Vol.2018-DBS-168 No.9
2018/12/22

# オブジェクトストレージの性能の解析に関する一考察

早川峻平[†1]　山口実靖[†1]

**概要**：近年，電子メールや画像などの非構造化データの量が増大してきている．膨大な非構造化データを管理するシステムとしてオブジェクトストレージがあり注目を集めている．オブジェクトストレージは，データをオブジェクト単位で扱い，ディレクトリのような階層構造を用いず個々のデータを URI を用いて管理する．ただし，多くのオブジェクトストレージシステムは複数のノードで構成される複雑な分散システムであり，その動作の把握や性能の考察は容易でないと考えられる．本稿では，ベンチマークによるオブジェクトストレージの性能の評価を行い，オブジェクトストレージの性能劣化原因の解析手法についての考察を行う．

**キーワード**：オブジェクトストレージ，SWIFT，TCP

## 1. はじめに

ストレージに保存するデータ量の増加とデータの性質の変化に伴い，これまでとは異なるデータ保存方法への要求が高まっている．ブロックストレージは複数計算機でのデータの共有などが困難であり，ファイルストレージのディレクトリ構造では膨大なデータの維持や管理が難しくなってきている．この問題を解決するために，データをファイルやブロックではなくオブジェクトとして扱うオブジェクトストレージが提案された．オブジェクトストレージはディレクトリ構造を持たず，オブジェクトは URI が付与されデータとそのメタデータで構成される．オブジェクトストレージでは，ファイルストレージにおけるディレクトリと類似したコンテナを作成し，そのコンテナに対し HTTP/REST API を用いてファイルのアップロードおよびダウンロードを行う．

多くの場合，オブジェクトストレージはストレージや認証ノードなどの複数のノードで構成される分散システムであり，その動作の把握や性能劣化原因の特定は困難になると考えられる．本論文では，OpenStack Swift [1][9]を用いてオブジェクトストレージシステムを構築し，通信の解析によるシステムの振舞の観察手法を提案する．そして，実際にシステムの動作を観察し，その有効性について考察する．

## 2. ネットワークストレージ

データバックアップなどのストレージ管理は企業などの組織にとって重要な事項であるが，計算機に直接接続されたストレージ(DAS, Direct Attached Storage)を用いている場合はそれぞれを個別に管理するのに膨大なコストがかかるという問題がある．この問題を解決するために，NAS (Network Attached Storage)や SAN (Storage Area Network)が提案され，ストレージが一元化され集中的に管理できる様になった[2]. NAS はファイルレベルのストレージアクセスを，SAN はブロックレベルのストレージアクセスを提供する．しかし，これらを用いてもデータはディレクトリ構造を持つファイルシステムで管理され，膨大な数のデータを管理することは困難であり，さらなる改善が期待される様になった．

## 3. OpenStack Swift

OpenStack Swift はクラウド環境を構築するためのソフトウェア開発プロジェクトである OpenStack のコンポーネントの一つである．Swift は図 1 の様に認証(Auth)ノード，プロキシノード，アカウントノード，コンテナノード，オブジェクトノードの複数のノードによって構成される．各ノードはそれぞれ，ストレージにアクセスするための API の提供や各サービスの管理，アカウント情報の管理，コンテナ情報の管理, オブジェクトの管理の機能を提供している．具体的には，認証ノードはユーザ名やパスワードを管理する．アカウントノードはコンテナ名の一覧やアカウント全体の統計情報などを管理し，コンテナノードはオブジェクト名の一覧や ACL 情報などの管理を行なっている．アカウントノードとコンテナノードは情報を管理するために SQLite3 データベースを用いている．オブジェクトノードはオブジェクトの保存を行う. オブジェクトストレージは，アップロードされたオブジェクトのレプリカを作成し分散して保存することにより，保存されたデータの安全性を向上する機能を有している．

