D-018

OpenStack のオブジェクトストレージへの コールドストレージ機能拡張に関する設計と実装

吉田 武俊† 中尾 鷹詔‡ 宮前 剛†

†株式会社 富士通研究所
〒211-8588 川崎市中原区上小田中 4-1-1
‡富士通クラウドテクノロジーズ 株式会社
〒169-0074 東京都新宿区北新宿 2-21-1
E-mail: †{lucky2u, miyamae.takeshi }@jp.fujitsu.com,
‡ nakao.takanori@nifty.co.jp

あらまし 大容量データの長期保存のニーズが高まり、アクセス遅延が大きい代わりに非常に安価な「コールドストレージ」が注目されている。我々は、OpenStack のオブジェクトストレージ（Swift）でコールドストレージを利用可能にした。既存の Swift は、遅延が大きなストレージを想定しておらず、コールドストレージへのアクセスで API のタイムアウトが発生しアプリケーションが正常に動かない。そこで、オートマトン方式による状態制御を実装し、Swift の API を拡張することで、低遅延と高遅延のストレージ間でオブジェクトを移動する API を追加した。オブジェクトの移動と更新の要求が並列に処理されたときにデータ消失しないように、ステータスと属性の一貫性を保つ排他制御を導入した。本稿では、これらの設計と実装を述べる。
キーワード コールドストレージ, オブジェクトストレージ, OpenStack, Swift

Design and Implementation of Cold Storage Service in OpenStack Object Storage

Taketoshi Yoshida†, Takanori Nakao‡, Takeshi Miyamae†

† FUJITSU LABORATORIES LTD.
4-1-1 Kamikodanaka, Nakahara-ku, Kawasaki-shi, Kanagawa 211-8588, Japan
‡ FUJITSU CLOUD TECHNOLOGIES LIMITED
2-21-1 Kitashinjuku, Shinjuku-ku, Tokyo 169-0074, Japan

Abstract These days "cold storage", which provides huge capacity with very low cost but high access latency, has attracted much attention because the needs of long term storage has been increasing. We extended OpenStack object store (Swift) to realize "cold storage" in an OpenStack environment. Since existing Swift APIs do not assume high access latency, a timeout error will be occurred if it accesses to "cold storage". As a result, applications can not be executed properly. To solve this problem, we extended Swift APIs. We added APIs to move an object between low latency and high latency storages. We designed state transition mechanism for objects, but we found that data lost can be occurred in case of move and update operations are executed in parallel. Preventing such kind of situations, we implemented an exclusive control mechanism to maintain consistency of state and attributes of an object. In this paper, our design and implementation are described.
Key words Cold Storage, Object Storage, OpenStack, Swift

１．まえがき

近年、取り扱われるデータは動画や画像などのデータ形式や、サイズ、数も多様化しており、さらにその容量も大容量化してきている。このようなデータの管理は、ファイルストレージよりも、オブジェクトストレージが向いており広く利用されている。実際、オブジェクトストレージは多くのクラウド事業社で提供されており、Amazon Web Services(以下 AWS)の S3 や、Google の Google Cloud Storage がある。また、法律や社会的要請を満たすために大容量のデータを数年～10 年を超える長期間に渡って保存し続けるニーズが高まっている[1]。そのため、AWS の Glacier[2] や Google Cloud の

Coldline[3] などのストレージサービスのように、アクセス性能を抑える代わりに容量あたりの保管コストを非常に低くした「コールドストレージ」に注目が集まっている[4]。以下、本稿ではテープや光ディスクなどを用いたデータを読み書きするときのレイテンシが大きいストレージを「コールドストレージ」、ハードディスクや SSD などを用いたレイテンシが小さいストレージを「ホットストレージ」と呼ぶ。我々は、オープンソースのクラウド基盤ソフトとして広く利用され、AWS の S3 と互換の API[5]を提供している OpenStack[6]のオブジェクトストレージ Swift[7]でも、コールドストレージを利用可能にすることとした。現在、OpenStack はコールドストレージを取り扱う仕組みがない。コールドストレージを単独の機能として提供することも考えられたが、ユーザが既に利用している Swift でコールドストレージを提供し、データを長期間安価に保存し続けられるようにすることを目標とした。様々なベンダのコールドストレージ装置をつなげられるアーキテクチャを設計した。

