情報処理学会論文誌　コンピューティングシステム　Vol.5　No.2　23–32 (Mar. 2012)

Sheepdog：仮想マシンのための対称型クラスタストレージ

森田 和孝[1,a)]　藤田 智成[1]　盛合 敏[1]

受付日 2011年7月19日, 採録日 2011年10月27日

概要：近年，クラウドコンピューティングの普及などにより，大規模なサーバ仮想化環境を構築する事例が増えている．サーバ仮想化環境におけるストレージには，複数の物理マシンから同一の領域にアクセスできることが求められるため，SAN ストレージを用いることが多い．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，性能や容量が一定の規模で頭打ちになってしまう．そのため，大規模なサーバ仮想化環境では複数の SAN ストレージが必要になり，管理が複雑になる．また，高信頼で大容量な SAN ストレージは，導入に要するコストが非常に高くなってしまう．本稿では，既存の SAN ストレージの代替として，安価な PC で構築可能な，仮想マシン用クラスタストレージ Sheepdog について述べる．Sheepdog は対称型のクラスタ構成で動作し，仮想マシンに高信頼な仮想ディスクを提供する．そして大規模な仮想化環境においても性能が低下せず，かつ，運用性を損なわないよう設計されている．Sheepdog に運用環境を模擬した負荷をかけて実験を行ったところ，既存クラスタストレージに用いられている実装や SAN ストレージに比べて，高い性能と高い拡張性が得られることを確認した．

キーワード：分散システム，ストレージ，仮想化

Sheepdog: Symmetric Clustered Storage for Virtual Machines

Kazutaka Morita[1,a)]　Tomonori Fujita[1]　Satoshi Moriai[1]

Received: July 19, 2011, Accepted: October 27, 2011

Abstract: Virtualization environments become larger because of the emergence of cloud computing. A SAN storage is well used for those environments to provide a shared storage device for virtual machines. However, it can be a bottleneck under the large-scale environments because of its centralized architecture. In addition, a high-end storage is much more expensive than commodity hardware. We have designed and implemented Sheepdog, a symmetric clustered storage for many virtual machines. It provides reliable virtual disks with commodity hardware, and it aims to scale to hundreds of machines without losing manageability. Our results show that Sheepdog can achieve higher performance than existing approach and product under realistic workloads.

Keywords: distributed system, storage, virtualization

1. はじめに

近年，クラウドコンピューティングの普及などにより，大規模なサーバ仮想化環境を構築する事例が増えている．サーバ仮想化環境において，仮想マシンに提供する仮想ディスクのストレージには，SAN ストレージを用いることが主流である．その理由として，仮想マシンのライブマイグレーションを用いるためには，複数マシンから同時にアクセス可能な共有ストレージが必須であるという点がある．また，SAN ストレージが持つ様々なストレージ仮想化技術が，仮想化環境の運用を容易にするということも理由の 1 つである．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，性能や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，高信頼で大容量，高性能な SAN ストレージは，導入に要するコストが非常に高くなるため，事前に仮想化環境の規模を正

[1] NTT サイバースペース研究所
NTT Cyber Space Laboratories, Yokosuka, Kanagawa 239–0847, Japan
[a)] morita.kazutaka@lab.ntt.co.jp

確に見積もることができない場合には，導入することが難しい．そのため，近年一般的な数百台以上の物理マシン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．

本稿では，既存の SAN ストレージの代替として，安価な PC で構築可能な，仮想マシン用クラスタストレージ Sheepdog について述べる．Sheepdog は単一障害点のない対称型のクラスタ構成で動作し，大規模環境においても運用性を損なわないよう設計されている．すべてのストレージサーバは同じ役割であるため，管理者は運用時に各ストレージサーバの役割について意識する必要がない．管理者は任意のサイズの仮想ディスクを Sheepdog 上に作成し，仮想マシンに提供することができる．Sheepdog の仮想ディスクは，任意のホストマシンからアクセスすることができるため，仮想マシンのライブマイグレーションを行うことも可能であり，さらに SAN ストレージと同様にスナップショット機能やクローン機能も実現している．また，管理者が Sheepdog のクラスタに加えるマシンを設定ファイルなどで静的に指定する必要はなく，動的な物理マシン構成で自律的に動作させることが可能である．そのため Sheepdog のデーモンが動作しているマシンをクラスタのネットワークに追加すると，自動認識されてクラスタストレージのマシンに加えることができる．また，障害が発生したときには自動的にそのマシンを切り離し，失われたデータは復旧される．Sheepdog のデータは複数のマシンに冗長化されて保存されているので，どの物理マシンに障害が発生しても，データが失われたりシステムが止まったりすることはない．Sheepdog はこれらの特徴を維持しつつ，個々の仮想マシンにローカルディスクと同等の性能を提供し，クラスタストレージ全体のトータル性能において SAN ストレージ以上の性能実現を目指している．そして Sheepdog は数台の小規模環境から数百台規模の大規模環境まで対応し，仮想ディスクの性能と容量を線形にスケールさせることを目指している．

本稿の構成は以下のとおりである．まず 2 章で Sheepdog のアーキテクチャについて説明する．続く 3 章で Sheepdog クラスタ内のマシン管理について述べ，4 章で Sheepdog が仮想マシンモニタに提供するオブジェクトストレージを説明する．5 章で Sheepdog の性能評価実験を行い，6 章で関連研究について述べる．最後に 7 章で本稿をまとめる．

