人工知能学会
第13回知識流通ネットワーク研究会
SIG-KSN-014-04

分散オブジェクトストレージのバックアップ方法に関する検討

○桑田喜隆(*1) , 梶波 崇(*1), 山田 大輔(*1), 三浦 広志(*1)
(*1) NTTデータ

A method to create a set of data for backup from Distributed Object-based Storage Systems

Yoshitaka Kuwata(*1) , Takasi Kajinami(*1),
Daisuke Yamada(*1), and Hiroshi Miura(*1)
(*1) NTT DATA Corporation, Japan

概要

近年，クラウドシステム上でペタバイトクラスのデータを蓄積するための技術として，分散オブジェクトストレージシステム（以下DOBSS）が注目をあびている．DOBSSは，データを複数のノードに冗長格納することで高い信頼性と可用性を実現することとともに，運用中にノードを追加することで事実上無制限の容量を持つストレージを構築することが可能である．

DOSS上でデータは冗長化して保持しているため，ハードウェアの同時多重故障等以外でデータが失われることはない．他方，ソフトウェアの故障への対応や人的操作ミスなど，より高い信頼性を求める応用例では，データ一式を別ストレージ上にバックアップしたいという要求がある．本稿では，DOSSに冗長保存されているデータを，別ストレージに効率的にバックアップをする方法に関して検討した．

Abstract

Distributed Object-based Storage System (DOBSS) is used as base technologies of Peta-byte class storage. In order to achieve high-reliability and high-availability, multiple copies of data are stored in independent nodes. It is possible to increase capacity of DOBSS by adding nodes, and that makes DOBSS unlimited-capacity storage.

As redundant copies are stored in DOBSS, no data-loss is occurred but multiple failure of hardware. To prevent software fault and human error, it is required to make a set of backup copy onto external storage systems. In this paper, we study efficient methods to make backup copies from DOBSS.

1. はじめに

近年，クラウドコンピューティングの社会的ニーズ増大と分散処理技術の発展を背景に，増大し続けるデータの格納先として，分散オブジェクトストレージシステム(Distributed Object-based Storage System，以下 DOBSS)が注目を集めている．従来の RAID を代表とするディスクシステムレベルでの冗長化の方式に比べ，DOBSS では複数のサーバにデータを分散し，冗長化することで信頼性を向上させることが可能である．特にペタバイトクラスの大規模ストレージシステムを構築する方法として，有望であると考えられている．

DOBSS は以下の特徴を持つ．

- ファイルシステムとは異なり「オブジェクト」という単位でバルクの入出力のみを対象とすることで，処理モデルを単純化してスケールする仕組みとしている．
- ノードを増設することで，格納するデータ容量および処理性能を適宜増加させることが可能である．データを格納するノードを増設する際には，既存のノードからデータを再配置する「リバランス処理」を実行することで，格納

[1] Yoshitaka Kuwata
NTT データ　基盤システム事業本部
東京都江東区豊洲 3-3-9 豊洲センタービルアネックス
kuwatay@nttdata.co.jp

されるデータ量のノード間のバランスを保つ.

- データを複数のノードに冗長化し分散格納することで，ディスクレベルの故障に加えてノードレベルでの故障にも対応することが可能である.
- 地理的に離れた場所に置かれた拠点のノードにデータを分散配置することで，拠点レベルでの障害に対してもデータの可用性を保つことが可能である.

図1に代表的な DOBSS である OpenStack Object Storage(Swift)[1), 2), 3)]の構成例を示す.

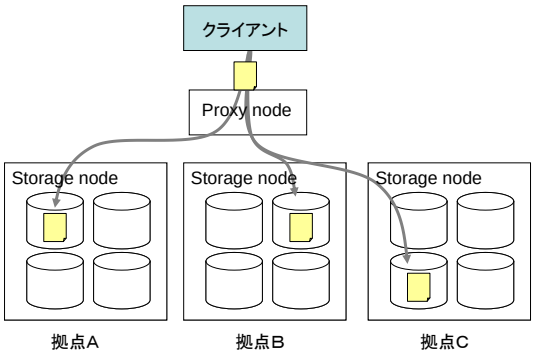

図1 OpenStack Object Storage の構成例

OpenStack Object Storage は処理を振り分ける Proxy node およびデータを蓄える Storage node から構成される.

