情報処理学会論文誌　コンピューティングシステム　Vol.6　No.4　38–48 (Oct. 2013)

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[1,a)]　小野 貴継[1]　三吉 貴史[1]

受付日 2013年4月9日, 採録日 2013年7月9日

概要：ディジタルデータの爆発的増加にともな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 ノードあたりのディスク数を増加させることで，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という問題がある．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え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．

キーワード：オブジェクトストレージ，フェイルオーバ，OpenStack Swift，ディスクエリアネットワーク

Fast Data Recovery for Object Storage Systems Using Disk Area Network

Yotaro Konishi[1,a)]　Takatsugu Ono[1]　Takashi Miyoshi[1]

Received: April 9, 2013, Accepted: July 9, 2013

Abstract: 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.

Keywords: object storage systems, failover, OpenStack Swift, disk area network

1. はじめに

ディジタルデータの爆発的増加にともない，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 1 つとして，Amazon S3 [1]（以下 S3）があげられる．S3 は個人向けのデータ保管先として用いられるほか，Dropbox [6] 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 [7]．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

[1] 株式会社富士通研究所
Fujitsu Laboratories Ltd., Kawasaki, Kanagawa 211–8588, Japan
[a)] konishi.yotaro@jp.fujitsu.com

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．

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 [10]（以下 Swift）がある．いくつかの企業ですで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 [11] 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させる．

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 等があげられる．仮に 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場合，ストレージ用ノード数の増加にともない余剰な計算資源が増加することとなる．この余剰な資源も駆動しなくてはならないため，Swift の容量あ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ともな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に鑑みると，ノードあ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ともなう．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度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 [18]．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え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）[20], [21] 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とおりである．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．3 章ではディスク切替えによるリカバリ手法を提案する．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．5 章では関連研究について述べ，最後に 6 章でまとめる．

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 [12] 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 [15], [19]．Swift の主な目的は高信頼かつ大容量のストレージ容量を安価に提供することである．

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等に用いられる．

Swift はオブジェクトの複製（レプリカ）を複数作成して

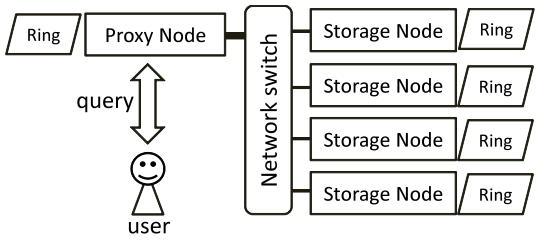

図 1　Swift 構成図
Fig. 1　Swift architecture.

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	1 Gbps

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．冗長度の低下にともな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すべ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 [16]．このため，低下したレプリカ数は早期に復旧しなくてはならない．なお，Swift はレプリカ間の同期に結果整合性を採用している．レプリカ間で不整合が発生した場合，一定時間ごとに最新のレプリカからデータの同期が行われ，整合性が保たれる．

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** の構成を用いる．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．

次に，性能評価に用いるベンチマークのクエリを決定する．Swift のような汎用的なストレージに対するベンチマークツールは現在研究開発が行われている [15] が，本稿執筆時においては利用できない．

Swift に入出力されるデータのサイズは Swift の利用形態によって異なる．ドキュメントの保存や Web ホスティング等の小ファイルサイズ主体のクエリや，大きい写真や動画の共有，アーカイブのバックアップ，VM イメージの供給等の大ファイルサイズ主体のクエリが考えられる．そこで本稿では，Swift のファイルサイズに応じた一般的な特性を調査するため，ベンチマークに用いるクエリを下記の 4 種類とした．

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

実運用上では複数ファイルサイズを混合したワークロード，およびアップロード/ダウンロード比率を混合したワークロード等が想定される．この場合でも，主となるワークロードが性能の中で支配的になることが推定できる[*1]．

[*1] 実アプリケーションに基づくワークロードにおける詳細な特性評価は本稿では議論の対象とせず，今後の課題とする．

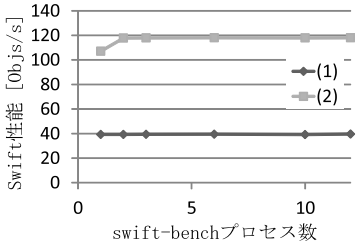

図 2　クエリ (1)，(2) による Swift 性能
Fig. 2　Swift performance with query (1) and (2).

