケーションによっては十分な転送速度が得られない場合がある。

3.2　StorSimple のローカルストレージのパフォーマンス

StorSimple は、400GB の SSD と 2TB の HDD（RAID1+0 の実効容量）を持った StorSimple5020 を

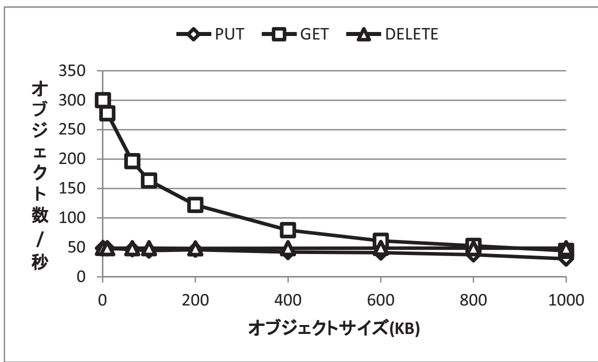

図- 6: ブロックサイズによる Swift のパフォーマンスの違い

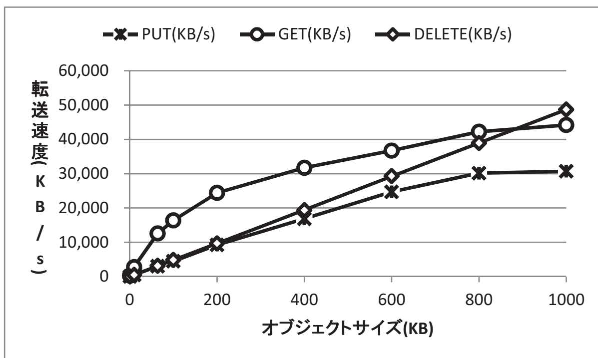

図- 7: ブロックサイズによる Swift のパフォーマンスの違い（転送速度）

用いた。StorSimple は Windows や Linux などからは iSCSI として認識される。ここでは、iSCSI としてマウントした際のローカルストレージのパフォーマンスを NFS 等のストレージと比較する。

同じ iSCSI ストレージとして、Buffalo 社製 TS-RIX4.0TL/R5 を用いた。TS-RIX4.0TL/R5 は 1TB の SATA HDD を 4 台搭載しており、RAID1+0 にて構成している。NFS については NetApp 社製 FAS3140 を用いた。FAS3140 は 300GB、15krpm の SAS HDD にて RAID-DP を構成している。ローカル HDD との比較のために、HP 社製 Proliant DL360 G6 を用いた。RAID コントローラとして Smart アレイ P410i を用いており、146GB、10krpm の SAS HDD 2 台にて RAID1 を構成している。

パフォーマンスの測定には fio[15] というツールを用いた。fio に与えるパラメータはブロックサイズを 64KB、ファイルサイズを 1GB、実行時間を 10 秒、実行スレッド数を 4 とし、ランダムリードとランダムライトを測定した。Proliant DL360 G6 以外の測定には、上述の vSphere 環境にて作成した仮想マシンを用いている。なお、OS はいずれも CentOS6 を用いた。表 2 はそれぞれ 5 回実行した時の平均値を表している。機器の構成が様々であり、また測定条件も簡易なものであるため目

表- 2: ストレージパフォーマンス比較

機器	ランダムリード (KB/s)	ランダムライト (KB/s)
StorSimple5020	75,115	22,776
TS-RIX4.0TL/R5	16,005	7,193
FAS3140	70,334	80,955
Proliant DL360 G6	39,082	81,355

表- 3: ローカルストレージと Swift のパフォーマンス比較

ファイルサイズ (MB)	ローカルストレージ (秒)	Swift (秒)	(予測) Swift (秒)
1	0.020	0.033	0.079
10	0.143	0.236	0.731
100	1.386	2.260	7.268
1,000	16.751	23.968	75.574

安でしかないが、StorSimple5020 のローカルストレージのパフォーマンスは他の機器と比較して読み込みが速く、書き込みが遅いと言える。これは、クライアントからの処理を受け付けるストレージが SSD であるためと思われる。

