ディスクエリアネットワークを用いたオブジェクトストレージの高速なデータ復旧手法

小西 洋太郎† 小野 貴継† 三吉 貴史†

ディジタルデータの爆発的増加に伴いクラウド型オブジェクトストレージシステムの需要が拡大している．オブジェクトストレージシステムは種々の要因で性能が律速されるが，一般にストレージノードの計算資源に余剰が発生する場合が多い．したがって，システム容量を拡張する場合にはノード数の増加を抑制し 1 ノード当たりのディスク数を増加することで，容量あたりの消費電力を削減することが可能である．しかしながら，この構成においてノード障害が発生した場合，従来のデータ転送によるリカバリ処理では処理が長期化する問題がある．この問題を解決するため，本稿ではディスクエリアネットワークを用いたディスク接続切替による高速なデータ復旧手法を提案する．ディスクエリアネットワークは複数のサーバノードと複数のディスクドライブとの間を任意に接続または切断可能とする技術である．提案手法を OpenStack Swift に適用して評価した．評価により，従来 13 時間以上を要した復旧処理を，提案手法ではディスクエリアネットワークを用いてデータ転送を不要とすることで，45 秒間で完了可能であるという結果を得た．

Fast data recovery for object storage systems using Disk Area Network

Yotaro Konishi,† Takatsugu Ono† and Takashi Miyoshi†

Digital data are increasing explosively, and the demand for clound object storage is becoming higher. Although there are a lot of elements which might be the bottleneck of cloud object storage systems, storage nodes tend to have room of CPU resources. To expand the capacity of the storage system, it is more efficient to increase the number of the disks per storage node and to reduce the number of the storage nodes. However, the conventional recovery approach from server failure takes a very long time for the recovery. This paper proposes fast data recovery mechanism by Disk Area Network, which enables storage nodes to change a connection of disks freely. Our evaluation results using OpneStack Swift show that our approach is able to recover 32 HDDs of data in 45 seconds while conventional method takes over 13 hours.

1. はじめに

ディジタルデータの爆発的増加に伴い，オンラインでデータの格納先を提供するクラウド型オブジェクトストレージサービスの規模が拡大し続けている．クラウド型のオブジェクトストレージシステムのひとつとして，Amazon S3[1](以下 S3) が挙げられる．S3 は個人向けのデータ保管先として用いられるほか，Dropbox[4] 等のオンラインサービスのバックエンドストレージとしても利用されている[5]．S3 が格納するデータ数は急激な増加傾向にあり[2]，2012 年 6 月には 1 兆個に達したと報告されている[3]．ディジタルデータの増加速度が著しいこと，およびオンラインストレージに対する高い需要が現れている．

† 株式会社富士通研究所
Fujitsu Laboratories Ltd.

S3 のアーキテクチャは非公開だが，S3 に類似した機能を提供するオープンソースソフトウェアとして，OpenStack Swift[6](以下 Swift) がある．幾つかの企業で既に商用サービスとして用いられており，現在も OpenStack プロジェクト[7] の主要構成要素として開発が継続されている．

Swift 等のオブジェクトストレージは利用者にストレージ容量を提供するサービスである．提供可能な容量の不足が見込まれる場合，ストレージ容量を随時拡張する必要が生じる．その手段として少なくとも以下の 2 通りが考えられる．

(1) システムにストレージ用ノードを増設する
(2) ストレージ用ノードに搭載するディスク数を増加する

Swift 等の分散システムでは，ノード増設によるシステム規模拡張に対応した設計であることが多い．上記 (1) の手段はこの仕様に沿ったものであるが，ストレー

ジ用ノードの計算資源（CPU，メモリ等）が余剰となることが考えられる．Swift の性能を律速する要因として，ネットワークのバンド幅，ノードの CPU 使用率，およびディスクに対する I/O などが推定される．Swift の性能がストレージ用ノードの CPU 使用率とは無関係に決定されると仮定すると，ストレージ用ノード数の増加に伴い余剰な計算資源が増加することとなる．この余剰な資源をも駆動しなくてはならないため，Swift の容量当たりの消費電力が増加する．

(2) の手段ではストレージ用ノードを追加する必要がなく，システムのノード資源をより効率的に利用することができる．(1) の手段と比較してシステム容量増加に伴う電力消費の増加が抑制できる．低コストで大容量を提供するクラウド型ストレージシステムの目的を鑑みると，ノード当たりのディスク数を増加させる手段は有効であると考えられる．