オブジェクトのアップロードは図 1 に示す手順により行われる．図は，オブジェクトのアップロードを行うコンテナはアカウント内に既に存在し，アップロードしようとしているオブジェクトはコンテナ内に存在しない例となっている．まず，クラアイアントから認証ノードにトークン発行の要求が送られる（図 1 の(1)）．要求を受け取った認証ノードは，アップロードを行う際に使用するプロキシノードの URL をプロキシノードに要求する（図 1 の(2)）．要求を受け取ったプロキシノードは，自身の URL を認証ノードに送信する（図 1 の(3)）．プロキシノードの URL を受け取った認証ノードは，トークンと URL をクライアントに

†1 工学院大学
Kogakuin University

送信する（図 1 の(4)）．クライアントは，受け取ったトークンと URL を使用してプロキシノードに対してアップロード要求を送る（図 1 の(5)）．要求を受け取ったプロキシノードは，アカウントノードに対してアップロードするためのコンテナが存在するかどうかの確認要求を送る（図 1 の(6)）．対象のコンテナが存在した場合，アカウントノードは，対象のコンテナが存在したことをプロキシノードに知らせる（図 1 の(7)）．コンテナが存在したという情報を受け取ったプロキシノードは，コンテナノードに対してアップロードしたいオブジェクトが存在しているかどうかの確認要求を行う（図 1 の(8)）．対象のオブジェクトが存在しなかった場合，コンテナノードは対象のオブジェクトが存在しなかったことをプロキシノードに知らせる（図 1 の(9)）．オブジェクトが存在しなかったという情報を受け取ったプロキシノードは，オブジェクトノードに対してオブジェクトのアップロード要求を送る（図 1 の(10)）．要求を受け取ったオブジェクトノードは，オブジェクトを格納し，格納が成功したことをプロキシノードに知らせる（図 1 の(11)）．成功のレスポンスを受け取ったプロキシノードは，クライアントに対してアップロード成功のレスポンスを送信する（図 1 の(12)）．

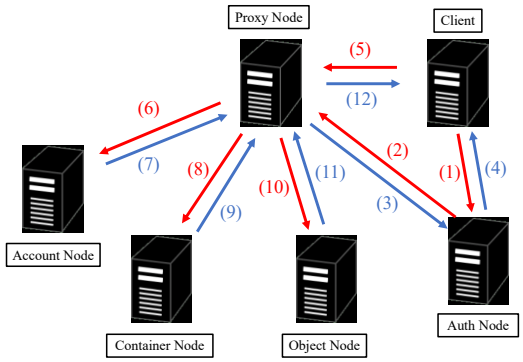


図 1　オブジェクトアップロードのフロー

この様に，オブジェクトストレージは多数の計算機による構成される複雑な分散システムとなっている．また，オブジェクトの操作も多数のノードが連携して行う複雑な処理となっている．よって，オブジェクトストレージシステムの動作の把握や性能劣化原因の発見は困難であり，これを容易にすることが重要であると考えられる．

## 4. 関連研究

複数の計算機で構成される分散システムの動作管理システムとして，我々は過去にネットワークストレージシステムの統合トレースシステム[4][5]や，OpenFlow[6]ネットワークにおける動的な転送制御情報の修正の観察システム[7]の提案を行っている．これらのシステムでは，計算機間で転送されるパケットを記録し，それらパケットの送信機と受信機における送受信時刻を調査し可視化することにより，システム全体の振舞の観察を可能にしている．ただし，これらの既存研究ではオブジェクトストレージシステムの様な多数の計算機で構成される複雑なシステムへの適用はされていない．また，本稿で提案される観察手法は，これら既存手法をもとにしており，これらを大規模なオブジェクトストレージシステムへ適用拡張したものである．

オブジェクトストレージシステムの性能に関する研究としては，ベンチマークについての研究[8]などがあるが，解析手法に関する考察などはされていない．

## 5. 性能評価

本章にて，Swift におけるオブジェクトアップロード性能の評価を行う．図 2 に測定環境を，表 1 に測定システムの仕様を示す．ストレージノード，プロキシノード，クライアントの 3 台の物理マシンが使用され，ストレージノード上では認証サーバープロセス，アカウントサーバープロセス，コンテナサーバープロセス，オブジェクトサーバープロセスが，プロキシノード上ではプロキシサーバープロセスが実行されている．オブジェクトノードにおけるオブジェクトを格納するストレージデバイスとしては，HDD あるいは SSD を使用した．