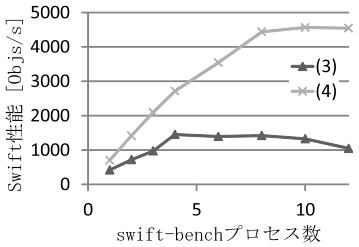

図 3　クエリ (3)，(4) による Swift 性能
Fig. 3　Swift performance with query (3) and (4).

表 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%	119 MB/s
(2)	31.2%	11.3%	119 MB/s
(3)	99.4–65.4%	82.0–55.1%	29.0–16.9 MB/s
(4)	99.2%	21.3%	21.4 MB/s

上記 4 種類のクエリに基づいて，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あ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飽和していることが分かる．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 以上で性能が飽和していることが分かる．

表 2 は，本節の測定において Swift 性能が飽和した際の，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 1 Gbps の理論値である 119 MB/s（1 KB = 1024 Byte 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分かる．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 システム構成の検討

2.2 節の評価から，クエリ (1)，(2) および (4) の 3 種類については，ストレージノードの CPU 使用率に余剰が生じていることが分かる．これら (1)，(2) および (4) のクエリのいずれかが主なワークロードである環境において Swift を用いた場合，Swift のストレージノード CPU 数の増加は Swift 性能向上に貢献しないことが推定される．

Swift 等のスケーラブルなストレージシステムでは，ストレージノード数を増加させることでユーザに提供する容量を拡張する設計である．しかしながら，クエリ (1)，(2) および (4) のいずれかが支配的となる環境においては，ストレージノードの CPU 利用率が小さく，他の要因で Swift 性能が決定される場合がある．したがって，ストレージノードの増加とともに余剰な CPU 資源も増加し，資源の利用効率が低い非効率なストレージシステムとなるおそれがある．

容量あたりの消費電力を低減させた高効率な Swift の構成のために，可能な限り多数の HDD を 1 つのストレージノードに集約し，余剰なストレージノードの増加を抑制すること考える．2.2 節の評価から，Swift の性能ボトルネックがストレージノードの CPU に依存しない環境のとき，ストレージノード数の増加抑制は Swift 性能に影響を与えない．HDD を集約することにより，ストレージとして提供可能な容量と性能を維持した状態でシステム全体の消費電力を低減させることが可能である．

2.4　多ディスク構成時に顕著化する問題点

2.3 節ではストレージノードに搭載する HDD 数を増加させ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を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．障害にともな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 [14], [17]．多数のストレージノード全体にわた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すべ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，相対的にレプリカを再生成する処理速度が遅滞しより多くの時間を要することとなる．

レプリカの再生が完了せず，あるオブジェクトの冗長度が低下した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．この場合，低下したオブジェクトの冗長度がさらに 1 段階低下する．障害発生による冗長度の低下速度がレプリカを再生成することによる冗長度の回復速度を上回るとき，オブジェクトの恒久的な損失に直結する．レプリカ再生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 [16] 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

2 章ではストレージノードの増設を抑制し搭載する HDD 数を増加させ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

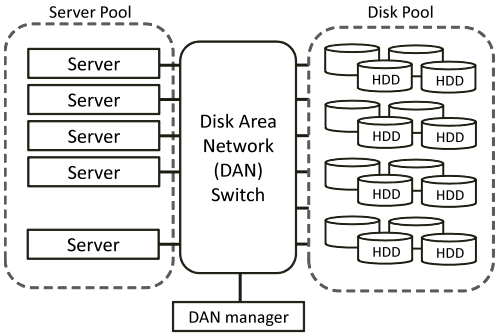

図 4　DAN スイッチによるサーバプールとディスクプールとの接続
Fig. 4　The server pool connected to the disk pool using DAN switch.

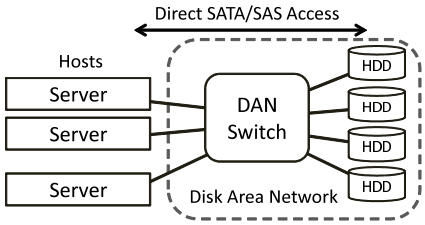

図 5　DAN におけるディスクアクセス
Fig. 5　Disk access on DAN.