2. アーキテクチャ

本稿で提案するクラスタストレージ Sheepdog の全体構成を**図 1** に示す．Sheepdog は運用性向上のため，完全に対称的なクラスタ構成によって仮想化環境を実現することを目指している．まず，Sheepdog は外部の記憶媒体を使わずに，各ホストマシンが自マシン内に持っているローカルディスクを利用し，ホストマシンのみでクラスタストレージを構成する．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による運用の負担を避けるためである．Sheepdog は，すべ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．また，特別な役割の集中サーバは存在しないため，一部のマシンに性能が高いマシンを用意する必要もない．そして，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設計になっている．Sheepdog にはクラスタストレージを構成するマシンに関する設定は存在せず，クラスタに加わったマシンを自動的に認識してクラスタストレージを構成する．Sheepdog のネットワーク内でストレージデーモンを立ち上げると，そのマシンは自動的にクラスタストレージに追加され，データは自動的に負荷分散される．また，障害が発生したマシンは自動的にクラスタストレージから取り除かれ，故障したマシンに保存されていたデータは別マシンに自動的に復旧される．また，Sheepdog は 1 つの巨大なストレージ空間を構築し，どのホストマシンからでもすべての仮想ディスクにアクセスすることが可能である．本稿では仮想マシンに提供される仮想ディスクのことを VDI（Virtual Disk Image）と呼ぶ．

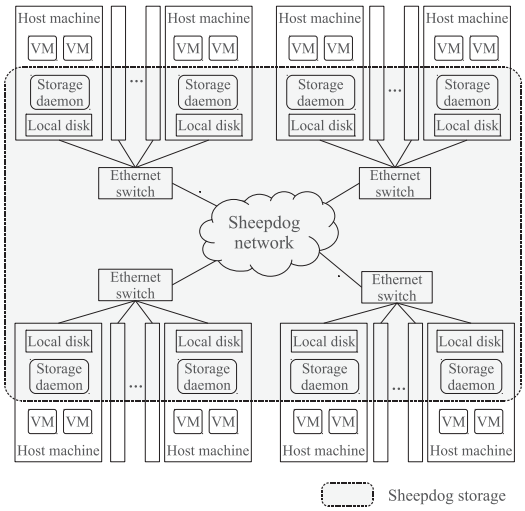

図 1　Sheepdog クラスタの全体構成
Fig. 1　Overall architecture of Sheepdog.

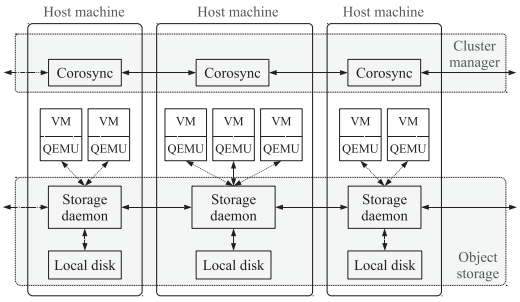

図 2　物理マシン内の構成
Fig. 2　Host machine components.

Sheepdog の各物理マシン内の構成を**図 2** に示す．Sheepdog はストレージのクライアントを QEMU のブロックドライバとして実装しており，QEMU ベースの仮想マシンか

ら利用可能である．Sheepdog は仮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ことでシンプルな設計を実現している．また，Sheepdog は複数の仮想マシンが同時に同一の VDI にアクセスすることを禁止している．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，高速化，実装の単純化を実現している．すべての VDI にはクラスタ全体で一意な文字列の名前がついており，ユーザは QEMU プログラムの引数に VDI の名前を指定することで，仮想マシンに Sheepdog の VDI を利用させることできる．Sheepdog は 3 章で説明する仮想同期を用いてクラスタを管理している．また，4 章で解説するオブジェクトストレージを構成して QEMU にストレージ空間を提供している．

なお，Sheepdog の設計自体は QEMU 以外の仮想マシンモニタや OS からでも仮想ディスクとして利用可能なものである．KVM を有効にした QEMU でも動作し，また，Xen の blktap や Linux の仮想デバイスとして Sheepdog のクライアント部分を実装することも可能である．

3. クラスタ管理

分散システムにおいて，マシンの死活監視を行うためやマシン間で合意を得るためには，集中管理を行う専用のサーバを用意することが多い．しかし，Sheepdog は対称型構成を実現するために，このような構成は避け，クラスタに含まれるマシンの管理に，仮想同期を用いて死活監視や合意を実現している．仮想同期はマシン間でメッセージのやりとりと死活監視を行うことができる技術であり，アトミックかつ高信頼で，全順序なマルチキャストをクラスタ全体に送信することができる．また仮想同期は，マシンの追加と離脱の検出を行い，マルチキャストメッセージと矛盾が起きない順序で全マシンに通知できる．仮想同期のスケーラビリティに関しては様々な研究 [1], [8] がなされており，数百台以上でも動作させることが可能な技術であることから，Sheepdog の用途にも適用可能である．Sheepdog は Pacemaker [17] などの実績がある死活監視ソフトウェアに採用されている，Corosync [6] を用いている．Corosync は Totem single-ring protocol [3] という高速な仮想同期を実現する技術を実装しているライブラリである．クラスタマシン全体で 1 つのリングを構成し，リング上でトークンを回して状態遷移をさせていくことで仮想同期を実現している．Corosync のマルチキャストメッセージは，通常の IP マルチキャストで全体に送られるため，非常に高速に動作する．以下，Sheepdog が Corosync を用いてどのようにクラスタ全体の管理を行っているかについて述べる．

3.1 物理マシン一覧の管理

Sheepdog のネットワークに新しく加わったマシンは，Corosync の仮想同期機能によって自動的に Sheepdog クラスタに認識される．クラスタにマシンが参加，離脱するたびに，クラスタ内の全マシンにその情報が通知され，Sheepdog の各ノードはマシン一覧の履歴をすべてローカルのディスクに保存する．仮想同期によって，すべてのマシンに同一の順序でマシン一覧の変更が通知されるため，Sheepdog 内のマシンが持つマシン一覧の履歴はすべて同じになる．マシン一覧の履歴を持つことで，不測の事態でクラスタ全体が停止してしまっても，各マシンのマシン構成履歴を合わせることで最後のクラスタの状態を探索することができ，安全にクラスタストレージを再開することができる．Sheepdog では，マシン一覧履歴のバージョン番号を epoch と呼んでおり，Sheepdog に保存されているデータの一貫性を保つうえで重要な役割を果たしている．その詳細は 4.4 節で述べる．