2．背景

Swift の標準 API は REST (Representational State Transfer) Web サービス[9]のメソッドとリソースパスの組み合わせとして実装されている。表１は、オブジェクトに対する Swift の標準 API を示している。リソースパスの account はテナント識別子、container はコンテナの識別子、object はオブジェクトの識別子である。

表１ Swift のオブジェクト用の標準 API

API		概要
REST ﾒｿｯﾄﾞ	リソースパス	
GET	/v1	オブジェクトとメタデータの取得
PUT	/{account} /{container}	オブジェクトの作成もしくは置換え
DELETE	/{object}	オブジェクトの削除
POST		オブジェクトのメタデータ作成もしくは更新
HEAD		オブジェクトのメタデータの取得

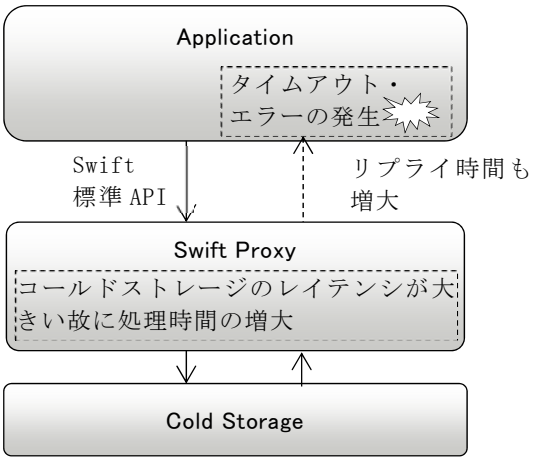

図１ 大きなレイテンシによるタイムアウトの発生

いずれも同期型の REST API として実装されており、レイテンシの大きなストレージ装置が使われることを想定した設計になっていない。そのため、コールドストレージに対し Swift 標準の API でオブジェクトの操作を行うとタイムアウトエラーが発生し、アプリケーションが正常に動かないという問題がある（図１）。

そこで、Swift の API を拡張し、Swift の配下に接続したホットストレージとコールドストレージの間でオブジェクトを移動するための API を追加することにした(図２)。アプリケーションがコールドストレージを利用するには、追加した拡張 API を明示的に呼び出す必要がある。また、この拡張 API に基づき両ストレージ間のオブジェクト管理を行う Swift High Latency Media(以下、Swift/HLM)ミドルウェアを開発した。図２の太枠及び太文字が今回開発部分を示す。

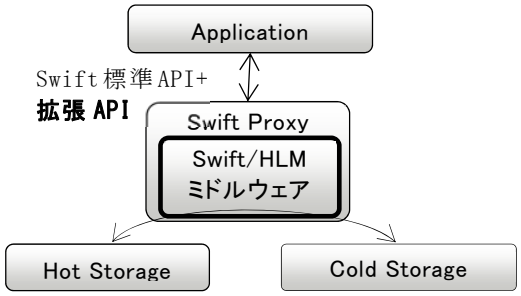

図２ コールドストレージを提供するシステム構成

3．提案手法と実装

本項では、API のタイムアウトを回避するための手法や、既存 Swift とコールドストレージ間でオブジェクトのステータスと属性情報の継承を確実にする手法について示す。

3．1 提案手法

(1) API の拡張

コールドストレージに直接 API を発行するのではなく、ホットストレージとコールドストレージを統合し、API は１つの口で受ける構成とした。ホットストレージからコールドストレージへオブジェクトを移動するための“MIGRATE”API を設け、その逆は“RECALL”API とした(図３)。これらの API は、タイムアウトが発生しない非同期型の API とした。