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

3.2.1　Disk Area Network（DAN）の概要

ディスク切替え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え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 [22] の主要構成要素である．

資源プール化アーキテクチャの概要図を**図 4** 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．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．

試作した DAN のハードウェア構成を**図 5** に示す．DAN では，ホストとなるサーバプールのサーバと，ディスクプール内の HDD が，DAN スイッチを介して接続されている．ホストは内蔵ディスクに対して実行する場合と同様に，ディスクプール内の HDD とは SATA あるいは SAS のプロトコルによって通信可能である．

電源ユニットや冷却ファンは故障率が高いユニットであり，試作機においては電源およびファンについて各ユニットを二重化することで対応している．また，図 5 に示すように，DAN では DAN スイッチが単一障害点となっている．単一 HDD におけるインタコネクト障害が発生した場合，当該 HDD に関するアクセスが不能となるが，他の HDD について影響が及ぶことはない．これに対し，DAN スイッチにハードウェア障害が発生した場合は全 HDD に対するアクセスが不能となる．したがって，信頼性を向上させるためには DAN スイッチを二重化することが重要である[*2]．

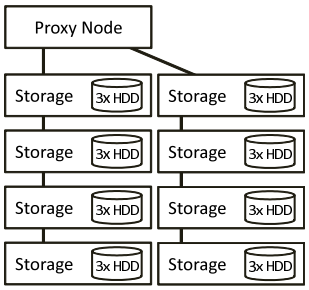

図 6　Swift 消費電力・性能評価構成（従来構成）
Fig. 6　Swift cluster (conventional).

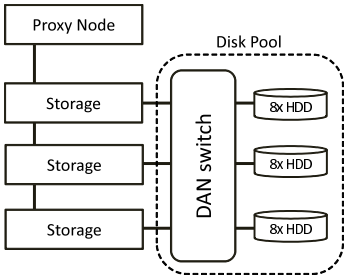

図 7　Swift 消費電力・性能評価構成（DAN）
Fig. 7　Swift cluster (DAN).

3.2.2　DAN で構成した Swift の消費電力

2 章で検討した HDD を集約した構成の有効性について定量的に評価する．DAN によって HDD を集約した構成について電力を測定し，従来の Swift の構成と比較した．

評価に用いた構成を**図 6**，**図 7** に示す．図 6 のように，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 Swift の従来構成とした．また，図 7 のように，3 台のストレージノードそれぞれに対し，DAN を用いてディスクプールから各 8 台の HDD を接続した構成を DAN による構成とした．サーバノードおよび HDD については各構成において同じものを用い，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

[*2] 著者らの試作では DAN スイッチの二重化は未実装であり，今後の課題とする．

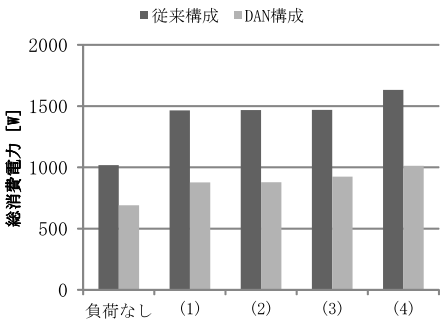

図 8　Swift 実行時の消費電力測定結果
Fig. 8　Power consumption while Swift running.

台とした．

2.2 節の評価で用いた 4 種類のクエリそれぞれについて，実行中のシステム消費電力を測定した．また，負荷をかけていない状態についてのシステム消費電力についても測定した．測定結果を**図 8** に示す．縦軸は Swift システムの総消費電力である．すなわち，従来構成においてはプロキシノードおよび 8 台のストレージノードにおける消費電力の合計値である．DAN 構成においてはプロキシノード，3 台のストレージノード，DAN スイッチおよびディスクプールにおける消費電力の合計値である．横軸は負荷について表し，負荷をかけていない状態および 2.2 節におけるクエリ (1) から (4) について示している．図 8 より，クエリ (1) から (4) すべてにおいて DAN 構成は従来構成と比較して 500 W 以上消費電力を削減できていることが分かる．クエリ (1)，(2) および (4) について，従来構成と DAN 構成とで同様の消費電力の推移を示している．クエリ (3) においては，DAN 構成ではノードあたりのディスク数が増加し，従来構成と比べてストレージノードの CPU 使用率が高まることによって，ストレージノードの消費電力の増加が発生している．DAN およびディスクプールの消費電力は 230 W（負荷なし）から 252 W（クエリ (4)）の範囲内であった．