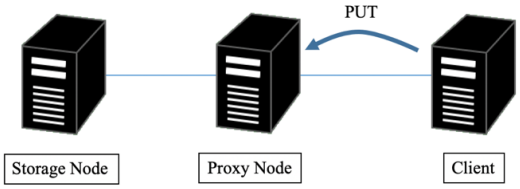


図 2　測定環境

表 1　測定環境の詳細

| | Storage Node | Proxy Node | Client |
|---|---|---|---|
| OS | Ubuntu 16.04 | | |
| CPU | Intel Core i5-4670 3.40GHz | Intel Celeron 2.27GHz×2 | AMD Athlon(tm) II X2 220 Processor |
| Memory | 8GB | 12GB | 2GB |
| Storage | HDD 3TB SSD 512GB | HDD 250GB | HDD 250GB |
| OpenStack Version | Pike | | |
| File System | xfs | | |

性能評価は，Swift のベンチマークツールである ssbench [3]を用いて行った．実験ではオブジェクトのアップロードを行い，Client から Storage Node に PUT リクエストを送信

することにより行った．アップロードするオブジェクトのサイズは 1B，1KB，1MB であり，それぞれ 2048 個のオブジェクトのアップロードを行った. 並列数は 1 から 2048 とした．

図 3，図 4，図 5 に HDD に各サイズのオブジェクトをアップロードした場合の並列数とオブジェクトアップロードスループットの関係を，図 6，図 7，図 8 に SSD にアップロードした場合の並列数とスループットを示す．図 3 から図 8 の横軸は並列数，左縦軸はスループット(単位は requests/sec)，右縦軸は失敗数とリトライ数(単位は回)である．