しかしながら，少数ノードに多くのディスクを搭載した構成では，ノード障害が発生した際に大きな欠点を伴う．ノード障害が発生した場合，システム信頼性を維持するため低下したデータの冗長性を速やかに回復しなくてはならない．従来のノード間のデータ転送による回復手法では，特に少数ノード・多ディスク構成において，リカバリ処理の長期化という問題が発生する．

この問題を解決するため，本稿ではノードに障害が発生した際に故障していないディスクを他のノードに切り替えることでリカバリ時間を短縮する手法を提案する．何らかの原因でノードに障害が発生し動作の継続できなくなった場合，障害ノードに格納されていたディスクは正常である場合が多い[13]．ここに着目し，ノード障害が発生した際に障害ノードがアクセスしていたディスクを他の正常なノードからアクセス可能にすることでデータを復旧する．障害ノードから正常ノードへのディスク切替を行うための機構として，著者らが開発した Disk Area Network スイッチ (DAN スイッチ)[15),16)] を用いる．少数ノード構成の Swift における評価の結果，理想的な条件下でのデータ転送において 13 時間以上要する復旧処理を，45 秒に短縮可能であるという結果を得た．

本稿の構成は以下の通りである．第 2 章では Swift について説明し，Swift の性能を律速する要因について予備評価を行い，少数ノード構成による Swift の利点と課題について述べる．第 3 章ではディスク切替によるリカバリ手法を提案する．第 4 章では 2 種類の評価構成に対して提案手法を評価する．第 5 章では関連研究について述べ，最後に第 6 章でまとめる．

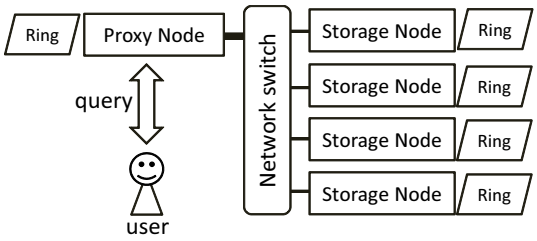

図 1 Swift 構成図
Fig. 1 Swift architecture

2. Swift に適したハードウェア構成の検討

2.1 OpenStack Swift の概要

本稿では提案手法を適用するシステムとして Swift を用いる．Swift は S3 に類似したサービスを提供するクラウド型のオブジェクトストレージであり，OpenStack コミュニティによりオープンソースで開発されている．Swift を用いた商用サービスが展開されている[8] ほか，オブジェクトストレージに関する評価実験にも用いられている[10),14)]．

Swift の構成を**図 1** に示す．Swift は主にプロキシノードと複数のストレージノードから構成される．プロキシノードはフロントエンドとしてユーザからのクエリを受信し，後段のストレージノード群に対して処理を伝達する．ストレージノードには実データが格納される．また Swift システムに対してユーザが送受信するデータの単位をオブジェクトと呼ぶ．Swift におけるデータの分散はオブジェクト単位で実行される．Swift はオブジェクト分散のためにハッシュテーブルを用いる．このハッシュテーブルを Swift では Ring と呼ぶ．Ring ファイルは各プロキシノードおよびストレージノードそれぞれに格納され，プロキシノードによるオブジェクト分散や，ストレージノード相互間のオブジェクト完全性検証および復元などに用いられる．

Swift はオブジェクトの複製（レプリカ）を複数作成して冗長化し，複数のストレージノードに分散させて保存することでユーザデータを保護する．ハードウェア障害等何らかの事故によりあるオブジェクトに対してアクセス不能となった場合，他のレプリカを用いてアクセスを継続することが可能となる．レプリカ数が多いほど，複数の障害に対してオブジェクトを保護することが可能である．冗長度の低下に伴いオブジェクト損失が発生しやすくなる．オブジェクトの恒久的な消失は，あるオブジェクトのレプリカ全てが失われた際に発生し，レプリカ数が小さいほどオブジェクト消失の発生確率は高まる[11]．このため，低下したレプリ

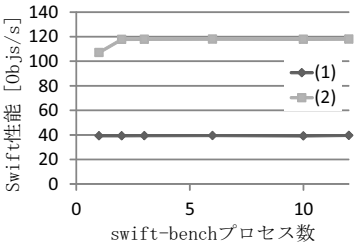

図 2 クエリ (1),(2) による Swift 性能
Fig. 2 Swift performance with query (1) and (2)

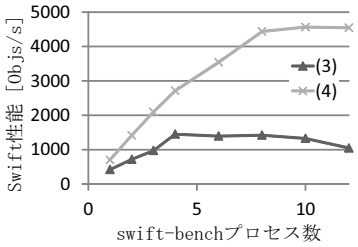

図 3 クエリ (3),(4) による Swift 性能
Fig. 3 Swift performance with query (3) and (4)

表 1 Swift 予備評価構成
Table 1 Evaluation environment of Swift performance

プロキシノード数	1
ストレージノード数	4
ディスク数	各 8
ネットワークバンド幅	1Gbps

カ数は早期に復旧しなくてはならない.