Proxy node では格納するデータのメタデータから計算したハッシュ値に基づき，データを格納する Storage node を決定する. Storage node はノード内のハードディスクにデータを保持する. 例えば，冗長数 3 で信頼性設計を実施した場合，3 個の Storage node が選択され，同一のデータが格納される. 読み出し時には，3 個の Storage node のうち一つが選択され，データを読み出す.

図1では Storage node を配置する拠点を拠点 A，拠点 B，拠点 C と分けることで，データを地理的に分散配置する方法を示している. OpenStack Object Storage では，各拠点を別のリージョンとして指定することで，複製されたデータが必ず異なるリージョンに分散して配置するアルゴリズムが実装されている.

また，複数のクライアントからの要求を処理するために，複数の Proxy node を設置し，データ格納や読み出しなどの処理を分散することが可能である. 複数 Proxy node の配置により，スループットを向上させることが出来る.

2. 課題

2.1 信頼性および堅牢性

DOBSS は高い信頼性や堅牢性を実現する必要がある. 一般には冗長化構成などをとることで高信頼なシステムとして設計する.

たとえば，堅牢性(Durability)を「1年間にデータにアクセス可能な確率」と定義した場合，単体のハードディスク (HDD) の堅牢性は次の式で計算可能である.

$$Durability(d1) = \text{HDD の稼働率} = MTBF / (MTBF + MTTR) \quad \cdots \quad (1)$$

但し，ここで $MTBF$ を HDD の平均故障間隔，$MTTR$ を平均修理間隔とする.

冗長数を 3 とした場合のディスクシステム全体の Durability は，冗長化した全てのディスクが故障する確率であるため，次の式で計算される.

$$Durability(d3) = 1 - (1 - d1)\ \hat{}\ 3 \quad \cdots (2)$$

実際には，$MTTR$ の中には HDD を交換する時間に加えて，新しい HDD にデータを書き戻す時間も含める.

仮に利用する HDD が $MTBF = 600,000H$, $MTTR = 72H$ であった場合，式(2)より以下の値を得ることが出来る.

$$d3 = 0.999999999998273 \quad \cdots (3)$$

この場合，いわゆる「イレブンナイン」の堅牢性を実現可能であることが分かる.

このように，事前に設計を行うことで，所与の信頼性を持つストレージシステムを実現することが可能である.

2.2 バックアップの要求

2.1 章はハードウェアの故障に対しての堅牢性の計算であり，(A)人的操作ミスや

(B)ソフトウェアの故障などの要素は含んでいない．このため，従来から以下の対策がとられていた．

(A)人的な操作ミスに対しては，以下のアプローチが取られる．

- 定期的にデータセットのスナップショットを取得することで，スナップショット取得時のデータを復元することが可能となる．これによって誤操作のリカバリが可能である．
- データの変更管理を実施し，履歴情報として保持する．履歴をさかのぼることで，変更を記録した時点のデータまで復元可能である．

また，(B)ソフトウェア故障に対しては，スナップショットをテープなどの物理媒体や独立した別管理のストレージシステム上にコピーする方法が取られる．

上記とは別の要件として，高信頼性を求められるシステムにおいて，独立したデータの保管が義務付けられている場合がある．また，コンプライアンスの観点から，利用中のストレージシステムから独立した物理媒体やシステムに原本としてデータを保管することが求められる場合がある．

本稿では，冗長化された DOBSS 上のデータを外部のストレージシステムに効率的にコピーすることを課題としてとりあげることとした．

稼働中のシステムのある時点のスナップショットを別システムに格納するために，以下の要件を満足することが求められる．

- 完全性：ある時刻のデータの完全なコピーを作成できること．データに重複や欠損が無いこと．
- 信頼性：故障中のノードにデータがある場合にも，完全なバックアップを作成すること．
- 効率性：運用中のシステムになるべく負荷をかけないこと．
- 制御性：バックアップにかかる処理負荷の制御が可能であることが望ましい．