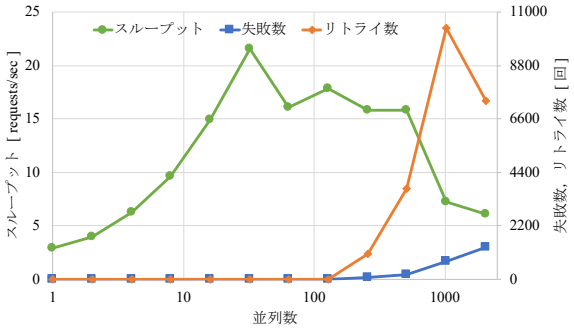


図 3　HDD における 1B オブジェクトをアップロードした際のスループット

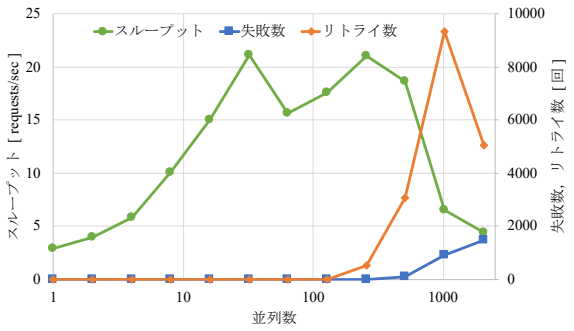


図 4　HDD における 1KB オブジェクトをアップロードした際の並列数に対するスループット

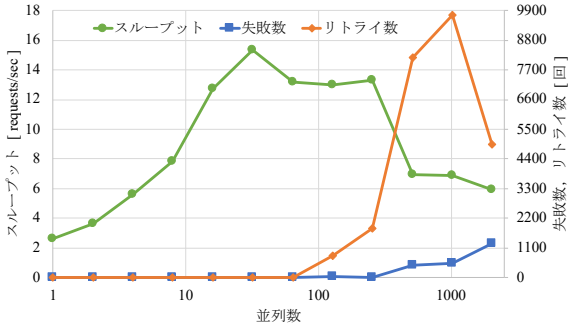


図 5　HDD における 1MB オブジェクトをアップロードした際のスループット

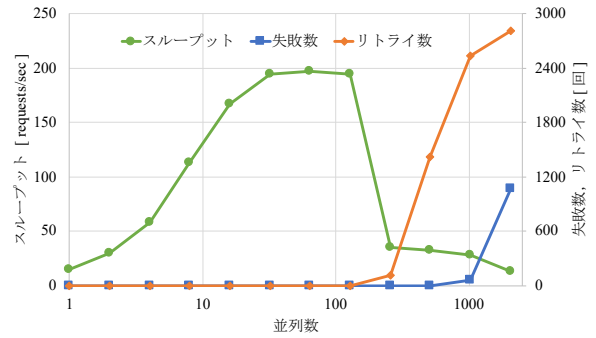


図 6　SSD における 1B オブジェクトをアップロードした際のスループット

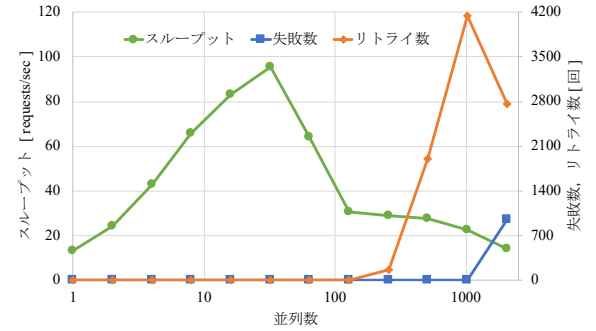


図 7　SSD における 1KB オブジェクトをアップロードした際のスループット

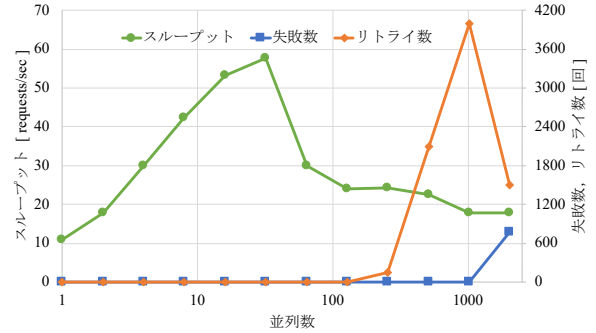


図 8　SSD における 1MB オブジェクトをアップロードした際のスループット

HDD でベンチマークを行った場合に着目すると，ある程度までは並列数を増加させるとスループットが増加し，それ以上に並列数を増加させるとスループットが低下し，過度に増加させるとリトライ数や失敗数が増加しスループットが大きく低下することが分かる．具体的には，並列数が 32 までは並列数の増加に伴いスループットが上昇し，並列数が 32 を超えるとスループットが低下している．そして，128 を超えるとリトライ数が増加し，256 を超えると失敗数が増加することがわかる．

SSD でベンチマークを行った場合に着目すると，HDD でベンチマークを行った際よりも最大スループットが大幅に高いことが分かる．また，HDD で行った場合と同様の傾向

がみられ，並列数を増加させるとある程度まではスループットが上昇し，それ以降はスループットの低下とリトライ数と失敗数の増加がみられる．具体的には，オブジェクトサイズが 1KB の場合を除いて，並列数が 32 までは並列数の増加に伴いスループットが上昇し，128 まではスループットが安定している．また，1KB の場合は並列数が 32 を超えると，1B と 1MB の場合は並列数が 128 を超えると，スループットが大幅に低下している．並列数が 128 を超えるとリトライ数が増加し，1024 を超えると失敗数が増加している．

次に，図 9，図 10，図 11 に HDD を用いてベンチマークを行った際の各マシンの CPU 使用率と I/O 使用率を，図 12，図 13，図 14 に SSD を用いてベンチマークを行った際の各マシンの I/O 使用率と CPU 使用率を示す．

