Vol.2010-OS-115 No.7
2010/8/3

# 仮想化環境用クラスタストレージシステムの設計と実装

森田 和孝[†1] 藤田 智成[†1] 盛合 敏[†1]

リソースの有効利用やコスト削減の観点から, 仮想化技術によるサーバの統合が進んでいる. 仮想化環境用のストレージには, スナップショットなどの高度な管理機能を備えた SAN ストレージが用いられることが多い. しかし SAN ストレージは初期投資時に購入したモデルによって性能, 容量が一定の規模で頭打ちになってしまう. そのためサーバ数が多い環境では複数の SAN ストレージが必要になり, 管理が複雑になる. また, 高価なハイエンドのストレージを用いた場合, サーバ統合の目的のひとつであるコスト削減に結びつかない. そのため, 近年一般的になりつつある大規模の仮想化環境には不向きである. 本稿では, 大規模な仮想化環境においては, 大容量, 高性能を低コストで実現可能であるクラスタストレージシステムが有効であると考え, 大規模な仮想化環境用として適したクラスタストレージシステムの設計と実装について述べる.

# Design and Implementation of Cluster Storage System for Virtualization Environment

MORITA Kazutaka,[†1] FUJITA Tomonori[†1]
and MORIAI Satoshi[†1]

Server consolidation through virtualization is growing more common to reduce costs and use resources effectively. A SAN storage is often used for virtualization environment because it has useful features such as snapshotting volumes. However, the storage system cannot scale against hundreds of clients, so we need to have many storages in the huge virtualization environment, and it makes a cluster management hard. High-end storage systems could solve the problem, but it cannot achieve the cost reduction, which is one of goals of server consolidation. We believe that a cluster storage system is more suitable for a large environment, and give design and implementation of a cluster storage system for virtualization environment.

## 1. はじめに

リソースの有効利用やコスト削減の観点から, サーバ仮想化技術によるサーバの統合が進んでいる. サーバ仮想化環境用のストレージには, SAN ストレージが用いられることが多い. その理由として, サーバ仮想化環境のメリットであるライブマイグレーションを行うためには, 複数マシンからアクセス可能な共有ストレージが必要という点がある. ライブマイグレーションとは仮想マシンを無停止で他の実マシンへ移動させる技術であり, 実マシンのメンテナンスや障害対策などのためにも重要な技術である. また, ひとつの仮想マシンイメージを用いて別の仮想マシンを瞬時に作成する操作が仮想化環境においては重要であり, これを実現するために SAN ストレージのスナップショット機能やクローン機能が必要になるという点も大きな理由である. これらの理由から SAN ストレージはサーバ仮想化環境において重要なものとなっている. しかし SAN ストレージは初期投資時に購入したモデルによって性能, 容量が一定の規模で頭打ちになってしまうため, サーバ数が多い環境では複数の SAN ストレージが必要になり管理が複雑になる. また, 高価なハイエンドのストレージを用いた場合, サーバ統合の目的のひとつであるコスト削減に結びつかない. そのため, 近年一般的になりつつある数百台以上の規模の仮想化環境を構築するには不向きという問題がある.

本研究では, クラスタストレージシステムがサーバ台数を増やすことでクライアント台数に対してスケールアウトしやすいという性質と, 安価な PC を用いて構築可能であるという性質に着目し, 仮想化環境に特化したクラスタストレージシステムの実現を目指す. 本研究では大規模な仮想化環境において求められる要件を整理し, その要件を満たすためのクラスタストレージシステムの設計を提案する. そして, 実装したストレージシステムによって実験を行い, 設計した手法が有効であることを示す.

本稿の構成は以下のとおりである. まず第 2 節で大規模仮想化環境において求められるクラスタストレージシステムの要件を挙げる. 続く第 3 節でその要件を解決するための設計について説明し, 第 4 節で本稿での実装を説明する. 第 5 節で実装したクラスタストレージシステムの実験を行い, 第 6 節で関連研究について述べる. 最後に第 7 節で本稿をまとめる.

†1 NTT サイバースペース研究所
NTT Cyber Space Laboratories

## 2. 仮想化環境におけるクラスタストレージシステムの要件

本節では, 大規模な仮想化環境におけるクラスタストレージシステムの要件を挙げる.

