Vol.2021-HPC-178 No.23
2021/3/16

# DAOSファイルシステムの性能評価

巨畠 和樹[1,a)]　建部 修見[2]

**概要**：大規模なデータを処理するワークロードが年々増加しており, 高性能な分散ファイルシステムへの需要が高まっている. そこでバイト単位でアクセス可能で, 電源がなくてもデータを保持できる不揮発性メモリをストレージシステムへ導入することが検討されている. しかし, 従来のディスクデバイスをストレージとして用いていたストレージシステムへ不揮発性メモリを導入した場合に不要なレイテンシが発生することが懸念されている. したがって不揮発性メモリの特徴を生かした新たなストレージソフトウェアの設計が必要である. DAOS は不揮発性メモリの利用を前提として設計されたオブジェクトストレージである. 本研究では DAOS で構成されるファイルシステムについて実装の特徴を元にその性能評価を行った. 実験の結果, 複数の条件によって性能向上が可能であることを示した. またファイルシステムへのアクセス方法によって性能が大きく変わることが分かった.

## 1. はじめに

不揮発性メモリデバイスの登場により, 不揮発性メモリを活用したファイルシステムの研究 [1] [2] [3] が盛んに行われている. 不揮発性メモリとは, プロセッサがバイト単位で直接アクセスでき, 電力を切ってもデータの保持が可能であるといった特徴を持ったデバイスである. この不揮発性メモリを既存のブロックデバイスの利用を前提としたファイルシステムで用いる場合, ブロックアクセスの最適化手法やそれに基づいたソフトウェアスタックが不必要なレイテンシを生んでしまい, パフォーマンス低下に繋がることが考えられる. そこで不揮発性メモリを用いた新たなファイルシステムの設計が必要となる.

またハイパフォーマンスコンピューティングの分野において, 大規模なデータの管理手法として複数のストレージ専用のノードから構成される並列ファイルシステムがある. 並列ファイルシステムはひとつのストレージデバイスに大規模なデータを保存できない場合や, 複数のストレージデバイスに分散させたデータの管理, さらに保存領域の拡張やデータの保護・複製の管理を行う目的で利用される. 並列ファイルシステムにおいてもブロックデバイスを基に設計されており, 不揮発性メモリの導入とそれに合わせた並列ファイルシステムを構築することで, ストレージシステムが低レイテンシでかつ細かな IO でも高いパフォーマンスを発揮できると考えられる. さらに, 不揮発性メモリの導入によりデータインテンシブなアプリケーションによる IO ボトルネックとなっていた. メタデータ性能の向上を図り, 幅広い IO ワークロードにおけるストレージシステムの性能向上に寄与できることが期待される.

不揮発性メモリを活用したストレージソフトウェアとして Distributed Asynchronous Object Store(DAOS) [4] がある. DAOS は不揮発性メモリを利用するためにゼロから設計されたオープンソースのストレージソフトウェアである. DAOS は従来のブロックデバイスベースのストレージシステムにおいて, DCPM や NVMe SSD という最新の低レイテンシなデバイスを利用する際のソフトウェアスタックによって生じるレイテンシを改善するように設計されている. また, 従来の HCP のワークロードとデータインテンシブなアプリケーションの両方に対応できるようなストレージシステムの機能を提供している.

DAOS をバックエンドとしたファイルシステムも開発されており, DAOS ファイルシステムと呼ばれる. 本研究では, DAOS ファイルシステムについて複数のベンチマークテストを行い, その IO 性能やメタデータ性能を評価することでその IO 特性を明らかにする.

## 2. 関連研究

ハイパフォーマンスコンピューティングの分野で, 大規模データを管理するために広く用いられている並列ファイルシステムとして Lustre [5], [6] が挙げられる. Lustre は管理サーバーとメタデータサーバー, オブジェクトストレージサーバーの三種類のサーバーから構成されている. メタデータサーバーはファイルシステムにおけるメタデー

[1] 筑波大学情報学群情報科学類
[2] 筑波大学計算科学研究センター
[a)] obata@hpcs.cs.tsukuba.ac.jp

タを管理しており, ファイルの読み込みや書き込みを行うときに, ファイルの実体の場所を知ることができる. 大きなサイズのファイルはストライピングされ, オブジェクトストレージターゲットと呼ばれるデータ保存領域に分割して HDD や SSD といったストレージデバイスに保存される. DAOS ではメタデータサーバーを用いない設計を採用している点や利用しているストレージデバイスが異なることから, 性能に大きな影響があると考えられる. 本研究では Lustre のオブジェクトストレージターゲットに当たるターゲット数を変化させてファイルの分散についてのスループットとメタデータ性能評価を行うことで DAOS の性能の傾向を調査する.

