Vol.2012-HPC-136 No.21
2012/10/4

予約に基づくストレージQoS実現のための S3 REST APIの拡張実装

谷村 勇輔[1]　柳田 誠也[1,2]

概要：Amazon S3 互換のインタフェースを持つストレージはクラウド環境においてよく利用されるが，多数のサーバから共有されることが多く，性能の低下やばらつきの問題が存在する．これを解決するため，本研究では性能予約機能を持つストレージを S3 のバックエンドに適用することを試みる．その最初の段階として，S3 の REST API を拡張して性能予約に基づくアクセスインタフェースを実装した S3 のクライアントとサーバを開発した．さらに，バックエンドのストレージが提供する高スループットを S3 のデータ転送でも利用可能にするため，S3 の Multipart データ転送の並列化の実装にも工夫を施した．予備評価実験により，今回開発した S3 のクライアント・サーバではバックエンドのストレージの高スループットを利用可能であることと，同時アクセスが行われる場合に各クライアントが要求する性能を予約に基づいて提供可能であることを示した．

1. はじめに

仮想化されたサーバ等のコンピューティング環境を提供する IaaS（Infrastructure as a Service）型のクラウドにおいては，各ユーザの仮想マシンのイメージやバックアップ，大容量のアプリケーションデータなど，大きなデータを大量に格納できる高いスケーラビリティと常時運用に耐えうる可用性，頑健性を備えたストレージが必要である．例えば，Amazon EC2[1] では Amazon S3（Simple Storage Service）[2] をそうしたストレージとして利用できる．また，IaaS 型のクラウドサービス環境を構築するツールの 1 つとして注目を集めている OpenStack[3] では，同様のストレージとして Swift[4] を利用できる．Swift は独自のインタフェースだけでなく，S3 互換のインタフェースを有するオブジェクトストアの実装である．

しかし，これらのクラウド向けストレージの多くはデータアクセスのレイテンシが大きく，時間帯によってアクセス性能のばらつきが大きい問題を抱えている．これは**図 1** に示すように複数のアプリケーション間で共有されるためである．高く安定した性能を必要とする場合は，EBS（Elastic Block Storage）[5] のような従来的な Block Storage のインタフェースを有し，特定の性能レベルを提供するストレージを利用せざるを得ない．しかし，EBS のスケーラビリティを考えると，アクセス頻度が低く，サイズが大きい Object が主体のデータセットに EBS を用いるのは効率的ではない．S3 のようにデータは長期間にわたって安価に保存可能であり，必要な時だけ対価を払って，ある一定以上のアクセス性能の提供を受けられるストレージサービスがあることが望ましい．

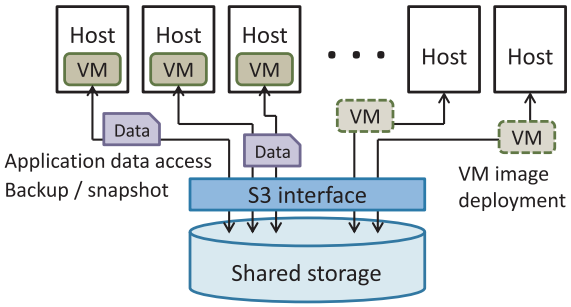

図 1　クラウド環境における大容量ストレージの共有利用の例

我々は分散ストレージにおいて，予約に基づいて性能を保証する研究を行っている [6], [7]．これまでに開発したプロトタイプシステム（Papio と呼ぶ）は，ユーザが明示的に性能を予約できるインタフェースを備え，予約に基づいてストレージ資源を適切に割り当て，アクセスを行うクライアントからデータを保持するストレージデバイスまでの I/O 性能制御を適切に行う機能を有する．Papio が S3 互換のインタフェースを提供できれば，この Papio の性能保証を S3 アプリケーションが享受できることになり，上記のようなサービスを実現できる．S3 インタフェースは Amazon 独自の規格ではあるが，S3 をサポートしているク