ネットワークの分断が発生した場合には，全マシンに同一のマシン構成が通知できなくなる．そのとき，分断された両ネットワークで別々にストレージのデータを更新するとデータの不整合が発生する可能性がある．この問題に対し，Sheepdog でネットワーク分断が発生した場合は，分断前のマシン数に比べて過半数のマシン数が所属しているネットワークのマシンでクラスタストレージを継続し，少数側のネットワークに属しているマシンはクラスタストレージを停止させることでデータの一貫性を保証する．ネットワーク分断後は，少数側のネットワークに属している仮想マシンからは，仮想ディスクへのアクセスはすべて I/O エラーとなる．また，同様に，多数側に属する仮想マシンから，少数側のネットワークにしか存在しないデータへのアクセスに関しても，I/O エラーとなる．これらはデータの一貫性を保証するために必要な制約である．分断されたネットワークが元に戻ると，基本的には多数側クラスタにあるデータを用いて，複製を再配置することで復旧が行われる．しかし，分断中に更新されなかったデータや，少数側クラスタにしか存在しないデータに関しては，少数側クラスタのデータも用いて復旧が行われる．分断中にデータの更新がされているかどうかは，データが保存されたときの epoch を調べることで確認できる．epoch を用いたデータ一貫性保持の詳細については 4.4 節で述べる．

3.2 分散ロック

Sheepdog は 1 つの VDI を複数の仮想マシンが同時に利用することを許容していないため，個々の仮想マシンが他の仮想マシンと競合せずに VDI にアクセス可能である．そのため，仮想マシンの入出力処理においてロック機構は必要ない．しかし管理者が行う一部の操作にはクラスタ内の物理マシン間で排他制御を行うための分散ロック機構が必要になる．たとえば，同時に同じ名前の VDI が作成されることを防ぐときや，複数の仮想マシンが同時に VDI にアクセスすることを防ぐときなどの VDI 管理の操作が該当す

る．高信頼な分散ロックを提供する技術として Chubby [5] や ZooKeeper [10] があるが，Sheepdog はすべて同じ役割のサーバで運用されることを目指しており，これらの外部分散システムは利用しない．Sheepdog はロック要求を仮想同期のマルチキャストで送信することでロック処理を行う．同時に複数の仮想マシンが同一 VDI のロック要求を出しても，全マシンには同じ順序でロック要求が届くため，最初に届いたロック要求のみを成功させればよい．この性質は仮想同期マルチキャストの全順序性によって保証されている．Sheepog が用いている Corosync は，マルチキャストによって受信したメッセージをバッファリングし，Totem single-ring protocol によって，順序性について合意がとれたメッセージから Sheepdog に送ることでこれを実現している．

4. オブジェクトストレージ

QEMU のブロックドライバからは，Sheepdog のクラスタストレージはオブジェクトストレージとして見える．オブジェクトストレージとは，可変長のデータ（オブジェクト）を保存する機能を持つストレージであり，オブジェクトの保存位置をクライアントが指定せず，サーバ側で決めるという特徴がある．各オブジェクトにはシステム全体で一意である 64 bit の整数（オブジェクト ID）が割り当てられており，クライアントは，オブジェクトの ID を指定するだけで，オブジェクトの作成，読み込み，書き込み，削除の操作を行うことが可能である．

Sheepdog のオブジェクトは，writable オブジェクトと read-only オブジェクトの 2 種類に分類される．Writable オブジェクトは 1 つのクライアントからのみ，書き込みと読み込み両方の要求を受付けるオブジェクトである．同時に複数のクライアントからは，I/O 処理を受付けることができない．そのため，Sheepdog のオブジェクトストレージにおいては，書き込み処理の衝突が発生せず，非常にシンプルな実装になる．Sheepdog では VDI が同時に複数の仮想マシンから利用されることがないため，このような設計が可能となっている．Read-only オブジェクトはすべてのクライアントから読み込み処理可能であるが，どのクライアントからも書き込み処理ができないオブジェクトである．Read-only オブジェクトへの更新要求はコピーオンライトとして処理される．つまり，新しく writable オブジェクトを作成したうえで，そのオブジェクトに対して書き込み処理が行われる．Read-only オブジェクトは VDI のスナップショットで利用されている．

4.1 VDI

Sheepdog の VDI は，VDI オブジェクトとデータオブジェクトの 2 種類のオブジェクトで構成されている．VDI の実データは固定長（デフォルトで 4 MB）に分割されて，データオブジェクトとしてオブジェクトストレージに保存されている．そして，VDI がどのデータオブジェクトを持っているかに関するメタ情報が VDI オブジェクトに保存されている．VDI オブジェクトの構成は**表 1** のとおりである．

表 1　VDI オブジェクトに含まれる情報
Table 1　VDI object.