DAOS の性能評価については IO500 List [7] に掲載された計測結果や, 不揮発性メモリを搭載したクラスタを構築した上で DAOS の性能評価をした研究 [4], [8], がある. いずれの評価も IO500 のレギュレーションに基づいた計測 [9] を行っている. IO500 では easy と hard のふたつのレギュレーションの下でストレージ性能を比較するため, 特定のストレージについてその特徴や性能の傾向を把握することはできない. 本研究では IO500 とは異なる複数のテストを用意して, スループットやメタデータ性能について調査する.

また, 科学技術計算で用いられるデータフォーマットである HDF5 に対応した DAOS の IO ライブラリについて性能評価した研究 [10] もあるが, 本研究ではファイルシステム実装に焦点を当てて実験・評価を行った.

## 3. DAOS と DAOS ファイルシステムの概要

Distributed Asynchronous Object Storage (DAOS) は, Intel ® DAOS 開発チームにより開発がはじめられた, スケーラブルなオブジェクトストレージであり, オープンソースソフトウェア [11] [12] として開発が進められている.

DAOS では, ストレージクラスメモリ (SCM) や NVM express (NVMe) SSD といったストレージデバイスのみを利用する設計になっており, HDD のような回転ディスクをストレージとして用いない構成が想定されている. SCM としては Intel ® Optane DC Persistent Memory (DCPM) [13] が主に利用される. DCPM は Intel ® 3D XPoint 技術 [14] を用いて開発された不揮発性メモリで, NVMe SSD よりも低レイテンシを実現しており, バイト単位でデータにアクセスすることが可能である点が特徴である. また, 電源を落としてもデータが保持されるという不揮発性メモリの性質も持っている. そのため, DAOS ではメタデータの保存や低レイテンシな IO が要求される場合などで利用されている. 一方で NVMe SSD は粒度の大きなデータを保存する場合に利用されるが, システムの要件として必須のデバイスではない. 本研究では, 評価実験において RAM による DCPM のエミュレーションを SCM として利用する.

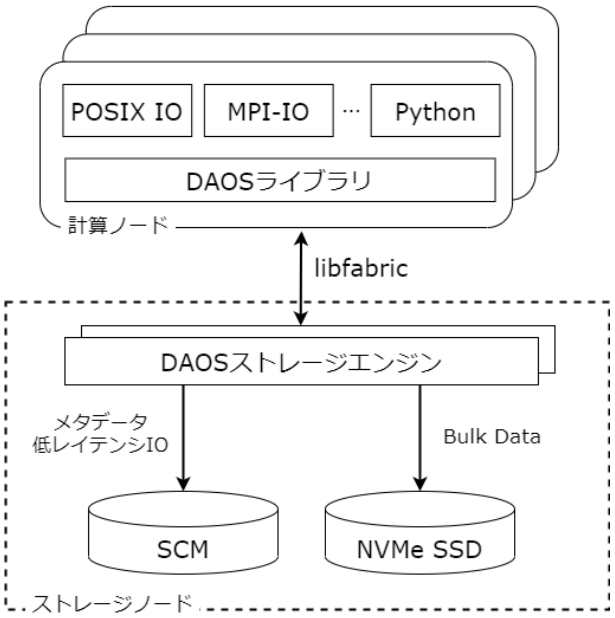


図 1: DAOS のソフトウェアスタック

計算クラスタとストレージクラスタの関係について説明する. 図 1 に DAOS のサーバー・クライアントモデルを示す. アプリケーションは主に計算ノードで行われ, DAOS ライブラリやミドルウェアを通じて IO リクエストを発行する. 発行された IO リクエストは `libfabric` [15] により抽象化された通信レイヤを経由して, DAOS ストレージノードに送られ, 適切に処理される. 通信レイヤとして Omni-Path や InfiniBand といった高速な通信網が主に用いられる.