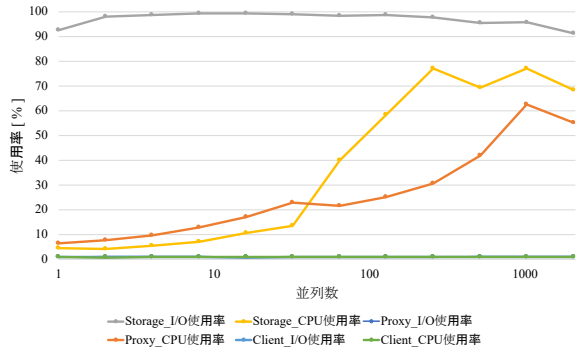


図 9　HDD における 1B オブジェクトをアップロードした際の各マシンの I/O 使用率と CPU 使用率

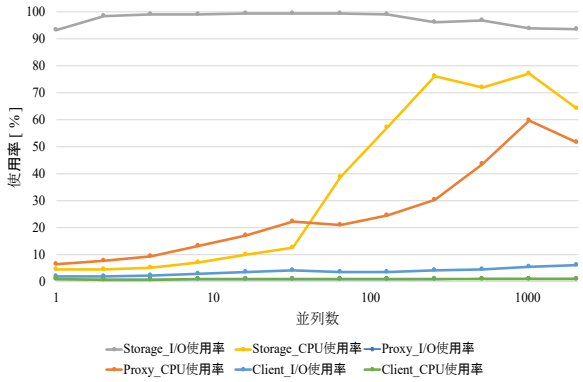


図 10　HDD における 1KB オブジェクトをアップロードした際の各マシンの I/O 使用率と CPU 使用率

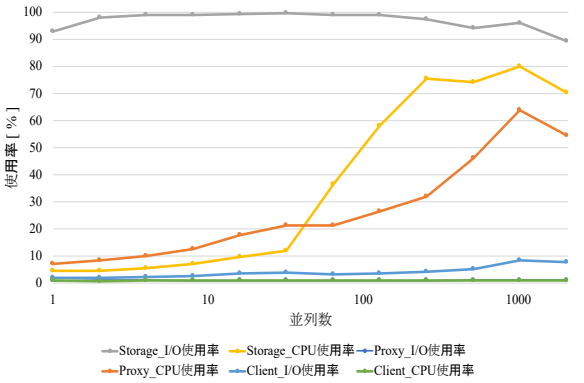


図 11　HDD における 1MB オブジェクトをアップロードした際の各マシンの I/O 使用率と CPU 使用率

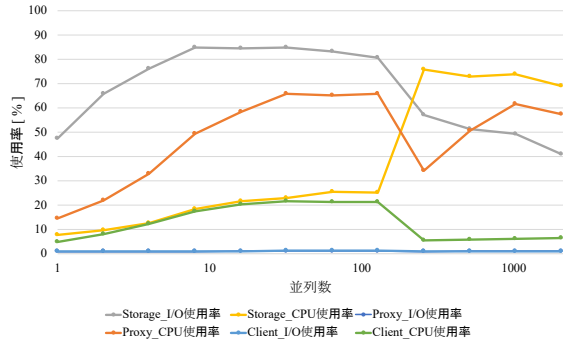


図 12　SSD における 1B オブジェクトをアップロードした際の各マシンの I/O 使用率と CPU 使用率

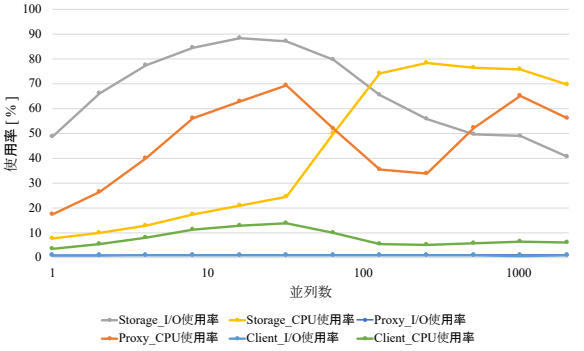


図 13　SSD における 1KB オブジェクトをアップロードした際の各マシンの I/O 使用率と CPU 使用率

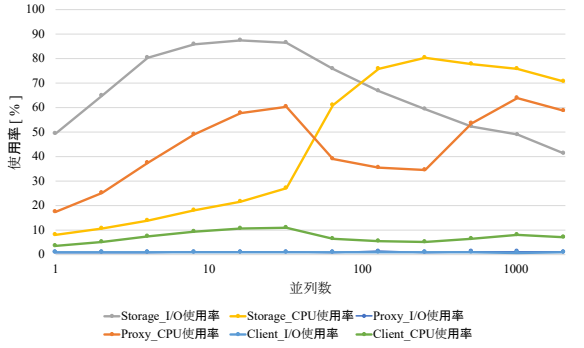


図 14　SSD における 1MB オブジェクトをアップロードした際の各マシンの I/O 使用率と CPU 使用率

図 9 から図 14 の横軸は並列数，縦軸は使用率であり単位は%である．HDD の場合は並列数に関わらず Storage Node の I/O 使用率が 90%以上と高い値となっていることがわかる．Storage Node と Proxy Node の CPU 使用率は並列数の増加に伴い増加しているが，本稿の実験の範囲においては高い並列度であっても Storage Node の I/O 使用率を超えることはなく，いずれの状態であっても Storage Node の I/O 処理がボトルネック(性能決定要因)となっていることがわかる．Proxy Node の I//O 使用率，Client の I/O 使用率と CPU 使用率は 10%以下であり，低い値であることがわかる．