### 2.1 高拡張性

仮想マシンは物理マシンと異なり, 容易に新しいマシンを作成可能である. そのため, 仮想化の利点を損なわないためには, クラスタストレージシステムにも仮想マシン作成と同様に柔軟な拡張可能性が求められる.

高い拡張性をもつクラスタストレージシステムであるためには, サーバ台数を増やすことで, 容量と性能を線形に拡張させることが望ましい. 本稿では近年一般的な環境となりつつある数百台規模のマシン台数までの拡張可能性を目指す.

### 2.2 高運用性

仮想マシンは物理マシンに比べて容易に管理できる. しかし, 仮想化環境に用いるクラスタストレージシステムの管理が煩雑であっては, 仮想化が運用の容易さに結びつかなくなってしまう. そのため, 仮想マシンの管理容易性と同様, ストレージに対しても高い管理容易性が求められる. 本稿では, 高運用性を実現するための要件として, 高度なディスク管理機能とクラスタの自律運用の二点を挙げる.

高度なディスク管理機能としては, 仮想ディスクのある時点での状態を瞬間的に保存するスナップショット機能, 既存の仮想ディスクと同一の新しい仮想ディスクを瞬間的に作成するクローン機能, 仮想ディスクの使用領域のみを物理領域に割り当てることでディスクの使用量を節約するシンプロビジョニングが挙げられる. これらは現在仮想化環境で用いられている SAN ストレージで提供されている機能でもあり, 仮想化環境用のクラスタストレージシステムではこれらを提供することが望まれる

クラスタの自律運用を実現するためには, クラスタ内のマシンに障害が発生した時に自動的にそのマシンを切り離して失われたデータを復旧する機能, クラスタのネットワークにマシンを追加すると自動認識してクラスタストレージシステムの一部とする機能, ディスクの使用量, 入出力の負荷を自動的に分散させる機能を実現する必要があり, これらを実現することにより管理者がクラスタストレージシステムの運用を容易にできるようになる.

### 2.3 高信頼性

仮想化環境では, 多数のサーバが集約されてストレージを利用しているため, ストレージに障害が発生した場合の影響が通常の環境に比べて大きい. そのため, 通常環境よりも高い信頼性が求められる.

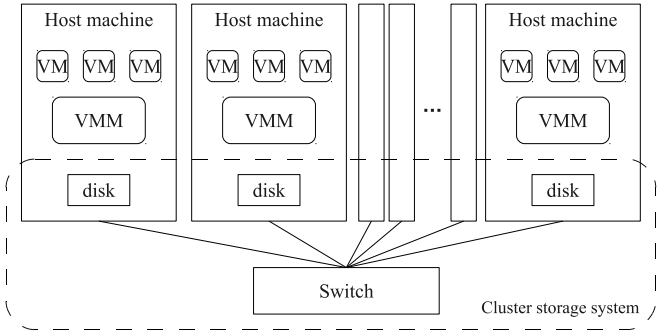


**図 1** クラスタストレージを用いた仮想化環境

高い信頼性を実現するためには, 任意のマシンの故障に対してもシステムが止まらない, つまり, 単一障害点がないクラスタストレージシステムを設計する必要がある. また, 仮想マシンがストレージから矛盾のあるデータを読まないように, データの一貫性を厳密に保証することも求められる.

## 3. 設　　計

本節では第 2 節で挙げた要件を満たすための, クラスタストレージシステムの設計について述べる.

### 3.1 全体構成

仮想化環境のストレージは SAN ストレージのように外部の記憶媒体を用いることが一般的である. これに対し, 本稿で提案するシステムは図 1 のように各マシンが自マシン内にもっているローカルディスクを用いて, 仮想マシンのホストマシンでクラスタストレージシステムを構成する. これは仮想化環境のクラスタとは別にストレージ用のクラスタを管理することを避け, 仮想化環境の運用性を下げないようにするためである.