### 3.1 DAOS オブジェクトの構成

DAOS オブジェクトは DAOS 内で扱われるデータを保存する際に利用されるデータ構造である. 各 DAOS オブジェクトは DAOS コンテナ内で一意なオブジェクト ID で識別される. オブジェクトの保存先のターゲットは, このオブジェクト ID から計算されるハッシュ値をインデックスとして決定される. オブジェクト内部は distribution-key (dkey) と attribution key (akey) と呼ばれる 2 種類のキーを活用した key-value ストアとなっている. dkey はデータをグループ化して, ストレージへの配置のために利用され, akey は格納されるデータを識別するために使われる. ユーザーデータは akey の値として格納される.

DAOS ではターゲット間でデータを分散して管理するためのシャーディング機能を備えている. シャーディングは大きなデータを分割して複数のデータ格納領域に保存することで, スループットを上げる目的で利用される. このシャーディングはオブジェクトに対して行われ, dkey ごとに分割されてターゲットに分散される. シャーディングの数や方法などデータの分散・管理についてのアルゴリズムはコンテナ作成時に決定される.

```
D-key: 0
  A-key: NULL
    < [magic value, record size, chunk size] (
        single value) >
D-key: 1
  A-key: NULL < array elements [0,1,2] >
D-key: 2
  A-key: NULL < array elements [3,4,5] >
D-key: 3
  A-key: NULL < array elements [6,7,8] >
D-key: 4
  A-key: NULL < array elements [9] >
```

図 2: 配列オブジェクトの例

ユーザーデータは single value と array value に区別して管理される．single value は数値や文字列といったデータを扱うことができ，値全体の取得や更新が可能である．array value は同じサイズのデータレコードをインデックスで管理された一連のデータであり，配列のように扱うことができる．つまり，インデックスを指定することで部分的なデータの取得や更新をすることが可能である．

DAOS オブジェクトを扱うクライアント API として，DAOS API や DAOS Array API などがある．DAOS API はオブジェクトの取得や更新，破棄といった基本的な操作を提供する API である．また DAOS Array API は配列オブジェクトと呼ばれる，dkey によって添字づけらた一次元配列を構成するオブジェクトへの操作を提供している．

DAOS 配列オブジェクトの構成について説明する．配列オブジェクトは複数の dkey を保持しており，それぞれの dkey には整数値が格納されている．これは配列オブジェクトのインデックスを表す．また各 dkey はそれぞれ 1 つの akey を保持し，チャンクサイズ分のレコードを持った array value を保持している．このとき akey には NULL が設定される．チャンクサイズは各インデックスに格納されているレコード数のことである．インデックスが 0 の dkey については配列オブジェクトのメタデータとして，各レコードのサイズやチャンクサイズが single value として格納されている．

図 2 は，10 のレコードについてレコードサイズを 1，チャンクサイズを 3 とした場合の例である．インデックスが 0 である dkey にレコードのサイズやチャンクサイズが格納され，それ以外の各 dkey に属する akey にはチャンクサイズである 3 のレコードを持つ配列データが格納されている．

### 3.2 DAOS File System

多くの並列ファイルシステムは POSIX 準拠のインターフェースを提供しており，アプリケーションについても POSIX IO を使用するものが数多く存在する．このような