3.3 Swift と接続した StorSimple のパフォーマンス

StorSimple のローカルストレージに保存されているデータと、Swift 上に保存されているデータを取得する際のパフォーマンスの比較を試みた。StorSimple では、ローカルストレージに空き容量を確保するよう、独自のアルゴリズムでローカルストレージのデータを Swift へ転送している。そのため、ローカルストレージの容量に対して極端に大きなファイルを書き込まない限り、書き込みのパフォーマンスについては Swift の影響を受けることはない。そこで、StorSimple に作成したボリュームを、上述の vSphere 環境にて構築した仮想マシン（OS は CentOS6）からマウントし、StorSimple からファイルのコピーを行うことで、読み込み時間の比較を行った。なお、本実験では StorSimple から Swift へオブジェクトを転送する際に暗号化しない設定とした。

準備するファイルは、StorSimple の重複排除によりデータサイズが変更されないよう、/dev/urandom ファイルを用いて dd コマンドにて作成した。StorSimple には、ローカルストレージのオブジェクトをクラウドストレージへ転送し、ローカルストレージから削除するよう

な操作方法は提供されていない。そのため、次の方法でSwift からデータを読み込むよう試みた。

1. StorSimple に作成したボリューム A を Swift にバックアップする。
2. ボリューム A のバックアップをクローニングして新しいボリューム B を作成する。
3. ボリューム B をマウントする。

上述の方法により、StorSimple はボリューム A のメタデータを Swift から取得してボリューム B を作成する。この時点でボリューム B はマウントできるようになるが、データ自体は時間をかけてコピーされる。

ファイルサイズを変化させたときのファイルコピーに要する時間を表 3 に示す。なお、表の右列のデータ「(予測)Swift(秒)」については後程述べる。Swift から取得した際は、ローカルストレージから取得した時より時間を要しているが、図 7 より、オブジェクトサイズが 64KB、同時実行数が 100 の時の GET リクエストの転送速度は 12,500KB/s であることが分かっている。実験環境では、StorSimple が Swift へ転送するオブジェクトのサイズを 64KB に固定しており、1,000MB のファイルを Swift から取得するためには 80 秒程度要する計算となる。StorSimple から Swift への同時接続数については、設定やモニタリングの方法がないため不明であるが、図 4 から同時実行数を減らして転送できるオブジェクト数を増やしても、24 秒程度でコピーが完了するためには 3 倍程度転送オブジェクト数を増やす必要があり、これは無理であることが分かる。そのため、本実験では Swift からデータを取得していないと考えられる。StorSimple のローカルストレージのパフォーマンスの目安である表 2 の値から計算すると、13 秒程度となり、表 3 の Swift の値はローカルストレージから転送したものと考えられる。表 3 の値に違いがあるのは、StorSimple においてボリューム A とボリューム B のデータを同定するなどの処理に時間を要しているためであると考えられる。

上述の実験では、StorSimple におけるローカルストレージと Swift からのデータ取得にかかる時間の比較ができなかったため、StorSimple における Swift へのバックアップと Swift からのクローニング時の転送速度を比較する。表 4 は、上述の実験において約 1.1GB のファイルを配置したボリュームのバックアップとクローニングを行ったデータであり、StorSimple の管理画面上に表示されたものである。バックアップ処理については Swift の PUT リクエスト、クローニング処理については GET リクエストが相当する。図 7 から、実験環境における Swift の転送速度は PUT リクエストが 3,000KB/s、GET リクエストが 12,500KB/s 程度であると予測されて、表 4 の結果はこの数値に近いものとなっており、Swift 上にあるデータを StorSimple 経由で取得する際も 17,000KB/s 程度の転送速度が得られると考えられる。

表- 4: バックアップとクローニングに係るデータ転送速度

バックアップ速度（KB/s）	クローニング速度（KB/s）
1,343	17,000

GET リクエストの転送速度を 17,000KB/s と仮定した場合においての Swift のパフォーマンスを予測したものが表 3 の右列である。Swift からのコピー時間は、計算した転送時間とローカルストレージからコピーする時間を合算することで求めている。ファイルサイズが小さい状況では差が小さいが、ファイルサイズが大きくなるに従い差が大きくなり、Swift からデータを取得する影響が大きくなると予測できる。