[1] 産業技術総合研究所
[2] 数理技研

表 1 Papio の予約機能に合わせて拡張を施した S3 の操作一覧

Operation	Description
PUT Bucket	Create a new bucket which ties to the space reservation of Papio
GET Object	Retrieve an object from the Papio storage system
PUT Object	Put an object to a specified bucket in the Papio storage system
Initiate Multipart Upload	Initiate a mulitipart upload

ラウドのツールは少なくなく，性能保証を広く適用できる可能性がある．

これを踏まえて，我々は Papio における S3 互換インタフェースの実装と予約に基づく性能保証のための拡張実装を試みた．本報告ではその実装の詳細と予備性能評価を行った結果について報告する．

2. 性能予約に対応するための S3 インタフェースの拡張

2.1 Amazon S3

Amazon S3 はインターネット上のどこからでもアクセス可能であり，Web Services に基づくアクセスインタフェースを提供する Amazon のオンライン・ストレージサービスである [2]．そのストレージ規模は年々拡大しており，2012 年には 1 兆個以上の Object を格納するに至っている [8]．現在，S3 では REST，SOAP のインタフェースがサポートされている．また，ダウンロードに関しては HTTP だけでなく BitTorrent も利用可能である．S3 の利用においては，ユーザは Bucket を作成し，Bucket の中にデータを Object として格納していく．1 つの Objcet の最大サイズは 5TB である．Object は内部で冗長化されて格納されており，Amazon は 99.99%以上の可用性を保証している．Amazon S3 の内部実装は公開されていないが，S3 互換インタフェースを実装したストレージやクライアントツールは多数存在する．

2.2 Papio とその性能予約機能

Papio は並列 I/O をサポートした分散オブジェクトストアであり [6], [7]，予約に基づいた性能とディスクスペースの管理機能を有する．アクセスインタフェースとしては Papio 独自のクライアント API ライブラリが提供され，それには S3 と類似の Bucket/Object のセマンティクスが実装されている．基本的に逐次アクセスを想定した設計になっており，Object へのアクセスが PUT と GET 操作主体の S3 と大きな差はない．S3 と明らかに異なる点は，Papio の Bucket がディスクスペースの予約と対応していることである．具体的には，Bucket を作成する際，ユーザは Bucket の生存期間，Bucket に格納される全 Object の合計サイズ（Bytes）を指定する．これにより，Papio のストレージ側では指定されたサイズのディスクスペースが確保される．性能予約は Bucket に対する Write，または Bucket 内に作成した Object に対する Read に関して行える．性能予約に用いる性能指標は現在のところスループット（MB/sec）のみサポートされている．性能予約の成立後，Papio は予約 ID をクライアントに返す．クライアントは予約時間内に Object にアクセスする際，この予約 ID を Papio のストレージに対して提示しなければならない．

2.3 S3 の拡張インタフェースの設計

本研究では S3 インタフェースを通して Papio を利用できるようにするため，Papio のクライアント API の上位に S3 互換インタフェースを実装する．ただし，本実装の目的は完全互換の S3 インタフェースを実装することではなく，あくまで S3 インタフェースを通じて Papio の性能予約・保証の機能を使えるようにすることである．そのため，予約に関係しない操作については S3 互換を維持し，予約に関する操作については S3 インタフェースを必要最小限の範囲で拡張することとした．

表 1 に Papio の予約に関連して拡張する S3 の操作一覧を示す．前節で述べたように，Papio の Bucket はディスクスペースの予約に対応するため，PUT Bucket の Request Element にスペース予約のパラメータ（サイズ, 開始終了時刻, アクセス時の基本性能）を設定できるようにする．それ以外の 3 つの操作はいずれも Object に対する I/O 操作である．Papio では Object のアクセスに際して予約 ID の提示が必要であるため，これらの操作要求の Request Header に X-Papio-Access-ID のパラメータ名で，予約 ID を設定できるようにする．なお，性能予約自体は Papio の予約ツールを使うことを想定し，性能予約操作は S3 のインタフェースに実装しない．