3.2.3　DAN で構成した Swift の性能

図 6，図 7 の 2 つの構成について，Swift の性能比較を行った．2.2 節の評価と同様に，負荷を 4 種類のクエリとし，swift-bench によって性能を測定した．測定結果を**図 9** に示す．縦軸は従来構成における Swift 性能（スループット）を 1 とした場合の相対値を示す．

性能測定の結果，クエリ (1)，(2) および (4) については構成による性能差は確認されず，両構成について同等の性能となることが確かめられた．クエリ (3) においては，DAN による構成は 15%の性能低下が確認された．

このクエリ (3) における性能低下の原因としてファイルシステム上のファイル数の増加について検討した．クエリ (3) におけるベンチマークにおいて，Swift に送信するファイル数を 210 万と設定していた．これに対し，クエリ (3) と同様 4 KB ファイルの書き込み処理において，Swift に送信するファイル数を 150 万と設定したクエリ (3') で同様の実験を行った．測定結果を**図 10** に示す．縦軸は 1 秒あたりに書き込んだファイル数であり，Swift の性能を示す．図 10 より，クエリ (3') においては両構成における性能差が大きく低減されていることが分かる．クエリ (3) における DAN 構成の性能低下の原因として，HDD のマウントポイントにおいて作成されるファイル数が増加するとともに，ファイルシステムのメタデータの管理においてオーバヘッドが増大することによるものと考えられる．

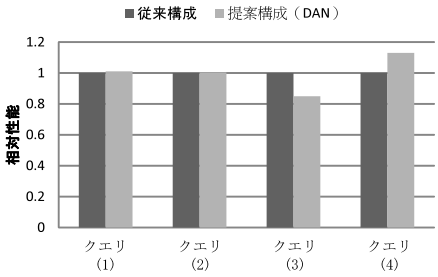

図 9　Swift 性能比較
Fig. 9　Swift pefromance.

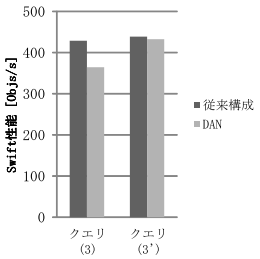

図 10　作成するファイル数差による Swift 性能
Fig. 10　Swift pefromance.

本 Swift 性能測定と同時に，DAN による構成において，ストレージノードの CPU 使用率を測定した．CPU 使用率測定の結果，各クエリ実行時におけるストレージノードの CPU 使用率の平均値は，クエリ (1) において 16.5%，クエリ (2) において 13.7%，クエリ (3) において 74.2%，クエリ (4) において 21.0%であり，2.2 節における予備評価と同様の傾向が確認された．図 7 の構成において，特にクエリ (1)，(2) および (4) について，ストレージノードの CPU 時間は多くがアイドル状態であることを確認した．

3.3　DAN と SAN の比較

従来の SAN（Storage Area Network）で用いられるストレージ装置 [9]（以下，SAN ストレージ）のハードウェア構成の一例を**図 11** に示す．ホストは SAN スイッチを介して SAN ストレージと通信を実行する．SAN ストレージには CPU やメモリ等のハードウェアを搭載した専用のコ

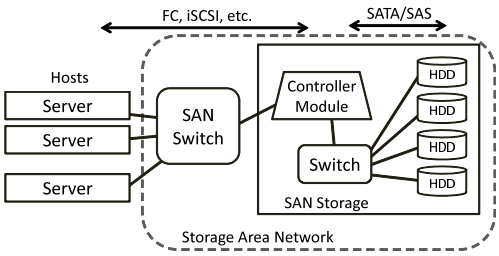

図 11　SAN におけるディスクアクセス
Fig. 11　Disk access on SAN.