SSD の場合は，並列度が低い状況では Storage Node の I/O 使用率が最も高く，並列数の増加に伴い増加するが，HDD の場合と比べて最大 I/O 使用率は低くなっている．また，並列数が大きくなると Storage Node の I/O 使用率が低下することがわかる．これは主にアップロードが失敗する様になるからであると考えられる．Proxy Node の CPU 使用率は並列数が高い状況では高く，並列度が極めて高い状況では Storage Node の I/O 使用率を超えている．最も高い使用率に着目すると，並列度が低い状況(1B にて 128 以下，1KB や 1MB で 32 以下)では Storage Node の I/O 使用率が最も高く，HDD 使用時と同様にストレージノードにおけるオブジェクトの書き込みがボトルネック(性能に支配的な要因)となっているが，並列度が高くなるにともに I/O 使用率が低下し，Proxy Node の CPU 使用率が増加している．これは，ssbench が Storage Node への書き込みを失敗しているためであると考えられる．

## 6. 観察手法

前述の様に，オブジェクトストレージはボトルネック箇所や性能劣化原因の考察が困難であると考えられる．この課題に対して本稿では，計算機間のパケットの送出と受信の時刻を記録，可視化し，システム全体の動作を俯瞰する手法を提案する．

提案手法ではまず，全計算機(Client，Storage Node，Proxy Node)にて，パケットの送受信を記録する．記録する情報は，送受信時刻，コネクション ID(ソース IP アドレス，ディスティネーション IP アドレス，ソースポート番号，ディスティネーションポート番号)，TCP シーケンス番号である．次に，各パケットの送信機および受信機における送信時刻および受信時刻をパケット送受信記録より特定する．そして，それらを線により結び図示，可視化する．パケットはコネクション ID とシーケンス番号により一意に特定できるため，上記情報により，あるパケットの送受信両機における対応を特定することができる．

ssbench を用いて 2 つのオブジェクトがアップロードされる動作を図 15 に，Swift API を用いて 1 つのオブジェクトがアップロードされる動作を図 16 に示す．とともに，アップロード並列数は 1 である．