4 まとめ

オブジェクトストレージゲートウェイを用いたオブジェクトストレージの利用に関する評価を目的として、StorSimple で作成したボリュームを Linux にてマウントし、StorSimple のローカルストレージと Swift からデータをコピーする時間の比較を試みた。StorSimple にはローカルストレージのデータをキャッシュアウトする適切な方法が存在しないため、比較の結果は予測を踏まえたものになったが、Swift のパフォーマンスに大きく依存するものとなり、また、扱うファイルサイズにも大きく影響を受けると言える。StorSimple では、長期間アクセスのないデータはローカルストレージからクラウドストレージへとキャッシュアウトされるため、今後の実験の手法として、測定対象外データを多量にローカルストレージへ書き込むとともに時間を置き、測定対象データがキャッシュアウトされるのを待って測定したいと考える。

実際の運用環境においては、Swift のパフォーマンスチューニングの実施や、利用するクラウドストレージのパフォーマンスの違いより、実験環境の Swift とは異なるパフォーマンスを示すと思われるが、いずれにしろオブジェクトストレージであることから同様の傾向になると考えられる。そのため、その利用方法には向き不向きがある。最も適していない利用方法は、大きいサイズのファイルを高速に読み書きするアプリケーションである。反対に適している利用方法としては、ファイルの読み書きに時間を要しても良いアプリケーションやサイズの小さいファイルを扱うアプリケーション、古

いファイルの読み込みがないアプリケーションなどが挙げられる。具体例としては、ログサーバのストレージや Dropbox のようなオンラインストレージ、バージョン管理システムのストレージなどが挙げられる。鳥取大学総合メディア基盤センターにおいては、ファイアウォールやネットワークスイッチ等のログを syslog にて管理しており、一日あたり 10GB 程度のログを記録しているが、データの転送速度としては 50KB/s 程度である。データの閲覧頻度は低く、また Splunk[16] というログ分析ソフトウェアによるログの集計も行っているが、Splunk で処理するデータは書き込み直後のデータであるため、StorSimple を用いた場合はローカルストレージから高速に読み込まれることになる。オンラインストレージにおいては、Windows の CIFS ボリュームや Unix の NFS と異なり、比較的タイムアウトの許容値を大きくできると考えられる。バージョン管理システムにおいては、古いバージョンのファイルも長期間保存するものと考えられるが、その参照頻度は低い。今後実際に、先の syslog のストレージとしての利用や、ownCloud[17] を用いたオンラインストレージサービスのストレージに Swift と StorSimple を利用して、その効果を検証したいと考えている。

参考文献

[1] Amazon Simple Storage Service . http://aws.amazon.com/jp/s3/,(参照 2013-06-30).

[2] Windows Azure . http://www.windowsazure.com/ja-jp/,(参照 2013-06-30).

[3] Jiyi Wu, Lingdi Ping, Xiaoping Ge, Ya Wang, and Jianqing Fu. Cloud storage as the infrastructure of cloud computing. In *"Intelligent Computing and Cognitive Informatics (ICICCI), 2010 International Conference on"*, pp. 380–383, 2010.

[4] M. Factor, K. Meth, D. Naor, O. Rodeh, and J. Satran. Object storage: the future building block for storage systems. In *"Local to Global Data Interoperability - Challenges and Technologies, 2005"*, pp. 119–123, 2005.

[5] M. Mesnier, G.R. Ganger, and E. Riedel. Object-based storage. In *"Communications Magazine, IEEE"*, Vol. 41, pp. 84–90, Aug 2005.

[6] Simpana . http://www.commvault.jp/simpana-software/,(参照 2013-06-30).

[7] FUSE . http://fuse.sourceforge.net/,(参照 2013-06-30).

[8] cloudfuse . http://redbo.github.io/cloudfuse/,(参照 2013-06-30).

[9] s3fs . http://code.google.com/p/s3fs/,(参照 2013-06-30).

[10] AWS Storage Gateway . http://aws.amazon.com/jp/storagegateway/,(参照 2013-06-30).

[11] Swift . http://swift.openstack.org/,(参照 2013-06-30).

[12] StorSimple . http://www.storsimple.com/,(参照 2013-06-30).

[13] RACKSPACE CLOUD FILES . http://www.rackspace.com/cloud/files/,(参照 2013-06-30).

[14] Swift Guide . http://docs.openstack.org/developer/swift/deployment_guide.html,(参照 2013-06-30).

[15] fio . http://freecode.com/projects/fio/,(参照 2013-06-30).

[16] Splunk . http://www.splunk.com/,(参照 2013-06-30).

[17] ownCloud . http://owncloud.org/,(参照 2013-06-30).