本クラスタストレージシステムは, 全てのマシンを同じ役割とし, システム管理者に各マシンの役割を意識させないことで, 運用性を向上させる. また, どのマシンに障害が発生したとしても, システム全体が止まることはないようになっている. さらに, ボトルネックとなりうるメタデータサーバのような集中サーバが存在しないため, サーバ台数に対する高い拡張性が見込める.

本稿で提案するクラスタストレージシステムは, 仮想マシンからのみ利用可能であり, 仮

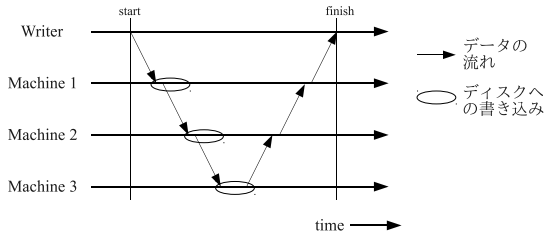


**図 2** 直列に複製を更新する処理の流れ

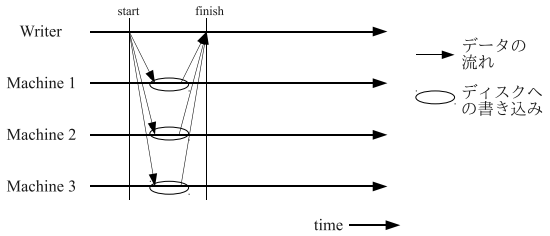


**図 3** 並列に複製を更新する処理の流れ

想マシンに対して仮想的なブロックデバイスのみを提供する. そのため通常のファイルシステムのように用いることはできない. 複数の仮想マシンが同時に同一の仮想ブロックデバイスにアクセスすることは許容せず, これにより通常の入出力時におけるロック処理を排除している. また, 提供するクラスタストレージシステムは第 3.2 節で解説するオブジェクトストレージを構成している.

### 3.2 オブジェクトストレージ

クラスタストレージシステムはオブジェクトと呼ばれる, 任意サイズのデータを保存する機能を持っている. オブジェクトにはシステム全体で一意である識別子が割り当てられている. オブジェクトを識別子で指定することで, 書込, 読込, 削除のオペレーションを行うことが可能である.

オブジェクトは自動的に複製されて保存される. 一貫性を保ちながら複製を更新するためには, 二相コミットやジャーナリングなどの技術を用いるか, Hadoop[9] や Ceph[11] などのように直列化させて複製を更新する必要がある（図 2）. これに対し, 本システムはひとつのオブジェクトに対して書き込みを行うことができる仮想マシンが高々ひとつであるため, 入出力処理の直列化が必要なく, 二相コミットなどを用いずに図 3 に示すように低遅延に複製を書き込むことができる. この手法では, 複製の更新中に仮想マシンに障害が発生した場合は複製の一貫性が崩れるが, 正常終了しなかった仮想マシンに対しては, その仮想マシンが次に起動した時に一貫性の復旧処理を行うことで対応する.

オブジェクトの配置はオブジェクトストレージ内で自動的に決まるため, クライアント側で保存先の指定をすることはできない. どのように保存先を決めるかについては, 第 3.4 節で述べる.

### 3.3 仮想ディスク

本システムが提供するオブジェクトには仮想ディスクオブジェクトとデータオブジェクトの二種類がある. 仮想ディスクは固定長に分割されてデータオブジェクトとして保存される. データオブジェクトはそのオブジェクトに対して最初に書込が行われた時に作成されるため, 作成された直後の仮想ディスクが持つデータオブジェクト数は 0 である.

全ての仮想ディスクは仮想ディスクオブジェクトを持っている. また, 仮想ディスクオブジェクトの中には, 割り当てられているデータオブジェクトの一覧を保存している. 仮想ディスクオブジェクトのオブジェクト識別子は仮想ディスク名とハッシュ関数によって計算して割り当てられており, 仮想マシンは起動後にまず仮想ディスクの名前から仮想ディスクオブジェクト識別子を求め, それを元にオブジェクトストレージから仮想ディスクオブジェクトを受け取る.