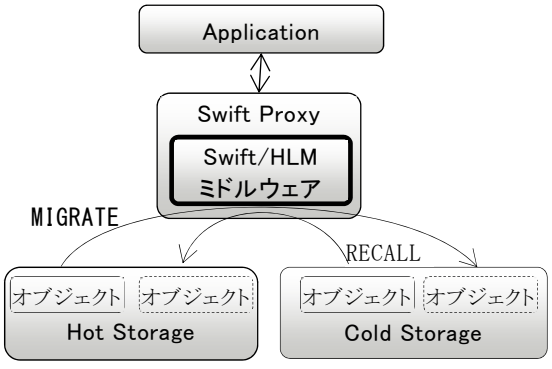

図３ オブジェクトの移動

また、非同期 API により処理されているオブジェクトのステータスを取得できる同期型の API を設定することにより、非同期処理が正常に完了したか検知できるようにした。これら Swift API を拡張した API を表２に示す。

表２ 拡張した API

API	概要	同期/非同期
MIGRATE	ホットストレージからコールドストレージへオブジェクトの実体を移動する。移動後、ホットストレージには、移動したことを示すサイズ０の特別なオブジェクトを stub として置き、そのメタ情報でオブジェクトのステータスを管理する。	非同期
RECALL	コールドストレージからホットストレージへオブジェクトの実体を移動する。	非同期
STATUS	オブジェクトのステータスを取得する。	同期

MIGRATE API の仕様を表３、RECALL API の仕様を表４、そして STATUS API の仕様を表５に示す。

表３ MIGRATE API

API 名	MIGTATE
REST API メソッド	POST
同期／非同期	非同期
パス	/v1/<ACCOUNT-ID>/<CONTAINER>/<OBJECT>
ヘッダ	X-Storage-Token: トークン ID
	X-Migrate-From (Optional):ProxyID+コンテナ ID
	X-Migrate-To (Optional):ProxyID+コンテナ ID
ボディ	なし

表４ RECALL API

API 名	RECALL
REST API メソッド	POST
同期／非同期	非同期
パス	/v1/<ACCOUNT-ID>/<CONTAINER>/<OBJECT>
ヘッダ	X-Storage-Token: トークン ID
	X-Migrate-From (Optional):ProxyID+コンテナ ID
	X-Migrate-To (Optional):ProxyID+コンテナ ID
ボディ	なし

表５ STATUS API

API 名	STATUS
REST API メソッド	GET
同期／非同期	同期
パス	/v1/<ACCOUNT-ID>/<CONTAINER>/<OBJECT>
ヘッダ	X-Storage-Token: トークン ID
ボディ	なし

MIGRATE API 処理は、ホットストレージからアプリケーションが指定したオブジェクトを読出し、コールドストレージへ同オブジェクトを書き込む。その後、ホットストレージへサイズが０のオブジェクトを stub として書き込む。その後、stub のメタ情報として、キーを“is-stab”とするメタデータを登録する（図４）。この API は非同期処理であり、Swift/HLM は同 API 受信後、アプリケーションへ受付応答を返すとともに MIGRATE 処理を行う。

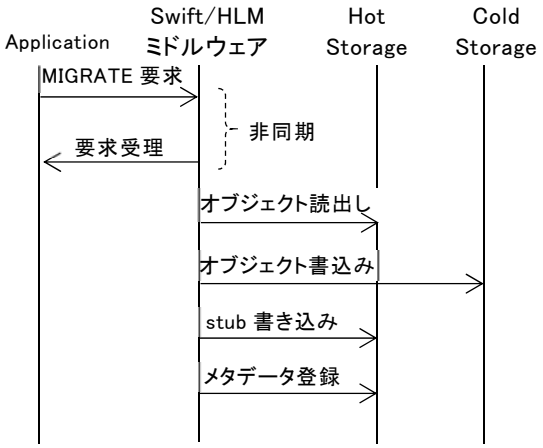

図４ MIGRATE 処理シーケンス

RECALL API 処理は、MIGRATE API 処理の逆で、コールドストレージからアプリケーションが指定したオブジェクトを読出し、ホットストレージに書き込む。この API も非同期処理であり、Swift/HLM は同 API 受信後、アプリケーションへ受付応答を返すとともに RECALL 処理を行う。