3. これまでの取り組み

外部のストレージシステムにデータのコピーを保管する場合，ストレージシステムの外部で稼働するバックアップアプリケーションから DOBSS に順次読み出し命令を発行し，データを読み出した上で更新の有無を確認し，外部ストレージに格納する方法が考えられる．

図２はバックアップアプリケーションによるバックアップ方式を図示したものである．図ではバックアップ先を DOBSS としているが通常のファイルストレージやテープメディアでも問題ない．

本方式では，Proxy node 経由でデータを取得するため，データの完全性は保たれるが，バックアップの処理は Proxy node を経由するため，バックアップ処理時に Proxy node に負荷がかかる．このため通常のサービスに影響を与える可能性がある．

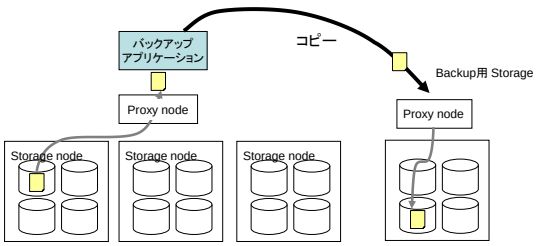

図２　バックアップアプリケーションによるデータのバックアップ方式

4. バックアップ方式の提案

本稿では，従来のバックアップアプリケーションによるバックアップ方式に対して，DOBSS の基本的な機能である Storage node のノード間同期機能を活用して，効率的にバックアップを取得する方法を提案する．

DOBSS では，格納したデータの不整合を発見する仕組みが実装されている．例えば，前述の OpenStack Object Storage においては，定期的にノード同士でデータを比較し，データの整合性を確認する仕組みを持つ．不整合が検出された場合には，多数決原理によってどのデータが誤っているかを検知し，正しいデータをコピーする機能（修復機能）を実現している．また，HDD の故障の際に，故障していない HDD にデータをコ

ピーすることで冗長度を保つ機能を有する．これらの処理はStorage node 上にノード間コピー機能として実装されている．ノード間コピー機能を拡張することでProxy nodeを経由せずに，Storage nodeから直接データのコピーを実現することが可能となる．

本稿ではノード間コピー機能を拡張してBackup用Storageにコピーする方式を提案する．図３にノード間コピー機能を拡張したバックアップ処理の実現イメージを示す．

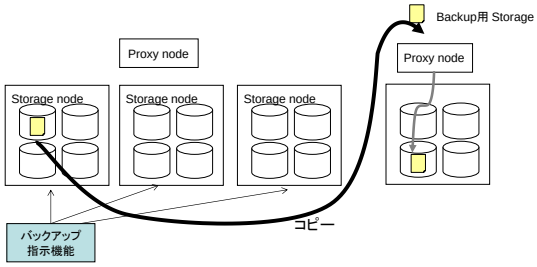

図３　ノード間コピー機能を拡張したバックアップ処理

バックアップ指示機能からは取得するスナップショットの時刻を指定し，それ以前に更新のあったデータのバックアップの実行を各Storage nodeに指示する．バックアップ指示に対して，要求を受けたノードでは以下の処理を実行する．

```
01 foreach (container) {
02   get container info from backup storage.
03   foreach (object) {
04    if (the object is updated) {
05      copy the object to backup storage
06    }
07   }
08 }
```

図４各ノードにおけるバックアップ処理（擬似コード）

上記の処理は全てのStorage nodeで並列実行することが可能である．また，処理の負荷を抑えたい場合には，ノードごとに順次実行することも可能である．

Storage nodeは保有するコンテナにわたり，全てのオブジェクトの更新をチェックする．ここで，コンテナとはデータを格納するフォルダのような概念である．データは必ずあるコンテナ内に保管され，コンテナは管理するデータの一覧を保有する．05行で更新のあったオブジェクトはBackup用Storageにコピーされる．更新のあったオブジェクトは複数のStorage nodeに冗長化して格納されているが，最初に04行の処理を行ったStorage nodeからだけコピーされる．一旦コピーが完了すれば，それ以外のStorage nodeは04行の処理で更新済みとしてコピーが実行されない．このため，更新のあったオブジェクトがBackup用Storageに一度だけコピーされる．