一方，S3 では HTTP サーバを介してデータ転送を行うため，Papio に直接アクセスした時に比べてオーバーヘッドが大きいことが予想され，事前評価でも我々が期待したスループットを十分に得ることができなかった．そこで，S3 のインタフェースを利用した場合でも Papio が提供する性能（スループット）を十分に享受できるように，Multipart 転送の実装を工夫する．Multipart 転送は 1 つの Object を分割して転送することで，スループットの向上や失敗した転送の回復を容易にする等の利点がある．今回，バックエンドの Papio ストレージと PapioS3 サーバ間では性能が保証された Papio の I/O ストリームでデータを転送し，PapioS3 のクライアント・サーバ間では Multipart

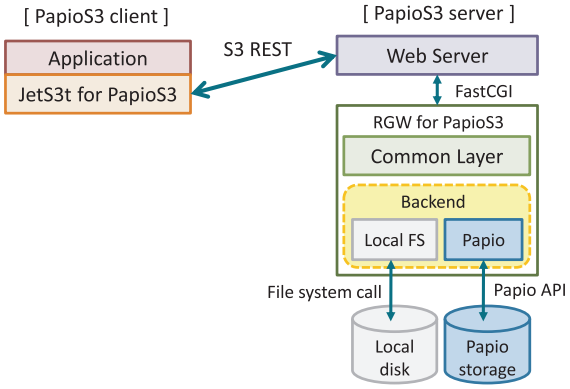

図 2 PapioS3 の実装の概要

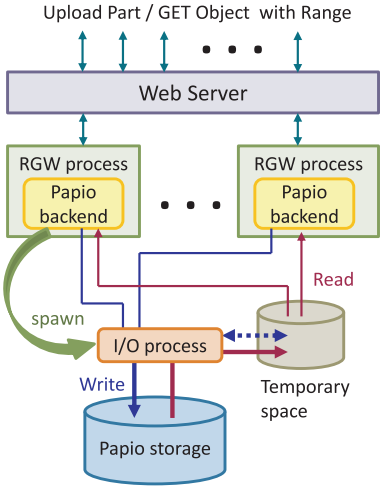

図 3 I/O プロセスの実装

の並列転送により，Papio のデータ転送性能にできる限り近づける．Multipart 転送の実装の詳細は 3.2 節で述べる．

3. 実装

3.1 概要

図 2 に Papio における S3 REST API[*1]の実装の概要を示す．これは Papio のフロントエンドとして拡張 S3 インタフェースを提供するサーバと，拡張 S3 インタフェースを用いてサーバに要求を行うクライアントからなり，本報告では PapioS3 と呼ぶ．PapioS3 サーバは，RADOS[9] 向けに S3 の REST API を実装した RADOS Gateway[10][*2](以降，RGW と記す．) をもとに開発した．RGW は CGI プログラムとして動作し，図 2 に示すように FastCGI を介して Web サーバと連携する．RGW では S3 の各要求を解釈し，バックエンドのストレージ API に適切にマップする仕組みが実装されている．今回の拡張ではこの RGW の構造は変更せず，バックエンドとして RADOS の代わりに Papio を用いるのに必要な修正を行った．

PapioS3 クライアントは S3 インタフェースを実装したオープンソースの Java のクライアントツールキットである JetS3t[11][*3]をもとに開発した．JetS3t 側の修正は主に Papio 用の拡張インタフェースの部分と Multipart データ転送に関する部分である．

以降では，S3 インタフェースにおいて高スループットを実現するための Multipart データ転送，ユーザ認証とアクセス制御，データの完全性の検証処理の実装について順に述べる．