名前	内容
vdi_id	VDI ID
name	VDI の名前
ctime	VDI 作成日時
vdi_size	VDI のサイズ
nr_copies	データ冗長度
block_size_shift	データオブジェクトのサイズ
parent_vdi_id	親 VDI の VDI ID
child_vdi_id	子 VDI の VDI ID のリスト
data_vdi_id	データオブジェクトのリスト
（以下はこの VDI がスナップショットの場合に使用）	
tag	スナップショットの名前
snap_xctime	スナップショットを作成した時間
snap_id	スナップショット ID

各 VDI には VDI ID と呼ばれる識別子が割り当てられている．VDI ID はオブジェクト ID と同様にクラスタ全体で一意であり，VDI 作成時に割り当てられる．VDI ID 割り当て時の ID の衝突は，仮想同期による分散ロックでクラスタ全体をロックすることで防いでいる．Sheepdog はクラスタ起動時に，クラスタに保存されている全 VDI オブジェクトを調べることで，VDI 名と VDI ID の対応表を作成し，メモリ上にその表を保持する．その表は全物理マシン上で作成されるため，仮想マシンはどの物理マシンからでも目的の VDI ID を VDI の名前から得ることができる．VDI オブジェクトの ID は VDI ID から計算できるようになっており，仮想マシンは VDI の名前で目的の VDI オブジェクトにアクセスすることができる．

VDI 全体はスナップショットの親子関係のリンクによって木構造（VDI 木）をなしている．VDI 木において，分岐にある VDI が read-only（スナップショット VDI）であり，葉にある VDI が writable（非スナップショット VDI）である．また，スナップショット VDI に割り当てられているデータオブジェクトはすべて read-only である．VDI のスナップショット作成は，VDI 木の葉にある writable VDI に対して新しい子 VDI を作成することで実現される．また，VDI のクローン作成は，VDI 木の枝にある read-only VDI に対して，新しい子 VDI を作成することで実現される．これら新規の子 VDI には親 VDI と同じデータオブジェクトのリストがコピーされるが，このデータオブジェクトはすべて read-only であるため，子 VDI に対する書き込み要求はすべてコピーオンライトとなる．そのため，ス

ナップショット VDI が参照するデータオブジェクトは必ず不変である．

4.2　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．Sheepdog は集中管理サーバを持たない対称型設計なので，集中サーバなしでデータ配置を決定可能なコンシステント・ハッシュ法 [11] を用いている．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ときや既存のマシンが離脱したとき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，自律的に動作することを目指している Sheepdog に適している．各オブジェクトのハッシュ値の計算には，オブジェクト ID をハッシュ関数の入力として用いる．コンシステント・ハッシュ法のリング上には各物理マシンが仮想ノードをデフォルトで 64 個ずつ持っている．この値は各マシンのディスクの空き容量によって増減し，仮想ノード数の変更は仮想同期の高信頼マルチキャストで全体に周知される．仮想ノードの各ノード ID は物理マシンの IP アドレスのハッシュ値によって自動的に計算される．

Sheepdog ではハッシュ関数として，高速に計算が可能な Fowler-Noll-Vo ハッシュ関数 [13] 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，ハッシュ値が一様に分布すれば問題ない．

4.3　複製

オブジェクトストレージにおいて，オブジェクトは自動的に複製されて保存される．Sheepdog のオブジェクトストレージに保存されるデータは，ブロックデバイスに用いられるためデータの一貫性が重要であり，1 度書いたデータを次に読むときには必ず最新のデータが返らなくてはならない．この性質を保証しながら複製を更新する技術として，primary-copy 方式 [2], chain 方式 [18], そしてこれら 2 つの手法を組み合わせた splay 方式 [19] がある．これらの方式の書き込み処理の流れを**図 3** に示す．図 3 において，たとえば splay 方式は writer がまずデータを machine 1 に転送し，その後 machine 1 は受け取ったデータを自分以外の複製サーバに転送する．ディスク書き込みの完了通知はすべて machine 3 に送信され，すべてのマシンから完了通知を受け取った段階で，machine 3 は writer に完了通知を送信する．

primary-copy 方式は write の遅延が複製の数に依存しないで固定であるが，write と read の要求を同一マシン（図 3 のマシン 1）に送らなくてはならない．一方 chain 方式は

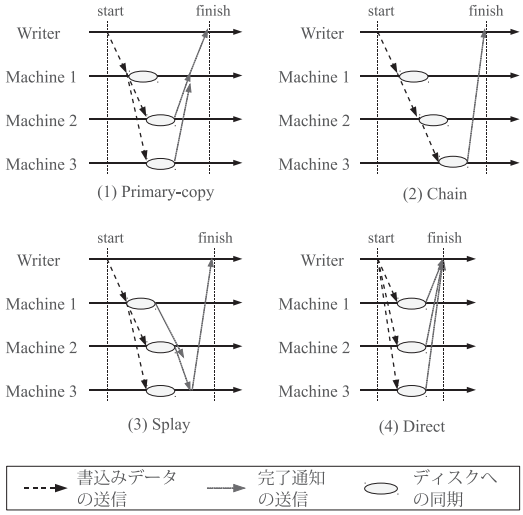

図 3　各複製方式の書き込み処理の流れ
Fig. 3　Write flow of data replication.

write リクエストと read リクエストが別のマシンへ送られるため，負荷の分散が可能になるが，write の遅延が複製の数に比例して大きくなってしまう．この両方の利点を兼ね備えたのが splay 方式であり，write の遅延を一定にしたうえで，read と write の負荷分散が可能な手法である．

これらの手法は，複数のクライアントが同時に書き込み要求を行ったとしても，データの一貫性を壊さない方式であるが，Sheepdog は 1 つのオブジェクトに対して書き込みを行う仮想マシンがたかだか 1 つであるため，仮想マシンが書き込み処理のコーディネータになることができ，ストレージ側でコーディネーションを行う必要がない．そのため，直接並列に書き込みが可能であり，primary-copy 方式や splay 方式よりもさらに低い遅延で書き込み可能である．またどのマシンからも読み込み処理が可能である．複製によってデータを冗長化している環境においては，書き込みのコストが高くなるため，書き込み処理の高速化は特に重要な要素である．Sheepdog の write 処理は，TCP によって接続されたストレージサーバに順に write(2) システムコールによってリクエストを送り，poll(2) システムコールによって I/O リクエストの完了を待つことで並列に行われる．