全てのデータオブジェクトはあるひとつの仮想ディスクオブジェクトに対応している. そして, データオブジェクトに対するデータの書き込み処理は, 対応する仮想ディスクを使用している仮想マシンのみが実行できる. その他の仮想マシンがそのオブジェクトに対して書き込み処理を行おうとした場合は, 新しくその仮想マシンが書き込み可能なデータオブジェクトを作成した上でその上にデータの書き込み処理が行われる.

各仮想ディスクオブジェクトはひとつの親仮想ディスクと複数の子仮想ディスクをもっており, 仮想ディスクオブジェクト全体は親子関係のリンクによって木構造を成している. 仮想ディスクオブジェクトの構造は図 4 のようになっており, 親仮想ディスクの仮想ディスクオブジェクト識別子, 子仮想ディスクの仮想ディスクオブジェクト識別子（複数）, その仮想ディスクに含まれるデータオブジェクト識別子の一覧を含んでいる.

仮想ディスクのスナップショット作成は, 既存の仮想ディスクに対して子仮想ディスクを

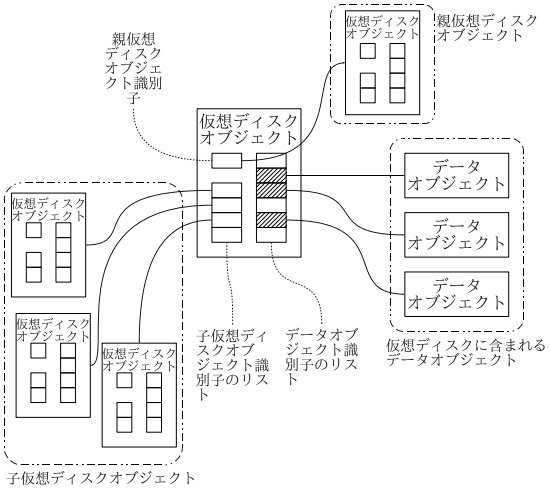


**図 4** 仮想ディスクオブジェクトの構造と他オブジェクトとの関係

作成することで実現される．まず, 新しい仮想ディスクオブジェクトを作成し, その後元の仮想ディスクオブジェクトの子仮想ディスクとして新しい仮想ディスクオブジェクトの識別子を, 新しい仮想ディスクオブジェクトの親仮想ディスクとして元の仮想ディスクオブジェクトの識別子を追加し, 保有するデータオブジェクト一覧をコピーする．

### 3.4 オブジェクトの配置

オブジェクトの冗長化を実現するためには, 各オブジェクトに対して保存先である複数のマシンを決定する必要がある．本設計ではコンシステント・ハッシュ法[3] を用いる．コンシステント・ハッシュ法は高い拡張性を実現し, 新しいマシンが参加した時, 既存のマシンが離脱した時のデータの移動量を少なく抑えられること, ハッシュの作用により自動的に負荷分散が実現できるという理由から, 本稿で提案するシステムには有用である．各オブジェクトのハッシュ値の計算には, オブジェクト識別子をハッシュ関数の入力として用いる．

クラスタストレージシステムの各マシンは動的に変化するマシン一覧情報の履歴を各マシンにローカルなストレージに保存している．このマシン一覧のバージョン番号を本稿では *epoch* と呼ぶ．本クラスタストレージシステムではオブジェクトの一貫性を守るために, オブジェクトの操作に対して, epoch を用いていくつかの制約を加えている．まず最初に, オブジェクトに対するすべての入出力処理は, 入出力要求の送信側と受信側で現在の epoch が等しくないと成立しない．そしてオブジェクトを更新する場合は, オブジェクトに現在の epoch を付加して保存する．マシンの参加や離脱によって現在の epoch が増えた時にも, ストレージに保存している既存のオブジェクトに最新の epoch を付加して更新する．また, 入出力処理が可能なオブジェクトは現在の epoch と同じ epoch で保存されているオブジェクトのみである．マシン構成変更直後で, epoch が等しいオブジェクトが存在しない場合は, そのオブジェクトが現在の epoch で更新されるまで入出力処理を中断する．