ントローラモジュールが搭載されており，コントローラモジュールがホストとのインタフェースとなる．この際の転送プロトコルは Fibre Channel（FC），iSCSI 等が用いられる．コントローラモジュールと SAN ストレージ内部の HDD 群とは，内部のスイッチを介して接続され，通信は SATA や SAS 等のプロトコルによって実行される．

図 5 に示した DAN の構成と，図 11 に示した SAN の構成とにおける大きな差異は次の 2 点に集約できる．すなわち，1. SAN ストレージに搭載されているコントローラモジュールが DAN には存在しない．2. DAN は HDD ごとに独立した転送帯域を供給可能である．この構成の差異をふまえ，DAN と SAN を以下の 3 つの観点より定性的に比較する．

3.3.1　性能オーバヘッドの有無

DAN はホストが直接 HDD を扱うことが可能である一方，ホストが SAN ストレージに対して HDD アクセスを行う際は必ずコントローラモジュールを経由しなくてはならない．SAN ストレージに対して複数ホストから多数のアクセス要求が生じた場合，コントローラモジュールの CPU 負荷増大による性能オーバヘッドが発生しうる．

3.3.2　アプリケーション機能との適合性

SAN ストレージに搭載されたコントローラモジュールには，高信頼性を確保し継続動作を保証するための種々の機能が組み込まれている．またベンダ独自の機能も多数組み込まれている．したがって，一般的に SAN ストレージは高価な装置である．

本稿で用いた Swift は，安価なコモディティサーバをスケールアウトすることで高価な SAN ストレージの使用を回避するように設計されている．アプリケーションが直接 HDD を管理することで，従来 SAN ストレージが提供していた機能のいくつかについてアプリケーション側で実装している．たとえば，Swift はデータのレプリカを作成する仕様であるため，SAN ストレージが提供する RAID 機能を必要としないことがあげられる．Swift 等のアプリケーションに高価な SAN ストレージを適用する意義は小さい．

一方 DAN は高価かつ高機能であるコントローラモジュールを搭載しないため，安価に製造可能である．また，コントローラモジュールを介さずに，アプリケーションが直接 HDD を管理することを可能とする．したがって，アプリケーション側では，従来サーバ内蔵 HDD に対して行われていた制御と同等の操作を実行することが可能であり，Swift 等のアプリケーションに対する適合性が高いと考えられる．

3.3.3　信頼性に関する考察

SAN ストレージではホストと HDD との間にコントローラモジュールが介在する．コントローラモジュールは，モジュール自体のハードウェア障害のほか，ソフトウェア処理上の障害が発生する可能性がある[*3]．

ホスト側が発行した処理要求や転送データはまずコントローラモジュールが受信し，ソフトウェア処理を実行した後，SATA や SAS 等のプロトコルによってディスクに I/O 処理が実行される．したがって，ディスクの I/O 処理において，プロトコルスタックにコントローラモジュールのソフトウェア層が加わることとなる．文献 [18] ではプロトコルスタック起因の障害についても多数報告されている．

DAN ではホストはモジュールを介さず直接ディスクアクセスを実行可能である．このため，SAN で必要とされていたコントローラモジュールのハードウェアおよびソフトウェア障害について考慮する必要がない．この点において，DAN では障害が発生しうる要素の数が減少していると考えられる．

3.4　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 12** に示す．Swift の構成要素であるプロキシノードおよびストレージノードに加え，DAN スイッチを操作する DAN マネージャ，およびデータ復旧処理の主体となるリカバリマネージャとしてのサーバノードを互いに LAN で接続する．リカバリマネージャは LAN を経由して DAN マネージャに対して命令を送信し，DAN スイッチの操作を実行させることが可能である．またリカバリマネージャは各プロキシおよびストレージノードが格納している Ring ファイルのコピーを格納する．各ストレージノードには DAN スイッチを経由してディスクが接続されており，これらのディスクが Swift におけるオブジェクトの格納先となる．

Swift 運用時にストレージノード故障が発生したとき，リカバリマネージャは**図 13** に示した方法でデータを復旧する．すなわち，

1. リカバリマネージャは DAN マネージャに命令を送信し，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．