2.2 Swift 性能ボトルネック評価

Swift の性能律速の要因を調査するため**表 1** の構成にて評価した．プロキシノードを 1 台，ストレージノードを 4 台とし，各ノードをギガビットイーサネットによって接続した.

Swift の性能律速要因はクエリの種類によって異なることが予想される．そこでクエリを下記の 4 種類とした.

(1) ファイルサイズ大 (1MB)，アップロード
(2) ファイルサイズ大 (1MB)，ダウンロード
(3) ファイルサイズ小 (4KB)，アップロード
(4) ファイルサイズ小 (4KB)，ダウンロード

Swift パッケージに付属する swift-bench というツールを用いて性能を測定した．swift-bench 実行プロセス数を増加させることで，Swift のプロキシノードが受信する時間当たりのクエリ数が増加し，Swift は高負荷を処理する状態となる．swift-bench プロセス数を 1〜12 の範囲として，Swift に与える負荷量と性能との関係を調査した.

(1) および (2) のクエリについての測定結果を**図 2** に示す．また (3) および (4) のクエリについての測定結果を**図 3** に示す．縦軸は Swift の性能値であり，単位は 1 秒あたりのオブジェクト送受信数である．横軸は swift-bench プロセス数であり，数値が大きいほどシステムの負荷が大きいことを表す．図 2 および図 3 より，クエリ (1) および (2) については早期に性能が上限に達していることがわかる．クエリ (3) についてはプロセス数 4 以上で，クエリ (4) についてはプロセス数 10 以上で性能の上限に達していることがわかる.

表 2 は，本節の測定において Swift 性能が上限に達した際の，プロキシノードの CPU 使用率，ストレージノードの CPU 使用率，およびプロキシノードのネットワーク転送量を示す．なお，プロキシノードネットワーク転送量の列については，受信速度と送信速度のうち転送量が大きいもののみを示している．すなわち，クエリ (1) および (3) についてはプロキシノードの受信速度，クエリ (2) および (4) についてはプロキシノードの送信速度である．(1) および (2) のプロキシノードバンド幅，(4) のプロキシノード CPU 使用率は性能律速要因と推定される．表 2 より，クエリ (1) および (2) においてはプロキシノードネットワーク転送量が 1Gbps の理論値である 119MB/s（1KB = 1024Byte 換算）に達し，ネットワークにより性能が決定されていることがわかる．各ノードの CPU 利用率は低水準となっている．クエリ (4) においてはプロキシノードの CPU 使用率がほぼ 100%となり，これにより性能が決定されている．クエリ (3) では各資源利用率の変動が激しく，安定した状態とはならなかった．図中では最大値と最小値を示している．プロキシノードの CPU 使用率，ストレージノードの CPU 使用率およびプロキシノードのネットワーク状況が最大となっていない状態においても Swift 性能の飽和が見られており，ディスクに対する I/O が性能律速要因の 1 つとなっていることが推定される.

2.3 容量あたりの消費電力を重視した Swift の構成

第 2.2 節の評価結果から，Swift の規模を拡張する際の効率的なハードウェア構成について検討する．Swift サービスが提供可能な容量の不足が見込まれる場合，随時 Swift の規模を拡大し容量を拡張する必要が生じる．Swift の容量を拡張するには，

(1) ストレージノードを増設する
(2) ストレージノードに搭載する HDD を増やす

の 2 通りの構成が考えられる.