3.2 Multipart データ転送

図 2 では予約により性能が保証されるのは，RGW の Papio バックエンドと Papio ストレージの間である．これに対して，PapioS3 の Multipart データ転送では，S3 のクライアント・サーバ間において並列ストリームを利用し，バックエンドの保証スループットに等しい性能を得られるようにするものである．この実現にはサーバ側において，各 RGW プロセスに同時に届くデータ要求をまとめ，Papio に対しては単一のアクセス要求に見せる必要がある．本実装では I/O プロセスを予約アクセス単位で起動して，データ要求の仲介を行う仕組みを採用した（**図 3**）．クライアント側では，JetS3t のオリジナルのコードでは未実装であった Multipart ダウンロードをサポートし，アップロードとダウンロードの両方において転送の際の分割サイズ（Part サイズ），および並列ストリーム数を任意に設定できるようにした[*4]．

3.2.1 サーバ側のアップロード処理

RGW プロセスは Multipart 処理の初期化要求（Initiate Multipart Upload）を受け取ると I/O プロセスを起動する．その後の Upload Part 要求では，I/O プロセスを経由して分割データを順に Papio に Write する．この時，分割データは概ね順番通りに各 RGW プロセスに届くことを想定している．各 RGW プロセスでは，もしデータが順序通りに到着せず，直前のデータが Papio ストレージに書き込まれていない場合には，Write 要求を I/O プロセスの待機リストに入れ，データ自体は一時領域に書き出す．I/O プロセスは直前のデータを書き込んだ後，一時領域からデータを読み出して，Papio ストレージに書き込む．なお，本実装では一時領域には高速アクセスが可能な RAM ディスク（/dev/shm）を用いる．RGW プロセスは Multipart 処理の終了要求（Complete Multipart Upload）または中断要求（Abort Multipart Upload）を受け取ると I/O プロセスを終了する．I/O 処理の途中でエラーが発生した場合には I/O プロセスが自動停止するように実装している．

[*1] 具体的には Amazon S3 API Reference Version 2006-03-01 に基づいている．
[*2] Ceph version 0.32 に含まれる RGW のコードを利用した．
[*3] JetS3t version 0.8.1a を利用した．
[*4] これらのうち，Multipart アップロードの並列ストリーム数についてはオリジナルの JetS3t でも任意に設定可能である．

3.2.2 サーバ側のダウンロード処理

Multipart ダウンロードではアップロードと異なり，S3 の操作としてクライアントが明示的に Multipart 転送の開始と終了をサーバに伝える仕組みがない．S3 クライアントが Request Header に Range パラメータを指定した要求を繰り返すだけであり，かといって Range パラメータがあってもサーバ側では Multipart ダウンロードであるかどうかは確定できない．そこで，本実装では Range パラメータがあった場合は，Multipart ダウンロードであるかどうかに関わらず，常に I/O プロセス経由で Read 要求を Papio に送るようにした．各 RGW プロセスは，以前の要求により既に I/O プロセスが起動されている場合は，その I/O プロセスを介して Read 要求を行い，そうでなければ新規に I/O プロセスを起動した上で Read 要求を行う．I/O プロセスは予約アクセス毎に起動し，各 RGW プロセスは Papio 用の S3 の拡張により導入した X-Papio-Access-ID を識別子として，各 GET 要求に対応する I/O プロセスを見分けるようにした．

I/O プロセスに届いた Read 要求はキューで管理される．I/O プロセスはキューから順に要求を取り出し，要求データを Papio から読み込んで一時領域に書き出す．一時領域への書き込みはバックグラウンドで行い，I/O プロセスはすぐに次のデータの読み込みを開始できるように実装している．各 RGW プロセスは一時領域へのデータの書き込み完了の通知を受けて，書き出されたデータを読み込み，PapioS3 のクライアントにデータを転送し，最後に一時領域上の該当データを削除する．

本実装では明示的な終了要求が RGW になされないため，PapioS3 クライアントのアクセス終了後も I/O プロセスが動作し続ける可能性がある．しかし，Papio は予約に基づいたアクセスであり，個々の予約には終了時刻が設定されている．そこで，予約終了時刻に到達すると I/O プロセスが自動的に終了するように実装している．