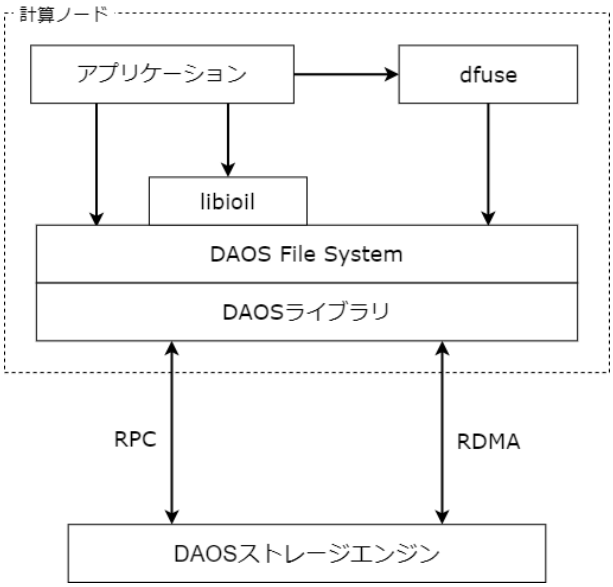


図 3: POSIX ファイルシステムサポートの概要

なアプリケーションをサポートするために，DAOS では DAOS ファイルシステム (DFS) と呼ばれるソフトウェアレイヤが用意されている．DFS は DAOS ライブラリ上に構築されており，`libdfs` ライブラリに実装されている．

POSIX IO により DAOS を利用するための方法は以下の 3 通り存在する．

- Linux FUSE を用いた `dfuse` を利用する
- `dfuse` と IO インターセプトライブラリ (`libioil`) を一緒に利用する
- `libdfs` ライブラリをアプリケーションから直接利用する

これらをまとめたものを図 3 に示す．

`dfuse` は ユーザー空間でファイルシステムの作成を可能にする Linux の FUSE を用いて計算ノード上で DAOS POSIX コンテナのユーザー空間をマウントするための DAOS のミドルウェアである．DAOS POSIX コンテナは DAOS コンテナ内に POSIX 名前空間を内包することのできる機能である．DAOS コンテナとは，DAOS の提供するコンポーネントの一つで，オブジェクトのアドレス空間を提供している．これは `libdfs` によってファイルやディレクトリの抽象化を行うことで実現されている．`dfuse` を利用することで DAOS POSIX コンテナ内の POSIX 名前空間をユーザーに提供することができ，ユーザーはアプリケーションを改変することなく DAOS コンテナを利用できる．`dfuse` を利用する方法は他の 2 つの方法よりも簡単に POSIX IO を利用できる一方，アプリケーションから発行される IO リクエストがカーネルと `dfuse` デーモンを経由する必要があるため，最もレイテンシが大きいと思われる．

IO インターセプトライブラリである `libioil` はアプリ

ケーションの POSIX read, write 操作を監視することで, IO データのカーネルバイパスを実現している. この機能により, dfuse の IO の発行で発生するレイテンシの一部を減らすことができる. メタデータ操作については dfuse の機能を用いる必要があるので, dfuse と連携して利用する必要がある.

libdfs ライブラリをアプリケーションから直接利用する方法では, open や write といった POSIX IO をそれに対応する DFS IO の呼び出しに変更する必要がある. これにより, dfuse やインターセプトライブラリを用いる場合よりも, 良いパフォーマンスを得ることができるが, アプリケーションコードに変更を加えなければならない.

本研究では, IO インターセプトライブラリによる dfuse の性能改善をベンチマークテストにより評価するとともに, libdfs を直接利用する方法との比較も行う.

### 3.3 ファイルとディレクトリの実装

DFS を用いて, DAOS オブジェクトによる POSIX ファイルシステムでのファイルとディレクトリの実装について述べる.

#### 3.3.1 ディレクトリ

ディレクトリは複数の dkey を持った DAOS オブジェクトで表現される. それぞれの dkey はそのディレクトリ内の各エントリに対応する. ディレクトリ名やファイル名は dkey に保存される. エントリはファイルやディレクトリ, シンボリックリンクのことである. 各 dkey は複数の akey を持っており, エントリの属性やデータが格納される. エントリ名は dkey として保持され, 親のオブジェクトが子のエントリのオブジェクト ID を管理している.

この例では, dir1 がディレクトリ名であり, akey には inode 番号が格納される.

#### 3.3.2 ファイル

ファイルは DAOS 配列オブジェクトとして表現される. 要素サイズは 配列オブジェクトに保存される各データのサイズであり, DFS におけるファイルの実装では 1 バイトである. チャンクサイズは各 dkey に含まれる要素の総サイズであり, ファイル作成時のみに設定することができる. デフォルトのチャンクサイズは 1MiB である. したがってファイルはチャンクサイズごとに分割された複数の array value から成る配列オブジェクトとして表現される.