### 3.5 グローバル情報へのアクセス

本クラスタストレージシステムはひとつのデータオブジェクトに対して書き込み処理が可能な仮想マシンを高々ひとつとしているため, 通常の入出力処理においては個々の仮想マシンが他の仮想マシンと競合せずに独立にオブジェクトにアクセス可能である．しかし, 仮想マシンのロック情報, 新しい仮想ディスクオブジェクト識別子の割り当てなど, システムでグローバルな情報にアクセスする一部の操作に関しては他の仮想マシンと調整して行わなければならない．Hadoop などのように, メタデータサーバを用意すればこれは容易に実現できるが, 本稿が提案するクラスタストレージシステムは全てのマシンが同じ役割であり, かつ動的なクラスタメンバ構成を行うため, 別のアプローチが必要になる．

本クラスタストレージシステムでは, これを解決するために Totem single-ring protocol[1] を利用する．Totem single-ring protocol は全順序高信頼マルチキャストとメンバ管理を実現するプロトコルであり, 本稿ではこれを用いている．また, このプロトコルは同時に動的なマシン構成における死活管理も実現でき, 運用性の高いシステムを実現できる．全順序マルチキャストはコストの高い処理であるが, 本システムでは即時性が求められないごく一部の操作（グローバル情報へのアクセス）にのみこれを使用しており, 通常の入出力処理では利用しないため問題ない．

## 4. 実　　装

本節では第 3 節で取り上げた設計の実装について述べる．本実装の全体概要を図 5 に示す．

本実装のクライアントは qemu のブロックドライバとして動作する．Qemu は多くの種類の仮想ブロックデバイスを扱うことができ, ユーザからは本実装もそのうちのひとつとして扱うことができる．

オブジェクト識別子は 64 ビットの整数で表す．4 ビットはオブジェクトの種類（仮想ディスクオブジェクト, データオブジェクト）を定め, 24 ビットは仮想ディスクに対してシステ

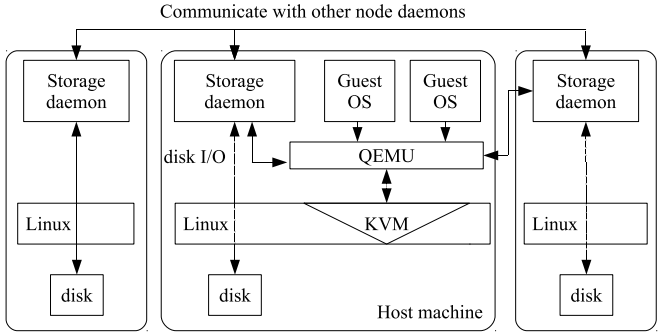


**図 5** 本クラスタストレージシステムの実装

ム全体で一意な値を割り当てるために用いられている．そのため本実装での最大仮想ディスク数はスナップショットの数を含め $2^{24} = 16,777,216$ 個である．20 ビットはデータオブジェクトのインデックス番号として用いてあり, 残りの 16 ビットを将来のために予約してある．データオブジェクトのサイズはデフォルトで 4 MB としており, 最大のディスクサイズは 4 MB $\times 2^{20} = 4$ TB である．この最大サイズは, データオブジェクトのサイズを変更することで拡張可能である．

各オブジェクトはディスク上のファイルとして保存されており, epoch などの付加情報はパスに含めることで実現している．クラスタのマシン構成が変更したときには epoch が更新されるため, ストレージシステム内の全オブジェクトを新しい epoch で保存しなおす処理が必要になるが, 本システムでは, 古い epoch のディレクトリから新しい epoch のディレクトリのオブジェクトへハードリンクを作ることでこれを高速に実現している．

本システムではハッシュ関数として Fowler-Noll-Vo ハッシュ関数[5] を用いている．この関数は SHA1 などの暗号化に用いられる関数とは違い, 逆元を求めることが難しくはないが, 今回の用途ではデータを適切に分散させるだけなので問題ない．

Totem single-ring protocol の実装には Corosync Cluster Engine[2] を用いている．Corosync は Pacemaker[10] などの実装にも用いられており, 実績があるため採用した．