[*3] このため，SAN ストレージ製品ではコントローラモジュールの二重化等の対処が行われている．

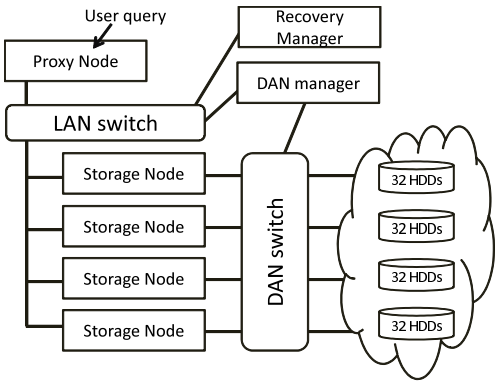

図 12　Swift における提案手法の構成例
Fig. 12　Exapmle of Swift using DAN.

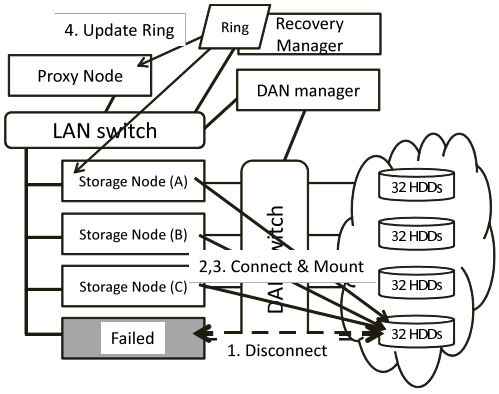

図 13　ノード故障発生時の処理
Fig. 13　Recovering method when a storage node failure occurred.

2. リカバリマネージャは DAN マネージャに命令を送信し，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. リカバリマネージャは各ストレージノードに対し，2. で接続したディスクをマウントする命令を発行する．
4. リカバリマネージャは自身が格納する 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され，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．

3.5　ノード復旧後のディスク再配置

3.4 節での接続変更処理を行った段階で，障害が発生したノードにに割り当てられているディスクはすべて切断される．しかしながら，ソフトウェアによる障害であればサーバノードの再起動等，またハードウェアによる永久故障であればサーバノードの交換等によって，障害が発生したサーバノードを復旧することが可能である．サーバノードの復旧後，再び Swift のストレージノードとして動作させるには，3.4 節で切り替えたディスクを復旧したサーバノードに接続する必要がある．以上で述べたディスクの接続状態を障害発生以前の状態に戻す処理を，ディスク再配置と定義する．

このディスク再配置処理を動作状態にある Swift に対して以下のように適用することができる．DAN マネージャおよびリカバリマネージャとして，3.4 節で説明したものと同じものを用いる．

1. リカバリマネージャは 3.4 節で接続変更したディスクを移動対象として事前に記録しておく．
2. リカバリマネージャは移動対象となるディスクを Ring ファイルより取り除き，各プロキシおよびストレージノードに再配布する．
3. リカバリマネージャは各ストレージノードに対して，移動対象となるディスクをアンマウントする命令を発行する．
4. リカバリマネージャは DAN マネージャに命令を送信し，移動対象となるディスクをストレージノードから切断する．
5. リカバリマネージャは DAN マネージャに命令を送信し，移動対象となるディスクを移動先のストレージノードに接続する．
6. リカバリマネージャは移動先のストレージノードに対して，5. で接続したマウントする命令を発行する．
7. リカバリマネージャは移動が完了したディスクを再び Ring に登録し，各プロキシおよびストレージノードに再配布する．

3.4 節のデータ復旧手法との手順の相違として，3. のアンマウント処理があげられる．障害が発生したストレージノードからディスクを切断することとは異なり，稼働中のストレージノードからディスクを切断するにあたり，事前に正常にアンマウントをしなくてはならない．これには一定の処理時間を要する．

以上の処理によって，各ストレージノードのデバイスはノード障害発生以前と同等の状態に復帰させることが可能である．3.4 節の切替え処理と本節の復帰処理とが高速に完了する場合，ハードウェアの永久故障を原因としない軽度のサーバノードの障害や，あるいはサーバノードのメンテナンス時においても，本手法を有効に適用することが可能である．