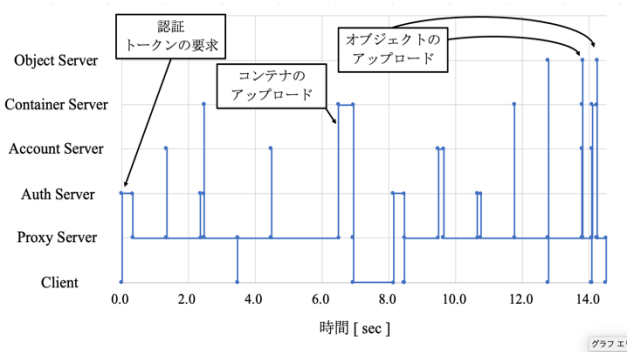


図 15　ssbench におけるパケット可視化例

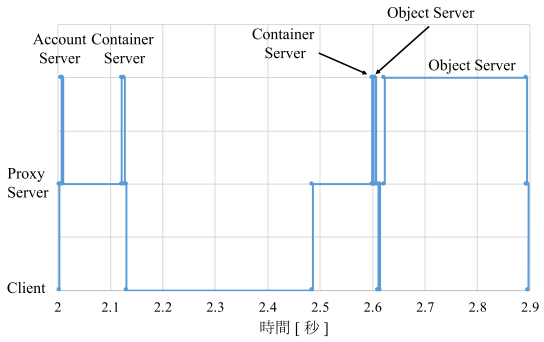


図 16　Swift API におけるパケット可視化例

図 16 より，Swift API を用いて 1B のオブジェクトをアップロードする際の最長の時間消費処理は，クライアント機において Swift プロセスが生成されてから最初のパケットがクライアント機から送出されるまでの処理(これには Python インタプリタの起動などが含まれている)であり，ストレージノードにおける処理ではないことがわかる．

図 15 より，並列度 1 の状況においてはストレージノードにおける処理は長い時間を要してはおらず，プロキシノードにおける処理が長い時間を要していることがわかる．

## 7. 考察

図 15，図 16 より，並列度が 1 の状況では本解析手法を用いてシステムの振舞を俯瞰でき，これは動作の把握を助けることができると考えらえる．ただし，高い並列度となり多数の処理が並行に動作している状況でも振舞の把握を適切に行えるかについても考察する必要があると考えらえる．

4 章の評価により，HDD と SSD のどちらの場合もある程度までは並列数の増加に伴いスループットが増加するが，ある並列数を超えるとスループットが低下し始め，過度に増加させると ssbench のリトライ数や失敗数が増加しスループットが大きく低下することが分かった．このリトライ

や失敗は ssbench クライアントにおいてソケット通信の確立の失敗によるものである．よって，クライアント実装の改善などにより回避や解決が可能であり，オブジェクトストレージシステム Swift の性能などの限界に起因するものではないといえる．

HDD 使用時のボトルネック処理(性能決定要因)がストレージノードにおける I/O 処理であった理由およびその改善方法に関する考察を行う．Swift は，オブジェクトをオペレーティングシステムのファイルシステム上にファイルとして保存する．よって，ファイルシステム(本稿の例では xfs)におけるファイル書き込み処理の負荷がかかり，ボトルネック処理になったと思われる．特に，オブジェクトサイズが小さい場合はファイルシステム上に細かいファイルが多数作成されることにあり，性能が劣化したと考えらえる．よって，細かいファイルの処理性能が高いファイルシステムを用いることや，オブジェクト群をデータ管理システムに格納するなどの手法により，これらの負荷の軽減や性能の向上が実現できると予想される．また，計算機外部にて観察可能なパケットの観察に加え，ボトルネック部の計算機内部システムソフトウェアの動作の観察[10][11][12]を行うことにより，性能劣化原因の特定と性能の向上が可能になると期待できる．

## 8. おわりに

本論文では, オブジェクトストレージシステムに着目し，その性能の評価と，性能や動作の解析方法について考察を行った．具体的には，OpenStack Swift を用いてオブジェクトストレージシステムを構築し, ssbench を用いて処理性能の評価を行った．そして，各計算機の負荷状況の調査結果を示し，パケット転送の記録によるオブジェクトストレージシステムの動作の観察方法につていの考察を行った．