(1) は Swift システムに内蔵 HDD を搭載したスト

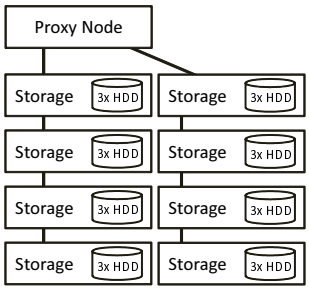

図 4 Swift 消費電力評価構成 (1)
Fig. 4 Swift cluster (1): 8 storage nodes and 24 HDDs

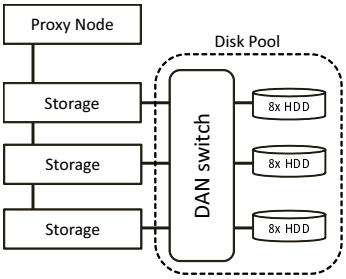

図 5 Swift 消費電力評価構成 (2)
Fig. 5 Swift cluster (2)

表 2 Swift 性能上限における資源使用状況
Table 2 CPU and Network utilization at Swift performance limit

クエリ種別	プロキシノード CPU 使用率	ストレージノード CPU 使用率	プロキシノード ネットワーク転送量
(1)	10.8%	14.8%	119MB/s
(2)	31.2%	11.3%	119MB/s
(3)	99-65%	82-55%	29-17MB/s
(4)	99.2%	21.3%	21.4MB/s

表 3 ストレージノード数と消費電力
Table 3 Relation between the number of storage nodes and power consumption

	構成 (1) (8 ノード)	構成 (2) (3 ノード)
プロキシノード	156W	155W
ストレージノード (1 台当たり)	152W	152W
ディスクプール	-	235W
総消費電力	1372W	846W

レージノードを増設することで，システム全体の容量を拡張する構成であり，Swift の本来の設計に則ったものである．しかしながら，第 2.2 節の予備評価の結果，Swift の性能ボトルネックとなる要因は複数存在することがわかる．クエリの種類や傾向によっては，ストレージノードの CPU 利用率が低水準であることが多く，必ずしもストレージノードの CPU 増加は Swift システムの性能向上に寄与しない．この場合，ストレージノード増設によって得られる効果は主としてシステム容量の増加となるが，容量増加に付随して増加する CPU やメモリ等の計算資源をも駆動しなくてはならないため，容量あたりの消費電力が増加する．

(2) はストレージノード数の増加を抑制し，何らかの方法によってストレージノード当たりの HDD 数を増加させることで，Swift の容量を拡張する構成である．ストレージノード数の増加を抑制することによって，容量あたりの消費電力を低減することができる．

(1) および (2) の構成にて Swift を実行し，消費電力の比較を行った．(1) による評価構成を**図 4** に，(2) による評価構成を**図 5** に示す．8 台のストレージノードそれぞれに 3 台の HDD を格納した構成を (1) の構成とした．また，3 台のストレージノードそれぞれに対し，DAN スイッチ（第 3 章で述べる）を用いたディスクプールから 8 台の HDD を接続した構成を (2) の構成とした．サーバ機材および HDD については同じ物を用い，システム総容量は HDD24 台分となり互いに等しい．プロキシノード数はそれぞれ 1 台とした．

上記 2 構成における電力測定結果を**表 3** に示す．プロキシおよびストレージノードあたりの消費電力は両構成ともほぼ同等の結果が得られた．ディスクプール部（DAN スイッチおよび全 24HDD を含む）の消費電力は 235W であった．システム全体の総消費電力ではノード数の差異による影響が大きく，図 4 および図 5 において，構成 (2) は消費電力を 526W 削減できていることがわかる．

2.4 多ディスク構成時に顕著化する問題点

第 2.3 節ではストレージノードに搭載する HDD 数を増加した構成における利点について述べた．しかしながら，この構成においてストレージノードの障害が発生した場合，障害からのデータ回復に長時間要するという問題が生じる．この問題は以下の 2 つの要因による．

- 復旧すべきデータ量の増大
- 転送バンド幅の制限

ストレージノードに障害が発生した際，システムからアクセス不能になるデータ量はストレージノードに搭載する HDD 数と比例する．多数の HDD を搭載したストレージノードでは障害発生時の容量的損失が相対的に大きくなる．障害に伴い再生成しなくてはならないレプリカが大量に発生するため，回復処理が長期化する．

また，レプリカ再生成におけるデータ転送の最大バンド幅はストレージノード数と比例する[9),12)]．多数のストレージノード全体に渡りレプリカが分散されているとき，レプリカ再生成処理は全てのストレージノード間において高バンド幅で実行される．しかしながら，ストレージノード数を抑制した場合，相対的にレプリカ再生成処理速度が遅滞しより多くの時間を要することとなる．