5. 評価方法および評価結果

4章で述べたアルゴリズムを，OpenStack Object Storageに実装して有効性の評価を行った．評価プログラムと構成の簡単化のため,バックアップ先もOpenStack Object Storageを使った．図５に評価システムの構成を示す．

5.1 評価条件

評価条件の概要を表１に示す．

表１　評価条件

項目	条件
バックアップ対象ストレージの構成	OpenStack Object Storage ・Storage node 数:6
バックアップ先ストレージの構成	OpenStack Object Storage ・Storage node 数:2
バックアップ方式	・一括バックアップ ・コンテナ指定バックアップ
ノードあたりのコンテナ数	5600 16800 28000 39200 50400
オブジェクトの変化率	・0% (変化なし) ・2.5%

上記の条件をベースにして，ノードあたりのコンテナの数を変えてバックアップにか

かる時間を測定した．今回は 3 種類の測定結果に関して報告する．

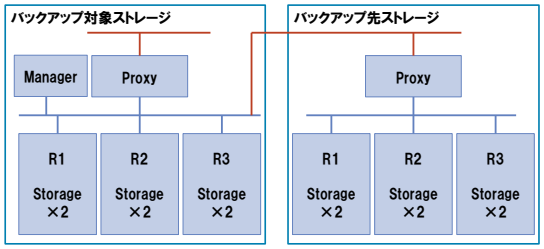

図５　評価システムの構成図

5.2 一括バックアップ

一括バックアップの処理特性を知るため，総登録データ数に対する更新データ数の比率を 2.5%で固定してデータ数(コンテナ数)を変化させて，処理にかかる時間を測定した．また，バックアップ先ストレージ側の処理性能が結果に影響しないように，r1, r2, r3 とリージョンごとに順番にバックアップの指示を行った．

本設定では，r1 から全てのデータがバックアップ用のストレージにコピーされ，r2, r3 は更新の確認のみが実行される．

測定結果を図６に示す．

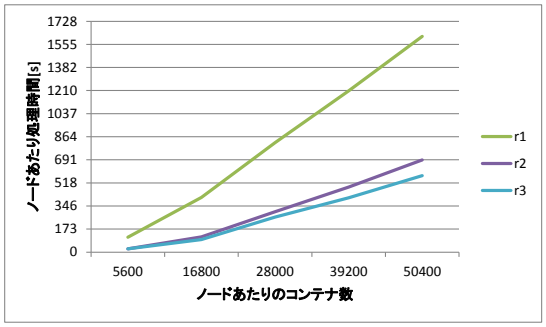

図６　一括バックアップ処理の実行時間
(差分 2.5%)

本測定結果から，データ量に比例してバックアップ時間がかかっていることが分かる．また，r1 と r2, r3 の比較から，コンテナの比較に処理時間の約 40%を費やしていることが分かる．

コンテナ数 50,400 のケースではコピー対象のコンテナはその 2.5%(1260)になる．r2 との処理時間の差（720 秒）が実際のコピー処理に使われた時間と考えられる．従って，コンテナ 1 個のコピーに要した時間は 0.57 秒になる．また，同様の計算から，コンテナの比較時間は 0.013 秒であった．

5.3 コンテナ指定バックアップ

5.2 章の検証から，コンテナのチェックに処理時間がかかることが分かったため，必要なコンテナだけを指定してバックアップする機能を実装した．

図 7 にコンテナ指定バックアップの検証の結果を示す．ここでは，差分比率を 1.25%とし，更新のあるコンテナ(640 個)のみを指定してバックアップを実行した．r1 は対象コンテナのチェック時間＋コピー時間を示している．図６に比べ，大幅にチェック時間を削減することが可能であることが分かる．

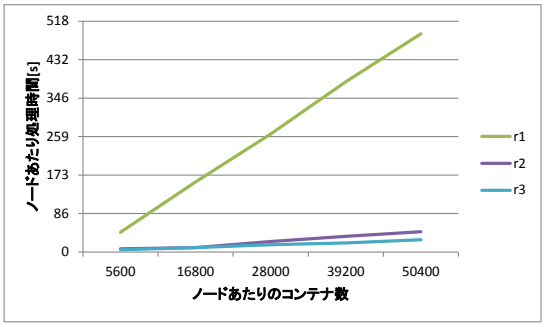

図 7 コンテナ指定バックアップ処理の実行時間(差分 1.25%)