4.4　マシン故障時の一貫性

Sheepdog ではオブジェクトのデータ一貫性を守るために，オブジェクトを更新する際に，オブジェクトに現在の epoch を付加して保存する．これは Sheepdog がクライアントに古いデータを送らないためである．**図 4** を例に説明する．マシン A，B，C の構成で Sheepdog が動いているときに，あるオブジェクトが B，C のマシン上で更新されたとする（epoch 2）．その後，新しくマシン D，E が Sheepdog に追加されて，先ほど更新されたオブジェクトの保存場所が，マシン D，E に移ったとする（epoch 3）．

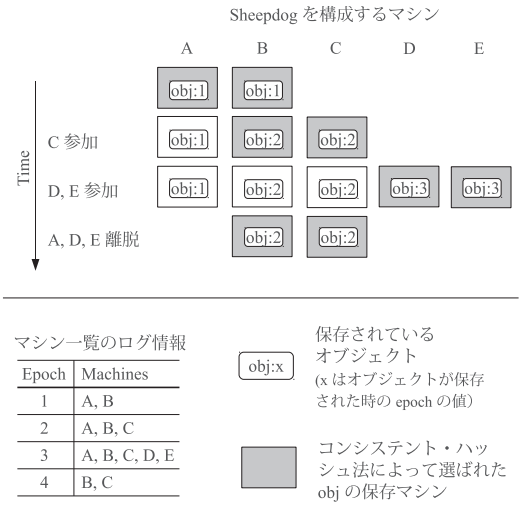

図 4　マシン情報の履歴
Fig. 4　Machine membership history.

そしてマシン D，E に障害が発生して，これらのマシンが離脱した場合（epoch 4），マシン B，C が持つオブジェクトは最新のデータでない可能性があり，これを検出する必要がある．この epoch 4 の状況において，仮想マシンがマシン B，C に保存されている epoch 情報 2 のオブジェクトに対してアクセスを行うと，Sheepdog はこのオブジェクトが epoch 3 で更新されている（最新でない）可能性を考慮し，仮想マシンにディスク I/O エラーを返す．このオブジェクトは最新データを持っている可能性があるマシン D，E のいずれかが復帰するまでは仮想マシンからアクセスできなくなる．これらは厳しい制約ではあるが，データの一貫性を保証するために必要な制約である．

Sheepdog は各オブジェクトをディスク上のファイルとして保存しており，epoch などの付加情報はパスに含めることでこれを実現している．クラスタのマシン構成が変更されたときには epoch が更新されるため，クラスタストレージ内の全オブジェクトを新しい epoch 情報を付加して保存しなおす処理が必要になるが，Sheepdog では，古い epoch のオブジェクトのパスから新しい epoch のオブジェクトのパス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．

5. 実験

Sheepdog が用いている手法が有効であることを示すため，実験を行った．実験環境は**表 2** のとおりである．Sheepdog を構成するマシンはすべて同じ構成であり，最大で 124 台のマシンを用いる．これらの物理マシンは 4 つのイーサネットスイッチに対して 31 台ずつ接続しており，スイッチ間は 20 Gbps のネットワークで接続している．仮想マシンはこれらの物理マシン上でなるべく均等になるように配置し，物理マシン間での仮想マシン数の偏りが小さくなるようにする．また，1 つの物理マシンで立ち上げる仮想マシンの最大台数は 4 台とする．SAN ストレージは 6 台の SAS ディスクによる RAID 6 構成であり，2 Gbps のネットワークで Sheepdog のクラスタに接続されている．

表 2　実験環境
Table 2　Experimentation environment.

物理マシン CPU	Core 2 Quad 2.4 GHz
物理マシンメモリ	2 GB
物理マシンネットワーク	GbE
物理マシン OS	Linux 2.6.32 (64 bit)
仮想マシンモニタ	QEMU 0.14
仮想マシンメモリ	256 MB
仮想マシン CPU 数	1
仮想マシン OS	Linux 2.6.32 (64 bit)
SAN ストレージ	NetApp FAS 2020 (iSCSI)
SAN ネットワーク	GbE × 2 (2 Gbps)
ローカルストレージ	SATA 7200 rpm
スイッチ間ネットワーク	HDMI × 2 (20 Gbps)

表 3　分散ロック性能の測定結果
Table 3　Results of distributed lock performance.

方式	ロック取得（回/s）	VDI 作成（個/s）
ZooKeeper	527	3.02
Corosync（4 台）	19,971	3.13
Corosync（8 台）	19,125	3.13
Corosync（16 台）	17,701	3.13
Corosync（32 台）	10,210	3.11
Corosync（64 台）	5,542	2.98
Corosync（124 台）	2,993	2.80

5.1　分散ロック

仮想同期を用いた分散ロックが，集中サーバを用いる分散ロックサービスと比べて，どのくらいの性能が出るのかを測定し，それが Sheepdog の用途に十分な性能であるのかを確認する．集中サーバを用いる分散ロックサービスとして ZooKeeper を利用する．Sheepdog を，仮想同期のマルチキャストの代わりに，ZooKeeper を用いて分散ロックを行うように改良し，Corosync を用いる場合との違いを計測した．

ZooKeeper のマシン台数は 3 台とし，ZooKeeper 内で持つデータはすべてメモリ上で保持されるように設定した．そのため，ZooKeeper も Corosync もローカルディスクへの I/O は発生しない．ZooKeeper も Corosync も死活監視のタイムアウトは 10 秒とした．Sheepdog のデータ冗長度は 3 で固定とし，ZooKeeper + Sheepdog クラスタ（64 台）と，Corosync を用いた Sheepdog のみのクラスタ（台数可変）を比較した．この環境のもと，分散ロックを毎秒何回取得できるか，そしてこの分散ロックを用いて毎秒何個の VDI を作成できるかを計測した．実験結果は**表 3** のとおりである．ZooKeeper は，すべてのロック要求を選出

表 4　シーケンシャル write 性能 (MB/s)
Table 4　Results of sequential write (MB/s).