レプリカの再生成が完了せず，あるオブジェクトの冗長度が低下したままの状態で他のストレージノード障害が発生することも考えられる．オブジェクトのレプリカ数低下速度が再生成速度を上回るとき，オブジェクトの恒久的な損失に結びつく．レプリカ再生成処理時間とシステムの信頼性は相関する指標である．文献 11) では，レプリカ手法を用いた分散システムにおいて，ノード障害発生に起因するデータ消失確率は復旧に要する時間の 2 乗に比例する (レプリカ数が 3 の場合）と報告されている．ストレージノードが多くの HDD を搭載する構成では，従来のストレージノード間のデータ転送より高速なレプリカ再生成手法が必要となることが考えられる．

3. ディスク接続変更によるデータ復旧手法の提案

3.1 ノード障害時の HDD を再利用するアプローチ

第 2 章ではストレージノードの増設を抑制し搭載する HDD 数を増加した構成の利点と問題点を示した．多数のディスクを搭載したストレージノードに障害が発生した際，当該ノードのディスクに対してアクセスが不可能となる．しかしながら，ストレージノードに障害が発生した場合でもディスクにあるデータは失われておらず，そのまま利用することが可能な場合がある．このとき，ディスクを再利用し他のストレージノードからアクセスさせることができれば，低下したオブジェクトのレプリカ数を即座に回復させることができる．従来では人手を介さずにサーバに格納されたディスクを再利用することは困難であったが，動的にディスクとサーバの構成を変更可能な機構を導入することで本節で述べたアプローチが実現可能となる．

3.2 提案手法で用いるハードウェア

ディスク切替によるデータ復元手法の実現のために，サーバ間での動的ディスク切替を可能とする機構が必要である．本稿ではこの機構を備えたハードウェアとして DAN スイッチを用いる．DAN スイッチとは，著者らが提案している資源プール化アーキテクチャ[17)] の主要構成要素である．

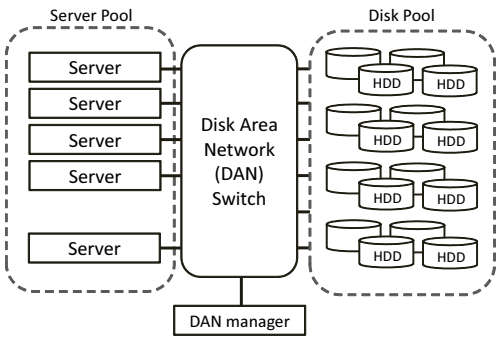

図 6 DAN スイッチによるサーバプールとディスクプールとの接続
Fig. 6 The Server Pool connected to the Disk Pool using DAN switch

資源プール化アーキテクチャの概要図を**図 6** に示す．サーバ機群を集積したサーバプールとディスクドライブ群を集積したディスクプールとの間でディスクエリアネットワーク (DAN) を構成する．2 つのプールは DAN スイッチによって連結されており，任意のサーバと任意のディスクとを排他的に対応付ける（接続する）ことが可能である．サーバに接続されたディスクは，サーバの OS からは内蔵されたディスクと同等に扱うことができる．DAN 経由でアクセスするディスクの性能はサーバ内蔵時と同等であり，DAN スイッチによる性能オーバヘッドは非常に小さい．DAN マネージャはサーバプールおよびディスクプールにおける資源を管理し，サーバとディスクの接続状況を保持する．また，DAN マネージャは外部からの要求に応じて DAN スイッチに対して命令を実行し，サーバノードとディスクとの接続および切断処理を行う．

3.3 ディスク接続変更によるデータ復旧手法

ノード障害時に故障ノードが搭載しているディスクを正常なノードに接続変更し，ノード障害により減少したオブジェクトのレプリカをデータ転送を行わずに復旧する．Swift における提案手法の構成例を**図 7** に示す．DAN スイッチを経由して各ストレージノードにディスクを接続しオブジェクトの格納先とする．

Swift 運用時にストレージノード故障が発生したとき，**図 8** に示した方法にてデータを復旧する．すなわち，

1. 故障が発生したストレージノードに接続されているディスクをすべて切断する．
2. 稼働を続けている他のストレージノードに対して 1. で切断したディスクを接続する．
3. サーバに対し，2. で接続したディスクをマウントさせる．