5.4 一括バックアップ（差分なし）

比較処理の処理特性を確認するため，差分の無い場合の一括バックアップ処理の特性をした．測定結果を図８に示す．

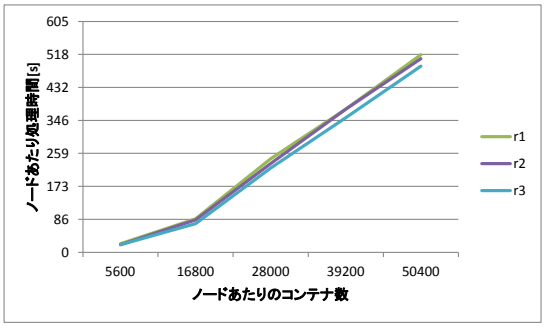

図８　差分の無い場合の一括バックアップ処理の実行時間

r1，r2，r3 とも，ほぼ同一の処理時間となっている.図 6 の r2, r3 と一致しており，コンテナ比較処理の処理時間を示している.

6. 考察

本方式はリージョンごとまたはノードごとにバックアップの指示を行うことが可能である．例えば，回線遅延の小さい地理的に近いリージョンのバックアップを先に指定することで，全体のバックアップ処理を効率化することが可能である.

処理にかかわる時間を減らし完了するタイミングを早めたい場合には，全ノードのバックアップ処理を同時に指定することが可能である. その際には, Backup 用 Storage に負荷がかかる可能性があるため，十分な性能を持つ装置を準備しておく必要がある.

7. 関連研究

DOBSS の冗長数を制御することで，DOBSS の処理を高速化する研究は多数ある.

例えば，Zhipeng らは文献 5 において DOBSS のリプリケーション方式やタイミング等を定式化した. Jindarak らは文献 6 において DOBSS に格納されたオブジェクトのリプリケーション数をオブジェクトのライフタイムに応じて制御することで高速化する方法を検討した.

本研究は，冗長数を変化させた場合でも，オブジェクトのバックアップを正確に効率よく取る方法を提供する．またバックアップにかかる Storage node 数を制御することによってシステム全体のバックアップにかかる負荷を制御することが可能であるため，上記の高速化手法と組み合わせて利用することが有効である.

8. まとめと今後の課題

本稿では, DOBSS に格納されたデータのバックアップを効率よく取得する方法を提案した．DOBSS のコンテナ同期機能を拡張することで，分散して負荷を下げたバックアップ処理が可能であることを示した.

また，OpenStack Object Storage 上に機能を実現し，有効性の確認を行った．実装に際しても，既存のノード間コピー機能のコードを活用することで，比較的容易に実装することが可能であった.

今後の課題として，Backup 用 Storage 側の容量削減のための圧縮機能の実現や，セキュリティ向上のための暗号化機能の実現などあげられる.

A. 参考文献

1) OpenStack Storage, The OpenStack Foundation, http://www.openstack.org/software/openstack-storage/
2) OpenStack Compute (Swift) in launchpad, The OpenStack Foundation, https://launchpad.net/swift
3) Swift's documentation, The OpenStack Foundation, http://docs.openstack.org/developer/swift/
4) Reliability mechanisms for very large storage systems, Qin Xin and Ethan L. Miller and Scott A. Brandt and et al, Proc. 20th IEEE / 11th NASA Goddard Conf. on Mass Storage Systems and Technologies, 2003, p.146-p.156
5) Dynamic replication strategies for object storage systems, Tan Zhipeng and Feng Dan, n *Proceedings of the 2006 international conference on Emerging Directions in Embedded and Ubiquitous Computing* (EUC'06)
6) Enhancing Cloud Object Storage Performance Using Dynamic Replication Approach, Kanatorn Jindarak and Putchong Uthayopas, In *Proceedings of the 2012 IEEE 18th International Conference on Parallel and Distributed Systems* (ICPADS '12).