方式	Buffer size					
	512 B	2 KB	8 KB	32 KB	128 KB	512 KB
direct（冗長度 1）	0.29	0.98	3.32	12.05	30.43	34.92
direct（冗長度 2）	0.17	0.62	2.13	7.31	19.59	21.82
direct（冗長度 3）	0.17	0.59	2.34	8.20	18.22	19.95
primary（冗長度 1）	0.29	1.00	3.56	12.40	31.19	35.52
primary（冗長度 2）	0.16	0.51	2.00	7.01	16.66	17.85
primary（冗長度 3）	0.15	0.50	1.94	6.60	15.01	15.70
ローカルディスク（仮想マシン）	0.63	1.06	19.14	52.68	54.49	50.92
SAN（仮想マシン）	0.51	0.94	14.14	28.96	37.23	37.52
ローカルディスク（物理マシン）	2.60	10.07	34.62	52.98	54.78	47.65
SAN（物理マシン）	1.96	6.28	17.66	32.81	38.29	38.60

した集中サーバに転送し，その後集中サーバがクラスタの残りのマシンへロック情報をコピーしてから要求元にロック成功を知らせるため，ロック 1 回あたりの遅延が大きくなる．また，ZooKeeper では API としてロックを提供しておらず，独自の ZooKeeper API の上でロック処理を実装していることも遅延が大きい理由の 1 つである．これに対し，Corosync ではロック要求はマルチキャストで全ノードに転送されるため，非常に高速である．VDI の作成処理は，ロックとアンロックの間に VDI オブジェクトの作成処理が入るため，実際にはこのロックの速度は隠れて見えなくなる．仮想マシンから仮想ディスクへの I/O は大量に発生しうるのに対し，仮想マシン起動や仮想ディスク作成などの分散ロックをともなう操作は，システム管理者が人の手によって行う操作であるため，現実には秒間に何回も行う操作ではない．そのため表 3 の結果で十分な性能である．また，仮想同期はマシン台数が多くなるとオーバヘッドが大きいという問題があるが，本実験規模の環境では十分動作した．

5.2　複製

Sheepdog で実装している，並列に書き込み処理を送る複製方式（direct 方式）が，既存の方式よりどの程度高速に動作するかを確認するため，そして既存の SAN ストレージなどに対してどのくらいの性能を達成できているのかを確認するために実験を行った．Sheepdog に変更を加えて primary-copy 方式を実装し，Sheepdog の direct 方式と比べてどのくらいオーバヘッドがあるのかを，それぞれの write 性能を計測することで調べる．また，複製を行わないで直接ローカルディスクに書き込みを行うときの性能と，SAN ストレージを用いたときの性能を測定し，Sheepdog との差を調べる．ローカルディスクと SAN ストレージに関しては，仮想マシンからアクセスしたときの性能と，物理マシンから直接書き込みを行ったときの性能の 2 種類を計測する．

まず，ベンチマークツール disktest によって各方式の基本的な性能を計測した．仮想マシンは 1 台，物理マシンの台数は 124 台で固定し，Sheepdog のデータ冗長度を 1 から 3 まで変化させて実験を行った．また，disktest は O_DIRECT によってページキャッシュを使わないモードで計測を行い，各方式の違いが分かるように評価を行った．結果は**表 4** のとおりである．

冗長度 1 のときには direct 方式と primary-copy 方式の差はないが，冗長度 2 以上になると，direct 方式の方が最大で 22〜24% 高速であった．また Sheepdog はバッファサイズが小さいときは，Sheepdog 内部の処理のオーバヘッドの影響が大きいため，ローカルディスクや SAN に比べて性能が悪いが，バッファサイズが大きくなると Sheepdog のオーバヘッドが見えなくなっていき，特に冗長度 1 のときには SAN と近い性能を実現している．

次に現実的な用途における負荷を計測するため，ベンチマークプログラムに，ファイルサーバの負荷を模擬した dbench を用いて実験を行った．disktest と同様に仮想マシンは 1 台，物理マシンの台数は 124 台で固定し，Sheepdog のデータ冗長度を 1 から 3 まで変化させた．そのうえで，仮想マシンで VDI 上に ext3 ファイルシステムを作成したうえで dbench を同期書き込みオプション（-s -S）付きで実行した．実験結果は**表 5** のとおりである．このベンチマークは同期書き込みオプションを付けて実行しているため，遅延の影響が大きい．そのため，冗長度 2 以上の write 時において，direct 方式よりデータ転送回数が 1 ホップ多い primary 方式は，非常に性能が悪くなっている．また，Sheepdog はローカルディスクや SAN に比べて低い性能結果を示している．これはデータ転送回数が 0 ホップであるローカルディスクや，バッテリバックアップのキャッシュに write-back で書き込みを行っていると予想される SAN ストレージは，Sheepdog に比べて遅延が非常に低く，本ベンチマークではその遅延の差が大きく現れたためであると考えられ，この結果は妥当である．

表 5　1 台の仮想マシンから dbench を実行した結果
Table 5　Results of dbench on one VM.

方式	dbench (MB/s)
direct（冗長度 1）	13.19
direct（冗長度 2）	11.76
direct（冗長度 3）	11.41
primary（冗長度 1）	12.81
primary（冗長度 2）	7.98
primary（冗長度 3）	6.97
ローカルディスク（仮想マシン）	22.60
SAN（仮想マシン）	35.44
ローカルディスク（物理マシン）	23.43
SAN（物理マシン）	50.48

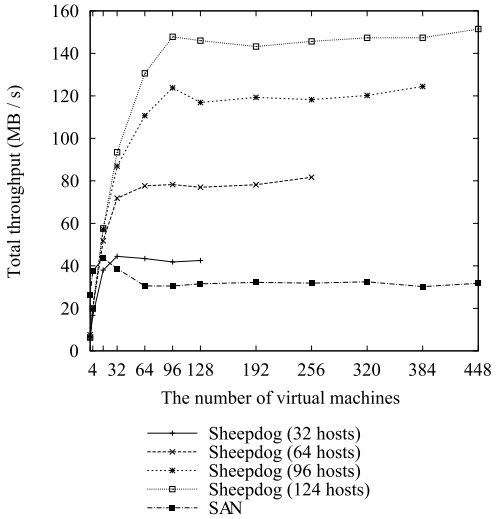

図 5　複数台の仮想マシンから dbench を実行した結果
Fig. 5　Results of dbench on many VMs.