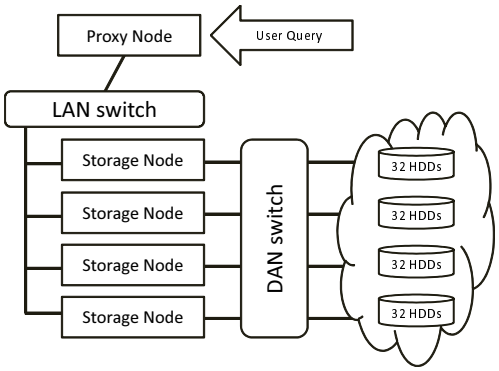

図 7 Swift における提案手法の構成例
Fig. 7 Exapmle of Swift using DAN

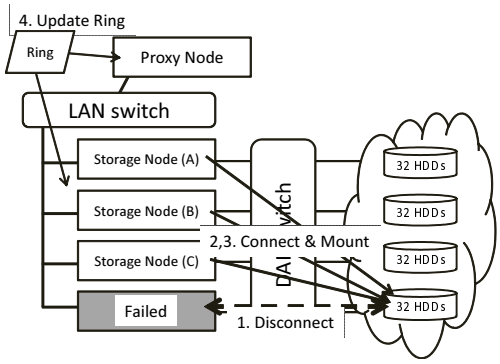

図 8 ノード故障発生時の処理
Fig. 8 Recovering method when a storage node failure occurred

4. Ring ファイルを更新し，各プロキシおよびストレージノードに再配布する．

本手法により，ノード障害発生以前に利用されていたディスクをそのまま再利用しサービスを継続可能である．リカバリのためのデータ転送が不要となり，システムのネットワーク帯域をより効率的に利用できる．さらに，従来データ転送速度に依存していたリカバリの所要時間が大きく低減し，大容量を搭載したストレージノードに対しても高速なリカバリが可能となる．

4. 提案手法の評価

4.1 提案手法の実装

第 3 章において提案した手法を図 7 の構成にて実装した．Swift 構成を 1 台のプロキシノードおよび 4 台のストレージノードとした．計 128 台の HDD を用い，DAN スイッチを介して各ストレージノードに対して 32 台ずつ HDD を接続した．各ノードはギガビットイーサネットにより相互に接続されている．

図 7 の構成においてストレージノードに障害が発生したとき，提案手法によるリカバリ処理を実行した．すなわち，障害が発生したノードに接続されている 32 台の HDD を切り離し，正常なストレージノードへそれぞれ接続した．この接続処理において，対象となる 32 台の HDD を 11 台，11 台，10 台の組に 3 分割し，図 8 のノード (A) に 11 台，ノード (B) に 11 台，ノード (C) に 10 台をそれぞれ接続することとした．HDD 接続処理後，各ストレージノードにおいて接続した HDD のマウント処理を実行し，Swift の Ring ファイルを更新した．

4.2 リカバリ処理時間の評価

第 4.1 節で示した実装において，リカバリ処理開始から完了までの時間を測定した．全体の処理時間は 44.5 秒であり，1 ディスク平均の処理時間は 1.4 秒であった．その内訳を**図 9** に示す．ディスクの切断および接続の処理が全体処理時間の 83%を占めており，提案手法のさらなる改善のためには DAN スイッチ操作部の高速化が必要であることがわかる．

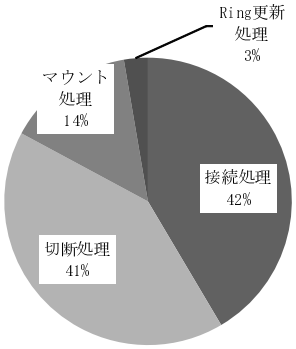

図 9 処理時間 44.5 秒の内訳
Fig. 9 Details of processing time (44.5 seconds)

提案手法における処理時間を，ストレージノード間データ転送によるリカバリ処理時間と比較評価する．比較にあたり，比較対象のハードウェア構成は第 4.1 節と同等とし，さらに以下の仮定を用いる．

(1) 各 HDD に記録されているデータ量を 512GB とする
(2) システムのユーザからのクエリが発生せず，各ストレージノード間にて理論上最大値である 1Gbps の転送速度で通信を行うものとする

(1) の仮定と HDD 台数が 32 であることより，リカバリを要するデータサイズは 16TB である．また (2) の仮定より，3 ストレージノード全体の転送速度は 3Gbps である．以上より転送時間を計算すると 13 時間となり，理想的な条件下でのデータ転送と比較した場合に