3.3 ユーザ認証とアクセス制御

RGW では，S3 のユーザアカウントは RGW のユーザ管理コマンドにより行う．新しくユーザを作成すると S3 の Access Key と Secret Key が作成され，S3 のユーザはこれらの Key を用いて S3 サーバ（RGW）にアクセスし，認証を受けることになる．Access Key と Secret Key を用いた認証の仕組み自体は Amazon S3 と同様である．ユーザ情報は元の RGW ではバックエンドのストレージを用いて管理されるが，本実装では Papio ではなく，RGW プロセスが動作するサーバのローカルファイルシステムを利用するように変更した．また，今回は S3 のユーザ ID と Papio のユーザ ID を 1 対 1 で対応させることとし，Papio は RGW によって認証されたユーザを信頼する仕組みとした．これについては，将来的にはユーザの対応表を持つことや鍵管理の方法を含む実装の見直しが必要であると考えている．

一方，現在の Papio の実装では，Bucket も Object もその所有者にしかアクセスを許可していない．すなわち，PapioS3 サーバがアクセスを全ユーザ，あるいは他の限定されたユーザに許可することは不可能であり，S3 で規定されている ACL のうち，PapioS3 は private（所有者だけが全権限を有する）のみをサポートしている．

3.4 データの完全性の検証処理

S3 では Object の PUT/GET においてデータの完全性を検証する仕組みがある．PUT Object ではクライアントは Object の MD5 を計算し，Request Header の Content-MD5 にその値を設定してサーバに送る．サーバ側では転送されたデータの MD5 を計算し，Content-MD5 の値と比較することでデータの完全性を検証する．Multipart データ転送では分割して送られるデータ毎に MD5 値の計算および検証がなされる．MD5 値は S3 サーバのバックエンドにおいて Object の属性情報として保存され，GET Object では ETag（Entity Tag）としてクライアントに送られる．クライアントは ETag を用いて，ダウンロードした Object の内容を検証できる．

PapioS3 の実装では PUT Object は上記の通りに処理が行われる．一方，GET Object においては ETag がクライアントに送られるものの，MD5 の計算を伴う Object の検証は行われない．PapioS3 を利用するアプリケーションのレベルで必要に応じて検証が行われることを想定しているためである．

Object とともに保存された ETag は，転送データの完全性を検証する以外にクライアントが Object の更新を確認するのにも利用可能である．S3 では GET Object において If-Match や If-None-Match を Request Header に指定し，変更があった場合のみダウンロードを実行する仕組みも用意されている．しかし，今回の実装では RGW がそれらの Request Header に未対応であるため，この機能を利用できない．

4. 予備性能評価

今回開発した PapioS3 の予備性能評価として，PapioS3 を用いたダウンロードとアップロードのスループット，および同時に複数のクライアントから単一の PapioS3 サーバに対して予約アクセスを行った場合の性能制御の効果について調査した．

評価には**表 2** のマシンを計 8 台用い，PapioS3 サーバに 1 台，PapioS3 クライアントに 5 台，バックエンドの Papio ストレージサーバに 2 台を割り当てた．なお，Papio にはストレージサーバの他に管理サーバが 1 つ必要であり，本実験では PapioS3 サーバと同じマシンで動作させた．そして，全マシンは 10 Gigabit Ethernet で接続した．

表 2　評価実験に用いたマシンのスペック

CPU	Intel Xeon E31230 3.2GHz (4 Cores)
Memory	8GB
Data disk	OCZ Vertex3 240GB (Connected by SATA 3.0)
OS	CentOS 6.2 (Kernel version 2.6.32)

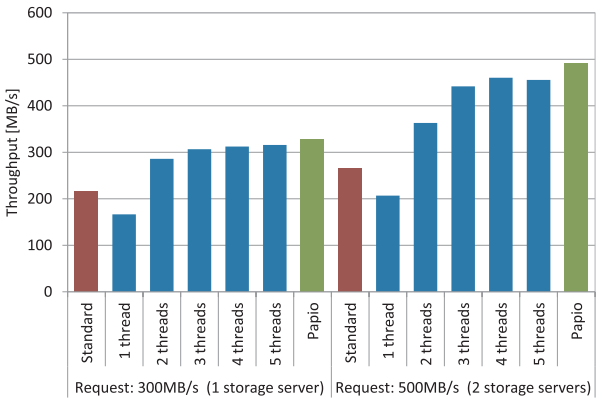

図 4　PapioS3 のダウンロード性能

PapioS3 サーバ側では，Web サーバとして CentOS 6.2 に含まれる Apache（version 2.2.15）を用い，FastCGI としては mod_fcgid（version 2.3.7）を用いた．