STATUS API 処理は、表６で示される現在のオブジェクトのステータスを取得し REST API のリプライメッセージのボディに記載しアプリケーションへ返信する。

表６ オブジェクトの状態

状態	説明
not exit	オブジェクトが無い状態
hot	オブジェクトの実体がホットストレージにある状態
cold	オブジェクトの実体がコールドストレージにある状態
migrating	オブジェクトが MIGRATE 処理中の状態
recalling	オブジェクトが RECALL 処理中の状態

表６で示した状態は、Swift 標準 API と拡張 API をトリガーとして遷移する（図５）。

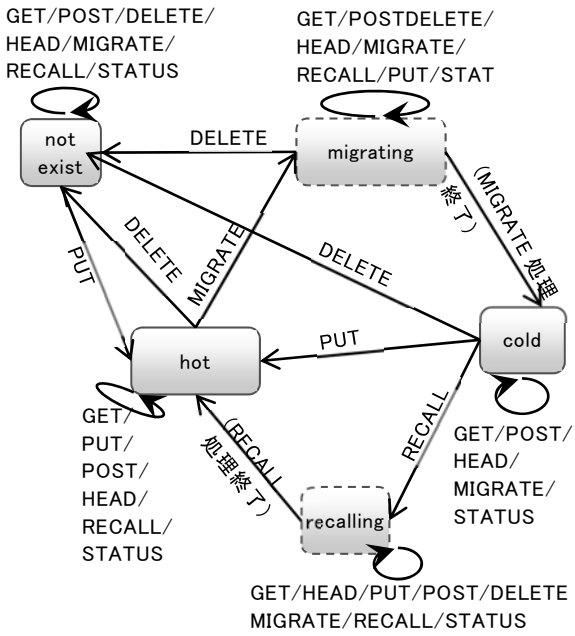

図５ Swift/HLM が管理するオブジェクト状態遷移

(2) 排他制御の導入

MIGRATE と RECALL は内部処理において、オブジェクトの更新を複数のステップに分けて実施している。その複数のステップの途中で標準 API の PUT が処理された場合に、この PUT が処理されない問題が生じた。そこで、オブジェクトを書き換える直前にステータスを確認して、期待するステータスでなければ処理を中止するようにした。しかし、ステータスを確認してからオブジェクトを書き換えるまでの間に、他の処理によるオブジェクトの書き換えが行われる場合に対応できずデータを失うことがある。

一例として、MIGRATE 要求が発行された直後に PUT 要求が発行されたケースで説明する。

各要求が単独に処理される場合のシーケンスを図６と図７に、要求が並列に処理されたためにデータを失う状況を図８に示した。図８では、MIGRATE 要求のシーケンスによる stub の書き込みの直前に、PUT 要求のシーケンスによるオブジェクトの書き込みが発生することで、PUT したデータを stub に置換してしまうことでオブジェクトが消失する。