おいても，提案手法は非常に高速であることがわかる．

提案手法の処理時間中に外部から新たなオブジェクトのアップロードや既存のオブジェクトの更新処理があった場合，切換中の 32 ディスクはこれらの書き込みに応答することができず，他のレプリカとの不整合が生じる．Swift は結果整合性に基づく設計となっており，この不整合は切替完了後にストレージノード間で補完しあうこととなる．この補完処理はデータ転送により行われるが，ネットワークバンド幅が 1Gbps の場合 44.5 秒間で最大で送られてくるデータ量は 5.6GB である．これは上記 16TB と比較すると十分小さいと言える．ダウンロード処理に偏っている負荷状況の場合は，不整合が発生するデータサイズはより小さくなると推測できる．

ユーザから負荷の高いクエリが送信されている場合など，外部の影響によりリカバリ処理に使用できるバンド幅が制限されているとき，データ転送時間はさらに増大するため，提案手法の有効性がより高まる．さらに，提案手法による処理時間は HDD に記録されているデータ量に依存せず，また HDD 数増加に対する影響も小さい．したがって，HDD の記録密度が向上や HDD 数の増加に対しても提案手法はより有効となる．

4.3 ノード障害が回復した後の復帰処理の評価

Swift は耐障害性を高めるため，それぞれ異なるストレージノードに対してオブジェクトのレプリカを配置する．ストレージノード障害の際に提案手法によってディスクの切替を実行した場合，あるストレージノードが同一オブジェクトのレプリカを複数管理する状態となり得る．この状態において他のストレージノードに継続して障害が発生した場合，同時に複数のレプリカが一時的にアクセス不能となるため，クエリに対して応答不能となる問題が考えられる．

障害が発生したストレージノードの状態が回復した際にディスクの切替を再度実行することによって，レプリカの分散配置を初期状態に戻すことが可能である．この復帰処理を実装し，処理時間を測定した．

復帰処理は Swift システム稼働時の動的切替による影響を抑制するため 1 ディスク毎の処理とし，下記の順に従った．

1. Ring ファイルの更新および配布
2. 復帰元ノードにおけるディスクのアンマウント
3. 復帰元ノードからディスクの切断
4. 復帰先ノードへのディスクの接続
5. 復帰先ノードにおけるディスクのマウント

このとき，処理完了までの時間は 196 秒であり，1 ディスクあたり 6.1 秒であった．復帰処理実行中における，復帰先ノードの Swift によるネットワーク転送量を**図 10** に示す．縦軸は復帰先ノードネットワーク受信速度 [MB/s] であり，横軸は復旧処理開始からの処理時間 [s] である．Swift に対するクエリは第 2.2 節の (1) を使用した．ディスク数増加に伴い転送量が増加しており，プロキシノードのレプリカ分散処理が動的に切り替えたディスクに対して実行されていることがわかる．また，性能の安定化まで 300 秒程度を要し，復帰処理時間の 196 秒よりも大きい．これは Ring を更新してから Swift の動作に反映されるまでに時間差があるためと考えられる．この 300 秒の期間中にオブジェクトのアップロード要求があった場合，いくつかのオブジェクトのレプリカについて不整合が発生する可能性がある．Swift の負荷が小さい時間帯に復帰処理を実行することが望ましいと言える．

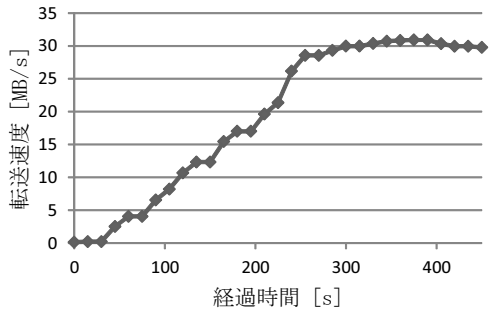

図 10 復帰処理中のデータ転送量
Fig. 10 Network bandwidth of the recovering node

5. 関連研究

分散ストレージにおいてノード障害が発生した際のデータの復旧に関する関連研究として文献 9)，11)，12) がある．ノード間のデータ転送によるデータ復旧速度は，クラスタ内のどのサーバにどのようにレプリカを配置するかというレプリカ配置のポリシに依存する．文献 9)，11)，12) では，異なるポリシそれぞれに対してデータ消失率をモデル化し，より信頼性の高いレプリカ配置ポリシについて提言している．これらの文献における議論は，ストレージノード間ネットワークによるデータ転送を前提としており，復旧速度は復旧すべきデータサイズおよび利用可能なネットワークバンド幅に依存する．本研究ではデータ転送によらないディスク単位でのデータ復旧を行うものであり，この点において異なる．また文献 9) ではデータ復旧処理を多数のノードから並列に実行することによって復旧を高速化する手法を提案している．第 2 章で述べたストレージノード数の増加を抑制した構成では高い並