4.　DAN および提案手法の評価

4.1　提案手法の実装

3 章において提案したデータ復旧手法を DAN を用いて実装した．構成を図 12 に示す．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．HDD の集積度が高い状態における処理時間を評価するため，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 12 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 13 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

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 14** 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，切替え対象となるディスク数の増減によってこれら切断および接続の処理時間も増減する．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分かる．

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 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 512 GB 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で理論上最大値である 1 Gbps の転送速度で通信を行うものとする．
(3) ディスクの性能はネットワーク速度と比べて十分高速であると仮定し，データ転送速度がディスク性能とは無関係にネットワーク速度のみで決定されるとする．

(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 16 TB 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 3 Gbps 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分かる．

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，切替え中の 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，この不整合は切替え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 1 Gbps の場合 44.5 秒間で最大で送られてくるデータ量は 5.6 GB である．これは上記 16 TB と比較すると十分小さいといえる．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．

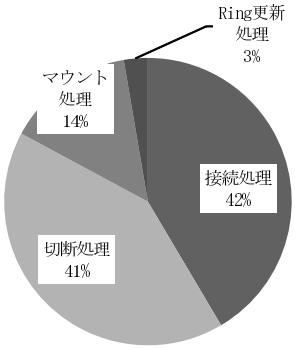

図 14　処理時間 44.5 秒の内訳
Fig. 14　Details of processing time (44.5 seconds).

ユーザから負荷の高いクエリが送信されている場合等，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

3.5 節で示したディスクの再配置処理を実装し，処理時間を測定した．実装においては，4.1 節における評価構成において，提案手法によるディスクの移動を行った後，再配置処理を行った．

測定の結果，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．3.5 節で述べたように，データ復旧処理と比べて，アンマウント処理の追加にともない処理時間が増加している．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 15** 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 2.2 節の (1) を使用した．図 15 よりディスク数増加にともな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分かる．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．

4.2 節の測定によるディスク切替え時間と，本節の測定によるディスク再配置時間とを合計すると，切替えから復

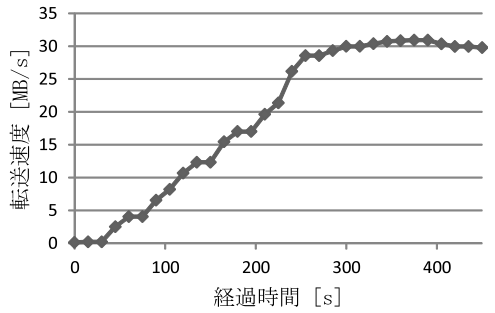

図 15　復帰処理中のデータ転送量
Fig. 15　Network bandwidth of the recovering node.

帰まで一連の動作が 6 分未満で完了するという結果が得られた．したがって，4.1 節の評価構成においては，6 分間以上のサーバ停止を要する短期的な障害やサーバノードのメンテナンスにおいても本提案は有効であるということが示された．

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 [14], [16], [17] 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 [14], [16], [17] 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 [14] 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．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 [14] による手法の効果は低いと考えられる．

本稿では Swift に対して DAN を適用し，ノード障害発生時における復旧処理に応用した．データのレプリカをディスクに保持する分散ストレージアプリケーションであれば原理的に本手法を適用することが可能であると考えられる．また，ディスク切替え中に発生する可能性があるレプリカ間の不整合について対応する方式が必要である．Swift のように，結果整合によって整合性が担保される場合はこれを利用することが可能である．Apache Cassandra [4] では，Swift と同様ハッシュの Ring を用いてデータを分散配置する．Ring の更新によって DAN のディスク切替えにアプリケーションの動作を対応させることが可能であると考えられるが，Cassandra ではディスク単位での細かなハッシュの割り振りができず，また新規に接続したディスクを Cassandra に組み込むことも困難であるため，DAN に対応した実装の追加を要する．Apache HBase [5] は HDFS [13] 上に構築されたデータベースであり主にメモリ上でデータを入出力する．このため HBase のノード障害時において DAN を適用することは困難である．しかしながら，メモリからフラッシュされたデータや，データベースに対するコミットログを後段の HDFS に格納する構成となっており，HDFS に対して DAN を適用することは可能であると考えられる．GlusterFS [8] では，DAN によるディスクの切替えを GlusterFS の Brick の移動と対応させることで提案手法を適用することが可能である．

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持つ．文献 [20], [21] 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 [21] 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うえで，ストレージノード数の増加を抑制した Swift 構成が容量あたりの消費電力の点で優れることを示した．DAN を用いて 24 台の HDD を集約した構成では，消費電力を 38%削減可能であることを示した．一方，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という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え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，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やディスクの高密度化が進むことによって，ノードが搭載するディスク容量は増加すると予想される．ノードあたりのディスク容量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, available from ⟨http://aws.amazon.com/s3/⟩.