Papio ではストレージサーバの I/O スケジューリング間隔を 32MB 単位，ストライプサイズを 4MB に設定した．PapioS3 サーバでは Papio への I/O サイズを 32MB とし，PapioS3 クライアントの Multipart 転送時の分割サイズもそれに合わせて 32MB とした．JetS3t のデータ転送のスロットリングは無効にし，Multipart 転送では任意の並列度を指定して実験を行った．また，ダウンロードまたはアップロードする 1 つの Object のサイズは 1GB とした．以降に示す実験結果は全て 5 試行の平均値である．

4.1　データ転送性能

図 4 は PapioS3 のダウンロード性能の計測結果である．この実験では単一のクライアントを用いて PapioS3 サーバにアクセスした．Papio に対しては 300MB/s と 500MB/s の Read 性能予約を行った場合の 2 通りを試した．Papio のストレージ資源の自動割り当て機能により，バックエンドでは 300MB/s の予約に対しては 1 台，500MB/s の予約に対しては 2 台の Papio ストレージサーバが割り当てられた．図中の Papio の値は，PapioS3 サーバ上で Papio クライアントを動作させ，Papio ストレージサーバにアクセスした時の Papio 単体の計測結果であり，ほぼ予約した性能が得られていることが分かる．それに対して，S3 の基本操作である 1 回の GET Object でデータをダウンロードする場合（図中の Standard），その性能は予約した性能に比べてかなり低い．一方，Multipart 転送（図中の 1 thread〜5threads）では並列度を上げることで S3 のオーバーヘッドが隠蔽され，Papio 単体の性能の 9 割以上が得られている．

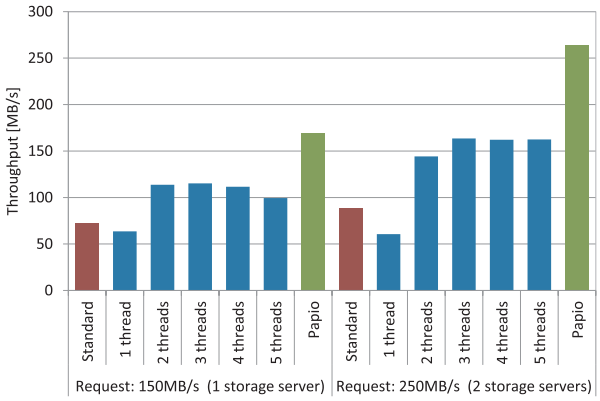

図 5　PapioS3 のアップロード性能

同様に**図 5** は PapioS3 のアップロード性能の計測結果である．この実験では Papio に対して 150MB/s と 250MB/s の Write 性能予約を行い，その結果，150MB/s の予約に対して 1 台，250MB/s の予約に対して 2 台の Papio ストレージサーバが割り当てられた．ダウンロードの性能評価実験と同様に，Papio 単体では予約した性能が得られる一方，1 回の PUT Object でデータをアップロードする場合（図中の Standard）には予約性能に比べてかなり低い値しか得られなかった．注目すべき Multipart データ転送では並列度を上げることでスループットを上昇させることができたものの，Papio 単体で得られた性能の 6 割程度が限界となった．この原因は I/O プロセス内の処理において，Papio ストレージサーバに書き込みを行う以外の部分でシリアライズされた処理が残っており，高スループット時にその影響が顕在化したためだと考えている．今後，I/O プロセスの挙動に関してより詳細な性能調査を行い，実装の改善を進める予定である．