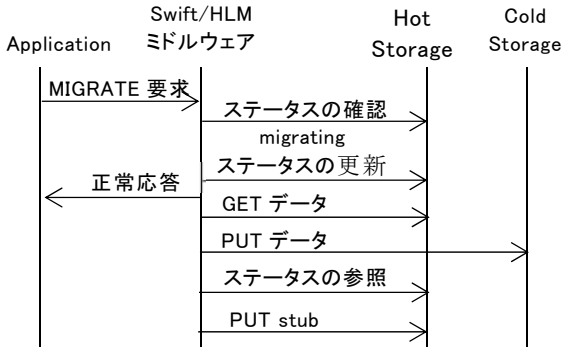

図６ MIGRATE 要求のシーケンス

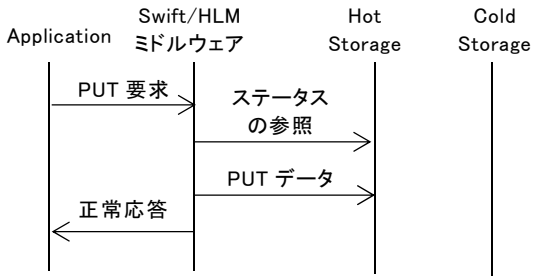

図７ PUT 要求のシーケンス

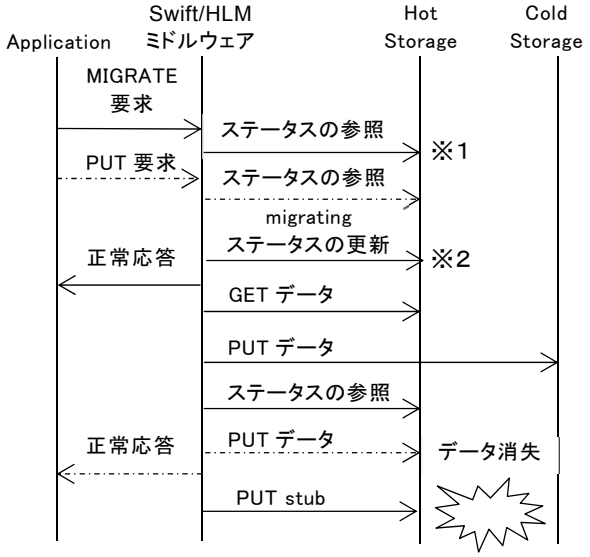

図８ MIGRATE 要求処理中の PUT 要求

そこで、排他制御を行なって、ステータスの参照（図８の※１）から migrating ステータスの更新（図８の※２）までをアトミックに実行できるようにした。

３．２ 実　　装

本システムで利用したソフトウェアを表７に示す。

表７ 利用ソフトウェア

OpenStack	Newton
Swift	2.7.1
python	Python 2.7.12
OS	Linux 4.4.0-71-generic Ubuntu 16.04.1

(1) メタ情報による状態管理

オブジェクトの各ステータスは、キー、バリューを元にした swift のメタデータ管理機能[10]を利用し、is-stub というメタデータを付けることで管理している。表８にステータスとそれを判断する is-stub メタデータおよびオブジェクトの組合せ関係を示した。ステータスが hot の時は、is-stub を付けていない。ステータスが cold の時は、is-stub を付けるが値はセットしていない。ステータスが migrating と recalling の時は、is-stub を付けて値にステータスを記述している。

表８　ステータスの管理テーブル

ステータス	メタデータ	オブジェクト
not exist	キーなし	なし
hot	キーなし	あり
cold	バリュー:空	あり(stub)
migrating	バリュー:migrating	あり
recalling	バリュー:recalling	あり(stub)

(2)機能構成

Swiftのアーキテクチャに従いWSGI を利用してSwift Proxy にレイヤーを挿入する形で実装した。また、どのベンダのコールドストレージを利用した場合も同じ実装になる部分を controller としてまとめて、デバイス固有の制御部をプラグイン化している。これにより様々なベンダの装置を最小限の実装で接続できる(図９)。