[2] Amazon S3 - 905 Billion Objects and 650,000 Requests/Second, available from ⟨http://aws.typepad.com/aws/2012/04/amazon-s3-905-billion-objects-and-650000-requestssecond.html⟩.
[3] Amazon S3 – The First Trillion Objects, available from ⟨http://aws.typepad.com/aws/2012/06/amazon-s3-the-first-trillion-objects.html⟩.
[4] The Apache Cassandra, available from ⟨http://cassandra.apache.org/⟩.
[5] Apache HBase, available from ⟨http://hbase.apache.org/⟩.
[6] Dropbox, available from ⟨https://www.dropbox.com/⟩.
[7] Dropbox – Where does Dropbox store everyone's data?, available from ⟨https://www.dropbox.com/help/7/en⟩.
[8] Gluster Community Website, available from ⟨http://www.gluster.org/⟩.
[9] FUJITSU Storage ETERNUS, available from ⟨http://storage-system.fujitsu.com/jp/products/diskarray/dx-entry/download/⟩.
[10] Object Storage – OpenStack, available from ⟨http://openstack.org/projects/storage/⟩.
[11] OpenStack, available from ⟨http://www.openstack.org/⟩.
[12] OpenStack – Open Cloud Operating System supported by Rackspace, available from ⟨http://www.rackspace.com/cloud/openstack/⟩.
[13] Shvachko, K. et al.: The Hadoop Distributed File System, *Proc. IEEE Symp. on MSST* (2010).
[14] Lian, Q. et al.: On the Impact of Replica Placement to the Reliability of Distributed Brick Storage Systems, *Proc. IEEE ICDCS* (2005).
[15] Zheng, Q. et al.: COSBench: A Benchmark Tool for Cloud Object Storage Services, *Proc. IEEE Symp. CLOUD* (2012).
[16] Venkatesan, V. et al.: Reliability of Clustered vs. Declustered Replica Placement in Data Storage Systems, *Proc. IEEE Symp. on MASCOTS* (2011).
[17] Venkatesan, V. et al.: Reliability of Data Storage Systems under Network Rebuild Bandwidth Constraints, *Proc. IEEE Symp. on MASCOTS* (2012).
[18] Jiang, W. et al.: Are Disks the Dominant Contributor for Storage Failures?, *Proc. USENIX Conf. on FAST* (2008).
[19] Hu, Y. et al.: NCCloud: Applying Network Coding for the Storage Repair in a Cloud-of-Clouds, *Proc. USENIX Conf. on FAST* (2012).
[20] 小野貴継ほか：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).
[21] 小西洋太郎ほか：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).
[22] 三吉貴史ほか：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).

小西 洋太郎

昭和 61 年生．平成 23 年東京大学工学部電子工学科卒業．同年株式会社富士通研究所入社，現在に至る．データセンタ向けサーバの研究開発に従事．

小野 貴継　（正会員）

昭和 57 年生．平成 21 年九州大学大学院システム情報科学府情報理学専攻博士課程修了．博士（工学）．同年より日本学術振興会特別研究員（PD）．メモリアーキテクチャの評価に関する研究に従事．平成 22 年株式会社富士通研究所入社，現在に至る．データセンタ向けサーバの研究開発に従事．電子情報通信学会，IEEE 各会員．

三吉 貴史

昭和 46 年生．平成 8 年東京大学大学院理学系研究科情報科学専攻修士課程修了．同年富士通株式会社入社，サーバ向けインタコネクトの開発に従事．平成 19 年株式会社富士通研究所に所属，現在に至る．次世代データセンタを対象とするサーバアーキテクチャの研究開発に従事．