## 5. 実　　験

本節では第 3, 4 節で述べた設計と実装が第 2 節で挙げた高拡張性を持っていることを実験によって示す．その他の要件に関する実験は今後の課題である．

実験環境は次の通りである．クラスタストレージシステムを構成するマシンは全て同じ性能であり, CPU は Intel Core 2 Quad Q6600 2.40 GHz, メモリ は 2 GB, ディスクは SATA 1 TB 7200 rpm, ネットワークはギガビットイーサネットである．ベンチマークプログラムには dbench 3.04 を利用し, ファイルとディレクトリの書込時に sync を行うオプションを用いた．ベンチマークプログラムは各仮想マシンの上で同時に実行させ, ベンチマーク結果の合計スループットを計測した．各仮想マシンのメモリサイズは 256 MB であり, 仮想マシンの台数は 1 台から 256 台まで変化させた．また, クラスタストレージシステムのデータ冗長度は 3 で設定し, 物理マシンの台数は 8, 16, 32, 64 台について実験を行った．さらに比較として, 商用の NFS ストレージを用いた時の性能の計測を行った．この NFS ストレージは 6 台の SAS ディスクによる RAID 6 構成であり, ギガビットイーサネットケーブル 2 本 (2 Gbps) でクラスタに接続されている．

実験結果は図 6 の通りである．なお, ローカルディスクに対して 1 台の物理マシンでベンチマークを行った結果は 34.1 MB/sec であり, NFS ストレージに対して 1 台の物理マシンでベンチマークを行った結果は 32.8 MB/sec である． クラスタストレージは台数が少ない時には仮想マシン数に対して線形に性能が拡張する．また, クラスタのマシン数が増えるにつれて, 線形に性能が拡張する仮想マシン数が多くなり, 頭打ちとなる性能の限界も高くなっていることがわかる．一方集中型アーキテクチャである NFS ストレージは仮想マシン数 4 台程度まではクラスタストレージより高い性能を示すが, 仮想マシン台数の増加に対しての拡張性は見られなかった．

## 6. 関 連 研 究

PC クラスタを用いたストレージシステムとしては, GlusterFS[8] や Lustre[7], Ceph[11] などがある．これらの分散ファイルシステムはシステムに含まれるマシンとそれらの役割をすべて設定ファイルに記述する必要があるが, 本稿のシステムは運用にとって容易なように設計されており, ノードの役割がないこと, マシンメンバの構成を動的に変更できることなどからこのような設定ファイルは不要である．また本ストレージシステムはスナップショットなどのディスク管理機能を持つことができるように設計されており, さらに, 集中サーバがないことからより高い拡張性を実現可能である．

サーバ仮想化環境用のストレージシステムとしては, Xen 用のストレージである Parallax[4] と, VMware 用のストレージである Ventana[6] がある．Parallax も Ventana もサーバ仮想化環境用ストレージの条件として高速なスナップショットが取得できることをあげており,

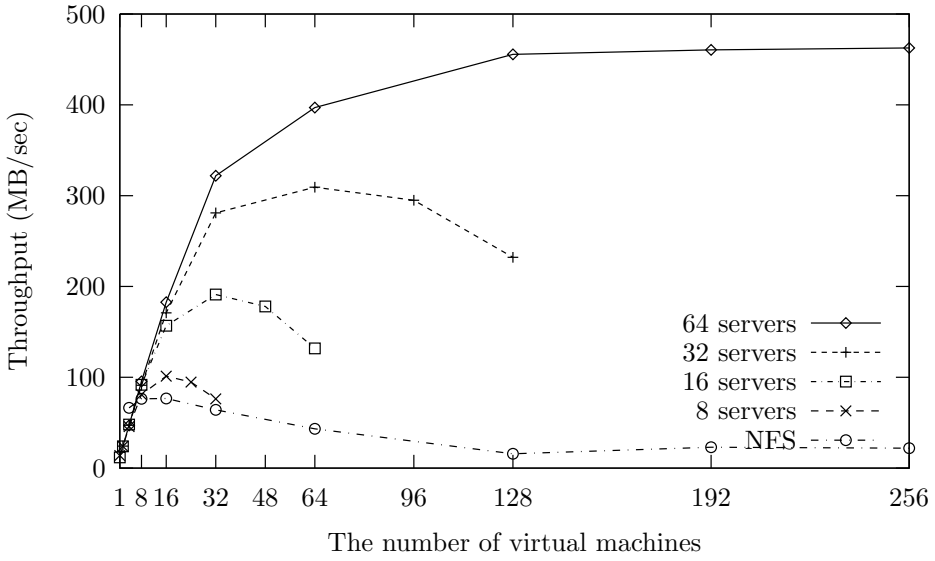


**図 6** 本クラスタストレージシステムの拡張性実験結果

それについて取り組んでいるが, Parallax はクラスタストレージシステムではないため, 共有型ストレージと同様の問題があり, Ventana は集中サーバが存在するため, その他のクラスタストレージシステムと同様の問題がある.