5.3　拡張性

Sheepdog が大規模な仮想環境において SAN ストレージよりも高い性能を示すことを確認するため，大量の仮想マシンから同時にアクセスが起きたときの性能を測定した．Sheepdog のデータ冗長度は 3 で固定し，Sheepdog の物理マシンの台数が 32，64，96，124 台のそれぞれの場合について，Sheepdog に同時にアクセスする仮想マシン数を変化させて実験を行った．また SAN ストレージは 124 台の物理マシンに対して iSCSI で LUN を提供し，Sheepdog と同様に同時にアクセスする仮想マシン数を変化させて実験を行った．仮想マシンの台数は，1 台からホストマシン台数の 4 倍までを最大として変化させた．この最大値の理由は，ストレージの性能を計るために，クライアントがボトルネックとなることを極力避けるためである．そのため，1 台で立ち上げる仮想マシン数をホストマシンのコア数 4 で制限した．そして，現実的な用途に近い負荷として dbench を実行したときの性能を計測した．dbench のオプションには同期書き込みオプション（-s -S）をつけて，ストレージに負荷がかかるようにした．結果は**図 5** のとおりである．縦軸はベンチマーク結果の合計スループットである．

SAN ストレージは仮想マシン数が少ないときには Sheepdog より高い性能を示すが，仮想マシン台数が増えてくると，性能が頭打ちになる．一方，Sheepdog は物理マシン台数に比例して，合計スループットの最大値が向上しており，また，性能が頭打ちになるときの仮想マシンの台数も多くなる．SAN ストレージの性能が頭打ちになっているとき，SAN ストレージが出力する統計情報では，ディスクがビジー状態であった．また，Sheepdog の性能が頭打ちになっているとき，各物理マシンで I/O Wait の割合が高い状態であった．これらから，本ベンチマークではディスクがボトルネックになっていると予想される．SAN ストレージにディスクをどこまで拡張できるかどうかは，SAN ストレージを導入したときのモデルで決まってしまうため，後からディスクの追加が難しいことがありうるが，クラスタストレージはコモディティマシンを追加することで性能を線形に拡張できる．また，大量のディスクを扱える SAN ストレージは非常に高価なものになってしまうが，クラスタストレージは導入コストの観点からも線形に拡張できる．

6.　関連研究

コモディティなハードウェアで動作することを目指した大規模クラスタストレージの研究は古くから多くある．まず Google File System [7] は大きなサイズの追記処理に重点をおき設計された分散システムである．Sheepdog は完全に等質な設計を目指しているのに対し，Google File System はマスタサーバが必要である．また Google File System の複製方式は Sheepdog よりもかなり複雑なものになっている．

Ceph [20] は POSIX を提供する分散ファイルシステムで，メタデータサーバ，モニタサーバ，データサーバの 3 種類のサーバで構成される．これに対し，Sheepdog はブロックデバイスを仮想マシンのみに提供するシンプルな設計であり，また，サーバの種類も 1 つしかなく，運用の容易さに力を入れている．

Ceph のオブジェクトストレージである RADOS [19] は Sheepdog が提供するオブジェクトストレージと似ているが，複製に splay 方式を使っており，Sheepdog よりもオーバヘッドが大きい．また，RADOS にもモニタサーバは必要であり，それに対して Sheepdog は完全に対称的なクラスタ構成で実現される．また，クラスタメンバの管理も Sheepdog が運用性を重視して動的に行っているのに対し，RADOS のメンバ管理は静的で，事前の設定が必要である．

対称型クラスタ構成のストレージの研究として，FAB [16] がある．FAB は書き込み時にはデータ一貫性の保証を行わず，読み込み時に多数決による合意アルゴリズムで正しいデータを読むという設計であるが，Sheepdog はクライ

アントがコーディネータになることで書き込み時に一貫性を保証した書き込みを行う．

サーバ仮想化環境用のストレージとしては，Xen 用のストレージである Parallax [12]，VMware 用のストレージである Ventana [14] 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，それについて取り組んでいるが，Parallax は SAN ストレージと組み合わせることが前提のシステムであり，Ventana は集中サーバが存在する．その他の大規模な仮想化環境を想定したクラスタ型ブロックストレージとして，Lithium [9] がある．Lithium は Sheepdog と同様に対称構成のクラスタストレージを目指しているが，セキュリティに関して特に力を入れており，クラスタの管理などについては触れていない．

仮想同期の実装として Isis [4] や Horus [15] などがある．また，仮想同期で高いスケーラビリティを実現するための研究として，文献 [1], [8] がある．Sheepdog が利用している Corosync も，将来的には文献 [1] を実装してより高いスケーラビリティを得る予定である．

7. おわりに

本稿では，高い運用性を持つ，大規模な仮想化環境用ブロックストレージとして，対称型クラスタ構成のクラスタストレージ Sheepdog について述べた．仮想同期をベースに用いた設計を示し，すべての処理において管理サーバ不要で，動的なクラスタ構成が可能になるシステム設計について説明した．また，提供するものがブロックデバイスならではの割り切りで，大半の処理においてロック処理を不要とし，容易にデータの一貫性を維持しながら高性能を実現した．また，実験により，仮想同期を用いても現実的な用途にはそれほどオーバヘッドにはならないということ，シンプルで高速な複製方式を実現していること，実運用環境で SAN ストレージより高い性能を出すことを示した．

今後の課題は階層化されてスケーラブルな仮想同期を用いたときの実験，そして多拠点などの広域環境において，大規模仮想化環境用のクラスタストレージをどのように構築するかということに関しての検討がある．本稿の実装は http://www.osrg.net/sheepdog/にある．

参考文献

[1] Agarwal, D.A., Moser, L.E., Melliar-Smith, P.M. and Budhia, R.K.: The Totem multiple-ring ordering and topology maintenance protocol, *ACM Trans. Computer Systems*, Vol.16, pp.93–132 (1998).