今後は，高い並列度にて動作するオブジェクトストレージシステムの振舞の解析方法についての考察を行っていく予定である．

**謝辞** 本研究は JSPS 科研費 15H02696, 17K00109, 18K11277 の助成を受けたものである．

本研究は，JST，CREST JPMJCR1503 の支援を受けたものである．

## 参考文献

[1] Sridevi Bonthu, Y S S R Murthy, M. Srilakshmi “Building an Object Cloud Storage Service System using OpenStack Swift”, International Journal of Computer Applications (IJCA’14), 2014.

[2] Saneyasu Yamaguchi, Masato Oguchi, Masaru Kitsuregawa, "iSCSI analysis system and performance improvement of sequential access in a long-latency environment," Electronics and Communications in Japan (Part III: Fundamental Electronic Science), Volume 89, Issue 4, Pages 55-69, Wiley Subscription Services, Inc., A Wiley Company, April 2006. DOI: 10.1002/ecjc.20238

[3] “GitHub - swiftstack/ssbench: Benchmarking tool for Swift clusters”. https://github.com/swiftstack/ssbench, (参照 2018-11-20).

[4] S. Yamaguchi, M. Oguchi and M. Kitsuregawa, "Trace system of iSCSI storage access," The 2005 Symposium on Applications and the Internet, 2005, pp. 392-398. doi: 10.1109/SAINT.2005.65

[5] Saneyasu YamaguchiMasato OguchiMasaru Kitsuregawa, “Trace System of iSCSI Storage Access and Performance Improvement,” Database Systems for Advanced Applications (DASFAA), pp. 487-497, Springer Berlin Heidelberg, 2005. DOI: 10.1007/11408079_43

[6] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner, “OpenFlow: enabling innovation in campus networks,” *SIGCOMM Comput. Commun. Rev.* 38, 2 (March 2008), 69-74. DOI=http://dx.doi.org/10.1145/1355734.1355746

[7] Saneyasu Yamaguchi, Akihiro Nakao, Masato Oguchi, Atsuhiro Goto, and Shu Yamamoto. 2016. Monitoring Dynamic Modification of Routing Information in OpenFlow Networks. In *Proceedings of the 10th International Conference on Ubiquitous Information Management and Communication* (IMCOM '16). ACM, New York, NY, USA, , Article 27 , 7 pages. DOI: http://dx.doi.org/10.1145/2857546.2857574

[8] Q. Zheng, H. Chen, Y. Wang, J. Duan and Z. Huang, "COSBench: A Benchmark Tool for Cloud Object Storage Services," *2012 IEEE Fifth International Conference on Cloud Computing*, Honolulu, HI, 2012, pp. 998-999. doi: 10.1109/CLOUD.2012.52

[9] Omar SEFRAOUI, Mohammed AISSAOUI, Mohsine ELEULDJ; “OpenStack: Toward an Open-Source Solution for Cloud Computing”, International Journal of Computer Applications (0975 - 8887) Volume 55 - No. 03, October 2012

[10] Kyosuke Nagata, Saneyasu Yamaguchi, "An Android application launch analyzing system," *2012 8th International Conference on Computing Technology and Information Management (NCM and ICNIT)*, Seoul, 2012, pp. 76-81.

[11] Yuta Nakamura, Kyosuke Nagata, Shun Nomura, and Saneyasu Yamaguchi. 2014. I/O scheduling in Android devices with flash storage. In *Proceedings of the 8th International Conference on Ubiquitous Information Management and Communication* (ICUIMC '14). ACM, New York, NY, USA, , Article 83 , 7 pages. DOI: https://doi.org/10.1145/2557977.2558025

[12] H. Hirai, M. Oguchi and S. Yamaguchi, "A proposal on cooperative transmission control middleware on a smartphone in a WLAN environment," *2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Lyon, 2013, pp. 693-700. doi: 10.1109/WiMOB.2013.6673432