## 4. 評価

### 4.1 実験環境

評価のための実験には研究室のクラスタを使用した. 表 1 に DAOS サーバーノードの構成を示す. また, 評価にあたって使用したソフトウェアのバージョンを表 2 に示す. 本研究では 1 台の DAOS サーバーと 1 から 2 台のクライアントノードを準備して実験を行った. なお, SCM は RAM によるエミュレート環境を使用した.

### 4.2 ベンチマークツール

ベンチマークツールには IOR と MDTest を用いる. IOR は並列ストレージシステムの IO 性能の計測のためのベンチマークである. また, MDTest はファイルシステムにおけるメタデータ操作の性能計測のためのベンチマークである. 同じような IO 性能の計測ツールである fio や LINUX の dd コマンドでは, DFS API を直接利用する場合の計測において追加の実装が必要となる. しかし, IOR や MDTest では既に DFS API を利用したベンチマークテストの実装が組み込まれているため, これらのツールを採用した.

### 4.3 実験 1. 発行する IO のデータサイズ

発行するデータサイズごとの読み込み・書き込み性能を調査することで, アプリケーションの最適化を施す指標を与えることができる. IOR ではトランスファーサイズを変化させることで, ストレージへ発行される各 IO サイズを制御してバンド幅を計測することができる. また, データサイズに関わる項目としてファイルのチャンクサイズがある. チャンクサイズはファイルを DAOS 配列オブジェクトに変換する際の分割に影響を及ぼすことから, IO 性能にも影響を与えると考えられる. したがって, 実験 1 としてトランスファーサイズを変化させたときの DAOS ファイルシステムの読み込み及び書き込み性能を計測し, 同時にチャンクサイズによる影響も調査する.

さらにファイルに対するアクセスパターンについても IO サイズごとに調査を行う. 具体的には, N-1 パターンと N-N パターンの二種類についてバンド幅の計測を行う. これら二種類のアクセスパターンは IOR の File-per-process オプション (N-N) と Single-shared-file オプション (N-1) を用いて計測する. File-per-process はクライアントプロセスごとにファイルを作成して読み書きを行うようなワークロードを実行する. Single-shared-file は各クライアントプロセスが同一のファイルに対して読み書きを行うようなワークロードを実行する. またこの実験では DAOS のシャーディング機能を有効にした場合との比較も行う.

#### 4.3.1 トランスファーサイズを変化させた場合のバンド幅

サイズを 16K, 32K, 64K,..., 4MiB と変化させ, プロセスあたり 2GiB のデータを読み込み・書き込みを行うことで, 読み書きの性能への影響を調査する. クライアントプロセス数は 4 とし, 全体で 8GiB のデータの読み書きを行った. DAOS へのアクセス方法には, IOR にて実装されている DAOS ドライバの DFS を使用した. また, DAOS サーバーのターゲット数は 4 とし, チャンクサイズについてはデフォルトの 1MiB に設定してそれぞれについて計測を行った.

以上の条件下で実験を行った結果を示す. 図 4 は読み

表 1: 実験に使用した計算ノードの構成

| | |
|---|---|
| CPU | Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz × 1 |
| メモリ | 96GiB (DDR4-3200 16GiB × 6) |
| ネットワーク | InfiniBand HDR100 × 1 |
| OS | Ubuntu 20.04.1 LTS (Focal Fossa) |

表 2: 実験に使用したソフトウェアのバージョン

| | |
|---|---|
| daos | v1.1.1 |
| IOR,MDTest | 3.4.0+dev (commit:cb397242f9b896acd3429d3125303a340112b629) |
| OpenMPI | 4.0.5 |

込み・書き込みにおけるトランスファーサイズを変化させたときの性能である．また図 5 はシャーディング機能を有効にして, 4 つのターゲット全体でシャーディングを行うときの読み込み・書き込みの性能である．N-1, N-N はそれぞれ single-shared-file と file-per-process としてファイルの読み書きを行ったことを表している．結果はそれぞれのトランスファーサイズにつき 3 回計測した平均値である．それぞれの図の縦軸はバンド幅を表し, 横軸はトランスファーサイズを表す．