4.2　同時アクセス時の性能

図 6 と**図 7** は 5 つの PapioS3 クライアント（A〜E）をそれぞれ別のマシンで起動し，PapioS3 サーバに同時にアクセスした時の各クライアントが得た性能を示している．この実験では Papio ストレージサーバを 1 台のみに限定し，必ず Papio の性能制御機能により，各アクセスの I/O 要求が制御される状況とした．Papio に対する性能要求（予約性能）の合計値は一定とし，ダウンロード（Read）では 300MB/s，アップロード（Write）では 150MB/s とした．そして，いずれにおいても各クライアントが要求する性能比率を 5:4:3:2:1 とした．また，各クライアントは Multipart 転送を有効にし，それぞれ 3 並列でアクセスを行うように設定した．

図 6 と図 7 より，ダウンロードとアップロードの両方において個々の PapioS3 クライアントが要求性能（Request）と同等以上の性能（Achieved）を得ているのが分かる．この結果は Papio の予約に基づいた性能制御を S3 クライア

図 6 5 クライアントによる同時ダウンロード時の性能制御

図 7 5 クライアントによる同時アップロード時の性能制御

ントが利用でき，また 1 つの PapioS3 サーバが少なくとも 5 クライアントの同時アクセスを扱えたことを示している．

5. まとめと今後に向けて

本報告では Amazon S3 の REST API を拡張実装したクライアントとサーバ（PapioS3）の設計と実装について述べた．PapioS3 の拡張 API は，予約に基づいて性能制御を行う機能を持つ Papio を S3 のバックエンドに用いることを想定し，主に Papio への予約 ID を PUT Object や GET Object 等の I/O 操作に埋め込む部分に関するものである．そして，予備評価実験により，まず PapioS3 の Multipart の並列転送を活用して，S3 のクライアント・サーバ間のデータ転送をバックエンドのストレージの性能に近づけられることを示した．その上で，複数の PapioS3 クライアントを同時に起動して，PapioS3 が Papio の性能制御機能を有効に活用できることを示した．

今後は Multipart データ転送におけるアップロード性能がバックエンドの性能に十分に届いていない問題を改善した上で，より大規模な環境での実験を進める予定である．さらに，実際的なクラウドのアプリケーションに対して PapioS3 の適用を行い，S3 における I/O 性能予約の有効性や既存の S3 アプリケーションとの互換性の問題等を検証していきたい．

謝辞 本研究の一部は日本学術振興会科学研究費補助金（23680004）の助成によるものである．

参考文献

[1] Amazon EC2: http://aws.amazon.com/ec2/.
[2] Amazon S3: http://aws.amazon.com/s3/.
[3] OpenStack: http://www.openstack.org/.
[4] Swift: http://swift.openstack.org/.
[5] Amazon EBS: http://aws.amazon.com/ebs/.
[6] Tanimura, Y., Koie, H., Kudoh, T., Kojima, I. and Tanaka, Y.: A Distributed Storage System Allowing Application Users to Reserve I/O Performance in Advance for Achieving SLA, *Proceedings of the 11th ACM/IEEE International Conference on Grid Computing*, pp. 193–200 (2010).
[7] 谷村勇輔，鯉江英隆，工藤知宏，　小島功，田中良夫：ユーザによる明示的な予約に基づき I/O 性能を保証する分散ストレージシステム，情報処理学会論文誌コンピューティングシステム，Vol. 5, No. 3, pp. 42–56 (2012).
[8] Amazon S3 – The First Trillion Objects: http://aws.typepad.com/aws/2012/06/amazon-s3-the-first-trillion-objects.html.
[9] Weil, S. A., Leung, A. W., Brandt, S. A. and Maltzahn, C.: RAODS: A Scalable, Reliable Storage Service for Petabyte-scale Storage Clusters, *Proceedings of the 2nd International Workshop on Petascale Data Storage*, pp. 35–44 (2007).
[10] RADOS Gateway: http://ceph.com/wiki/RADOS_Gateway.
[11] JetS3t: http://jets3t.s3.amazonaws.com/index.html.