列性は得られず，文献 9) による手法の効果は低いと考えられる．

ディスクエリアネットワークはサーバノードとディスクプールとの間の接続を任意のディスク単位で設定できる特性をもつ．文献 15)，16) ではこのディスクエリアネットワーク特性を利用した提案がなされている．文献 16) ではオブジェクトストレージにディスクエリアネットワークを適用した際，従来と比較して性能面でも貢献することが報告されている．

6. おわりに

本稿では，Swift の性能律速要因について評価した上で，ストレージノード数の増加を抑制した Swift 構成が容量当たりの消費電力の点で優れることを示した．この利点の一方で，上記構成ではストレージノード障害時にレプリカの復旧処理が長期化する問題が生じることを述べた．この問題を解決するため DAN スイッチを利用したディスク切替によるリカバリ手法を提案した．Swift に対して提案手法を適用して評価し，32 ディスク相当のレプリカ復旧において，理想的なデータ転送において 13 時間以上要する処理を 44.5 秒に短縮可能であることを示した．さらに，ストレージノードの障害が回復した後において，ディスクの状態を動的に元に戻すことが可能であることを示した．

今後，オブジェクトストレージのさらなる大容量化に伴い，ノードが搭載するディスク数は増加することが予想される．ノード当たりのディスク数の増加とともに提案手法の有効性はより高まると考えられる．

提案手法の処理時間のうち 83%が DAN スイッチ操作によるディスク切断および接続によるものであることを示した．今後，DAN スイッチ操作関連処理を高速化し，提案手法をより改善する予定である．提案手法はストレージノード障害の際に，ディスクに記録されているデータは正常であるという前提に基づく手法であり，ディスクの故障には対応できていない．ディスク故障時における高速なリカバリ手法についても今後検討する必要がある．

参 考 文 献

1) Amazon S3: http://aws.amazon.com/s3/
2) Amazon S3 - 905 Billion Objects and 650,000 Requests/Second: http://aws.typepad.com/aws/2012/04/amazon-s3-905-billion-objects-and-650000-requestssecond.html
3) Amazon S3 - The First Trillion Objects: http://aws.typepad.com/aws/2012/06/amazon-s3-the-first-trillion-objects.html
4) Dropbox: https://www.dropbox.com/
5) Dropbox - Where does Dropbox store everyone's data?: https://www.dropbox.com/help/7/en
6) Object Storage - OpenStack: http://openstack.org/projects/storage/
7) OpenStack: http://www.openstack.org/
8) OpenStack — Open Cloud Operating System supported by Rackspace: http://www.rackspace.com/cloud/openstack/
9) Q, Lian, et al: On the Impact of Replica Placement to the Reliability of Distributed Brick Storage Systems, in Proc. of the IEEE ICDCS, pp. 187-196, 2005.
10) Q, Zheng, et al.: COSBench: A Benchmark Tool for Cloud Object Storage Services, In Proc. of the IEEE Symp. on CLOUD, pp. 998-999, 2012.
11) V. Venkatesan, et al.: Reliability of Clustered vs. Declustered Replica Placement in Data Storage Systems, In Proc. of the IEEE Symp. on MASCOTS, pp. 307-317, 2011.
12) V. Venkatesan, et al: Reliability of Data Storage Systems under Network Rebuild Bandwidth Constraints, In Proc. of the IEEE Symp. on MASCOTS, pp. 189-197, 2012.
13) W. Jiang, et al.: Are Disks the Dominant Contributor for Storage Failures?, In Proc. of the USENIX Conf. on FAST, 2008.
14) Y. Hu, et al: NCCloud: Applying Network Coding for the Storage Repair in a Cloud-of-Clouds, In Proc. of the USENIX Conf. on FAST, 2012.
15) 小野貴継，他: ディスクエリアネットワークを用いたフラッシュストレージの性能改善手法の検討, 信学技報, vol. 112, no. 237, CPSY2012-44, pp. 79-84, 2012.
16) 小西洋太郎，他: 分散ストレージを対象としたディスク接続切替による高速なデータ復旧手法, SWoPP, pp. 121-126, 2012.
17) 三吉貴史，他: 次世代グリーンデータセンターを構成するシステム:Mangrove，FUJITSU Vol.62, No.5, pp. 545-551, 2011.