[2] Alsberg, P.A. and Day, J.D.: A principle for resilient sharing of distributed resources, *Proc. 2nd International Conference on Software Engineering*, pp.562–570, IEEE Computer Society Press (1976).

[3] 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. Computer Systems*, Vol.13, No.4, pp.311–342 (1995).

[4] Birman, K., Schiper, A. and Stephenson, P.: Lightweight causal and atomic group multicast, *ACM Trans. Computer Systems*, Vol.9, pp.272–314 (1991).

[5] Burrows, M.: The Chubby lock service for loosely-coupled distributed systems, *Proc. 7th Symposium on Operating Systems Design and Implementation*, pp.335–350, USENIX Association (2006).

[6] Dake, S., Caulfield, C. and Beekhof, A.: The Corosync Cluster Engine, *Proc. 2008 Linux Symposium*, pp.85–99 (2008).

[7] Ghemawat, S., Gobioff, H. and Leung, S.-T.: The Google file system, *Proc. 19th ACM Symposium on Operating Systems Principles*, pp.29–43, ACM Press (2003).

[8] Guo, K., Vogels, W. and van Renesse, R.: Structured virtual synchrony: exploring the bounds of virtual synchronous group communication, *Proc. 7th ACM SIGOPS European Workshop*, pp.213–217, ACM (1996).

[9] Hansen, J.G. and Jul, E.: Lithium: Virtual machine storage for the cloud, *Proc. 1st ACM Symposium on Cloud Computing*, pp.15–26 (2010).

[10] Hunt, P., Konar, M., Junqueira, F.P. and Reed, B.: ZooKeeper: Wait-free Coordination for Internet-scale Systems, *Proc. 2010 USENIX Conference on USENIX Annual Technical Conference* (2010).

[11] 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, *Proc. 29th Annual ACM Symposium on Theory of Computing*, pp.654–663, ACM (1997).

[12] Meyer, D.T., Aggarwal, G., Cully, B., Lefebvre, G., Feeley, M.J., Hutchinson, N.C. and Warfield, A.: Parallax: Virtual disks for virtual machines, *Proc. 4th ACM European Conference on Computer Systems*, pp.41–54 (2008).

[13] Noll, L.C.: Fowler/Noll/Vo (FNV) hash, available from ⟨http://www.isthe.com/chongo/tech/comp/fnv/⟩.

[14] Pfaff, B., Garfinkel, T. and Rosenblum, M.: Virtualization aware file systems: getting beyond the limitations of virtual disks, *Proc. 3rd Conference on Networked Systems Design and Implementation*, p.26, USENIX Association (2006).

[15] Renesse, R.V., Birman, K.P. and Maffeis, S.: Horus: A Flexible Group Communication System, *Comm. ACM*, Vol.39, No.4 (1996).

[16] Saito, Y., Frølund, S., Veitch, A., Merchant, A. and Spence, S.: FAB: Building distributed enterprise disk arrays from commodity components, *Proc. 11th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp.48–58, ACM (2004).

[17] The Pacemaker Community: Pacemaker, available from ⟨http://www.clusterlabs.org/⟩.

[18] van Renesse, R. and Schneider, F.B.: Chain replication for supporting high throughput and availability, *Proc. 6th Conference on Symposium on Opearting Systems Design and Implementation*, p.7, USENIX Association (2004).

[19] Weil, S., Leung, A., Brandt, S.A. and Maltzahn, C.: RADOS: A Fast, Scalable, and Reliable Storage Service for Petabyte-scale Storage Clusters, *Proc. ACM Petascale Data Storage Workshop 2007* (2007).

[20] Weil, S.A., Brandt, S.A., Miller, E.L., Long, D.D.E. and Maltzahn, C.: Ceph: A scalable, high-performance distributed file system, *Proc. 7th Conference on USENIX Symposium on Operating Systems Design and Implementation*, p.22, USENIX Association (2006).

森田 和孝　（正会員）

2005 年東京大学工学部計数工学科卒業．2007 年同大学大学院情報理工学系研究科数理情報学専攻修士課程修了．同年日本電信電話株式会社入社．以来，分散システム，ストレージシステムに関する研究に従事．

藤田 智成　（正会員）

2000 年早稲田大学大学院理工学研究科修士課程修了．同年日本電信電話株式会社入社．オペレーティングシステム，ストレージシステムに関する研究に従事．ACM 会員．

盛合 敏　（正会員）

1983 年東北大学工学部電気工学科卒業．1988 年同大学大学院博士課程（情報工学専攻）修了．工学博士．同年日本電信電話株式会社入社．入社以来，プロトコル処理，インターネットシステム運用技術，分散 OS，リアルタイム OS（Real-Time Mach），セキュア OS の研究に従事．2001〜2004 年（株）ぷららネットワークス（現 NTT ぷらら）．2004 年より NTT サイバースペース研究所主幹研究員，高信頼 OS カーネル，仮想マシン，大規模分散システム，ユビキタスコンピューティング基盤の研究開発に従事．日本ソフトウエア科学会，電子情報通信学会，USENIX 各会員．The Linux Foundation Japan アドバイザリメンバ．