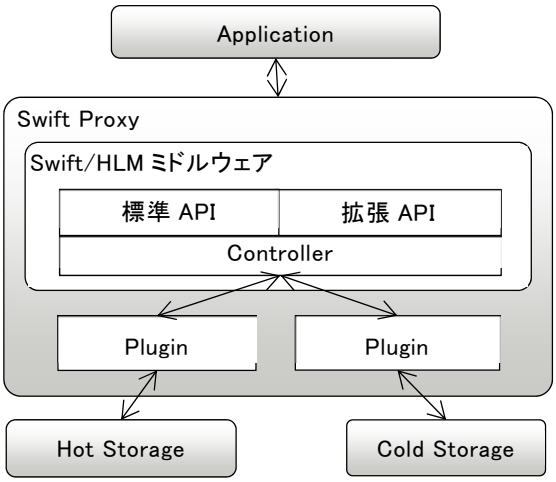

図９　アーキテクチャ

(3)複数 Swift Proxy 対応

Swift では、複数の Swift Proxy を用いた並列アクセスが可能である。複数の Swift Proxy を用いる場合、前節で述べた migrating ステータス更新処理は、Swift Proxy 間にまたがって排他制御を行う必要がある。そこで､分散メモリキャッシュサーバである memcached と OSS の分散排他機構である sherlock[11]を用いて、Swift Proxy 間にまたがった分散排他を実現した(図１０)。

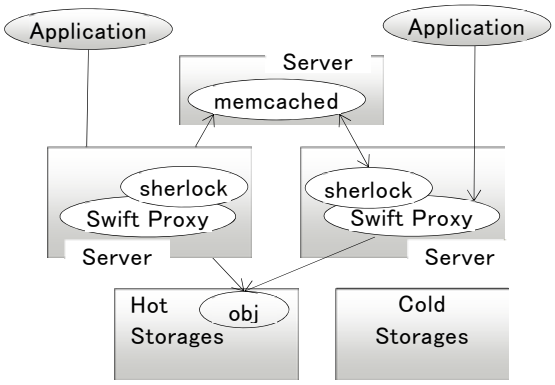

図１０ 排他制御を実現するシステム構成

４．OpenStack のコミュニティ動向

我々は、今回設計・実装した拡張 API を OpenStack の upstream に反映することを目指し、OpenStack Swift/HLM のプロジェクトに参加して、コミュニティの中で検討を進めている。プロジェクトには、我々の他に IBM,BDT,NTT,RedHat などからの参加メンバーがコミュニティで先行して検討されていたテープ装置のサポートに加えて、我々が提案した光学メディア装置のサポートも実現する方向で議論を進めている。

５．まとめ

OpenStack の Swift で、コールドストレージを利用できる汎用的なアーキテクチャの設計と実装を行い、光学メディア装置をつないで動作を確認した。

この結果、Swift 標準の API はそのままに、その API を拡張することにより、Swift の配下に接続したレイテンシの小さなストレージ装置とレイテンシが大きなストレージ装置の間でオブジェクトの移動を可能にし、安価に大量のデータを長期保存できるようにした。また、オブジェクトのステータスの更新と属性情報の継承を確実にするために、排他制御を導入することで、データ消失が発生しない実装をした。

文　献

[1] Bill Kleyman:"Understanding Cold Storage:Best Practices around Google Nearline & Amazon Glacier“, CloudBerry Lab ,Aug, 2015
[2] Amazon Glacier: “https://aws.amazon.com/jp/glacier/”
[3] Cloud Storage Coldline:” https://cloud.google.com/storage/archival/”
[4] 藤巻秀明, 久留米修, 入江将勝:”コールドストレージを活用したビックデータ解析基盤”, FUJITSU, 66(4), pp.49-54, July, 2015
[5] Object Storage API:” https://docs.OpenStack.org/newton/config-reference/object-storage/configure-s3.html”
[6] OpenStack:"https://www.OpenStack.org/"
[7] OpenStack swift:”https://docs.OpenStack.org/developer/swift/”
[8] Object Storage API:”https://developer.OpenStack.org/api-ref/object-storage/index.html”
[9] Roy Thomas Fielding:” Architectural Styles and the Design of Network-based Software Architectures”, UNIVERSITY OF CALIFORNIA, ch.5, 2000
[10] Middleware and Metadata:”https://docs.OpenStack.org/developer/swift/development_middleware.html”
[11] sherlock: Easy distributed locks for Python with a choice of backends:”https://github.com/vaidik/sherlock
[12] Swift/HLM: “https://github.com/ibm-research/SwiftHLM”