## 7. おわりに

本稿では, サーバ仮想化環境のためのクラスタストレージシステムにおける要件として, 高拡張性, 高運用性, 高信頼性の三点を挙げた. そしてこれらの要件を満たすために, すべてのマシンの役割が等質なアーキテクチャを提案し, コンシステント・ハッシュ法, totem single-ring protocol によってシステムの設計を示した. 本設計ではひとつのオブジェクトにアクセスする仮想マシンを高々ひとつに制限したことによりロックを排除し, そしてブロックのみを提供することで, シンプルで高信頼なシステムを実現した. 本システムはすべてのマシンの役割が同じで, 動的なメンバ構成が可能である設計であることから, 設定が不要な, 高運用性システムを実現した. 本システムは SAN ストレージと同様, 高度なディスク管理機能も実現した. さらに, 通常の入出力においてロックを排除し, ボトルネックとなりうる集中サーバを排除したことで, 高性能で高拡張性を持ったシステムを実現した. また, qemu や corosync を用いた実装についても述べた.

今後はより詳細な性能の確認, とくに本設計, 実装が大規模仮想化環境用クラスタストレージの要件を実際に実現できているかの実験を行う. また, 仮想化環境用管理ツールと組み合わせた時の運用性の確認, 高負荷時での性能や信頼性の保証ができているかどうかの実験を行う. 本稿の実装は `http://www.osrg.net/sheepdog/` にある.

## 参 考 文 献

1) Amir, Y., Moser, L.E., Melliar-Smith, P.M., Agarwal, D.A. and Ciarfella, P.: The Totem single-ring ordering and membership protocol, *ACM Trans. Comput. Syst.*, Vol.13, No.4, pp.311–342 (1995).
2) Dake, S., Caulfield, C. and Beekhof, A.: The Corosync Cluster Engine, *Proceedings of the 2008 Linux Symposium*, pp.85–99 (2008).
3) Karger, D., Lehman, E., Leighton, T., Panigrahy, R., Levine, M. and Lewin, D.: Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web, *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, New York, NY, USA, ACM, pp.654–663 (1997).
4) Meyer, D.T., Aggarwal, G., Cully, B., Lefebvre, G., Feeley, M.J., Hutchinson, N.C. and Warfield, A.: Parallax: virtual disks for virtual machines, *Proceedings of the 4th ACM European conference on Computer systems*, pp.41–54 (2008).
5) Noll, L.C.: Fowler/Noll/Vo (FNV) hash (2004).
6) Pfaff, B., Garfinkel, T. and Rosenblum, M.: Virtualization aware file systems: getting beyond the limitations of virtual disks, *Proceedings of the 3rd conference on Networked Systems Design & Implementation*, Berkeley, CA, USA, USENIX Association, pp.26–26 (2006).
7) Schwan, P.: Lustre: Building a file system for 1000-node clusters, *Proceedings of the 2003 Linux Symposium* (2003).
8) The Gluster Community: GlusterFS, `http://www.gluster.org/docs/index.php/GlusterFS`.
9) The Hadoop Community: Welcome to Hadoop! , `http://lucene.apache.org/hadoop/`.
10) The Pacemaker Community: Pacemaker, `http://www.clusterlabs.org/`.
11) Weil, S.A., Brandt, S.A., Miller, E.L., Long, D. D.E. and Maltzahn, C.: Ceph: a scalable, high-performance distributed file system, *Proceedings of the 7th conference on USENIX Symposium on Operating Systems Design and Implementation*, USENIX Association, pp.22–22 (2006).