トランスファーサイズを 16KiB から 1MiB まで変化させるとバンド幅も大きくなっていき, 読み込みでは N-1,N-N の両方で約 11,135MiB/eec, 書き込みでは N-1 で約 9600MiB/sec, N-N では約 10,700MiB/sec という結果となった．トランスファーサイズ 1MiB の場合がピークとなり, それ以上のトランスファーサイズの場合はバンド幅の変化は読み込み・書き込みの両方でその差があまり見られなくなった．また読み込みでトランスファーサイズが 512KiB と 1MiB の変化が小さくなっているように見れるが, Infiniband によるネットワークのバンド幅が約 10GiB であることから, 本実験の設定ではネットワーク帯域がボトルネックとなったためであると考えられる．

読み込み・書き込みの両方で N-N のほうが N-1 よりも高い性能を発揮していることが分かる．N-N, N-1 間の関係についてはトランスファーサイズが 256KiB の場合を例に取ると, N-1 に対して N-N は読み込みで約 1.26 倍, 書き込みで約 1.45 倍のバンド幅であった．DAOS ファイルシステムではファイルあたりにオブジェクトが作成されおり, 分散されたオブジェクトごとに読み書きが行われる．N-N ではそれぞれのファイルでオブジェクトが作成され, それらがターゲットに分散されたことによりオブジェクトへの読み書きも分散され, 性能が N-1 よりも高かったと考えられる．

シャーディングを行った場合は読み込み・書き込みのそれぞれで, アクセスパターン間の差は殆ど見られなかった．シャーディングを無効にした場合は, オブジェクト ID から計算されるハッシュによってオブジェクトが単一のターゲットに分散される．一方でシャーディングを有効にした場合は, 各オブジェクトが複数のターゲットで分散される．具体的には, 各オブジェクト内の dkey が保持するファイルのデータは dkey のハッシュ値から決定されるターゲットに分散して保存される．DAOS ファイルシステムのファイルの実装を踏まえると, N-1 と N-N のどちらのアクセスパターンも作成されるオブジェクト数は異なるものの, dkey を元にターゲットに対して分散して保存されている．また, DAOS では各ターゲットごとにターゲットを管理するプロセスが起動され, ターゲットにデータを保存するためのプロセスが各ターゲットで独立にデータを処理できる．よって N-1 の場合でも N-N と同様に, チャンクサイズごとにターゲットで分散されたファイルの読み書きを行っていたと考えられる．したがって, シャーディングを有効にした場合に N-1 と N-N の性能差が見られなかったと考えられる．

#### 4.3.2 チャンクサイズを変えた場合のバンド幅

また, 図 6 はそれぞれチャンクサイズに 256K, 1MiB を指定したときの読み込み・書き込みバンド幅についてトランスファーサイズサイズを変化させて計測を行った結果である．アクセスパターンは N-N とした．結果はそれぞれのトランスファーサイズにつき 3 回計測した平均値である．それぞれの図の縦軸はバンド幅を表し, 横軸はトランスファーサイズを表す．

図 6 から, 読み込みについてはチャンクサイズを 256KiB にしたときと 1MiB にしたときのバンド幅について, それぞれトランスファーサイズが 256KiB と 1MiB の場合にバンド幅がピークとなり, それ以上のトランスファーサイズではバンド幅の変化がないことが分かる．また書き込みについてチャンクサイズが 256KiB の場合に, トランスファーサイズを 16KiB から 256KiB まで変化させるとチャンクサイズが 1MiB である場合よりもバンド幅が低い結果となった．

また, それぞれのチャンクサイズ以上のトランスファーサイズで読み込みや書き込みを行った場合のバンド幅には殆ど変化は見られなかった．ファイルはチャンクサイズで分割され, 各 dkey に関連した akey のデータが読み込み・書き込みの単位となるのでそれぞれの dkey の処理がチャンクサイズよりも大きなデータの読み込み・書き込みではターゲットごとに処理が逐次実行されてしまうためである

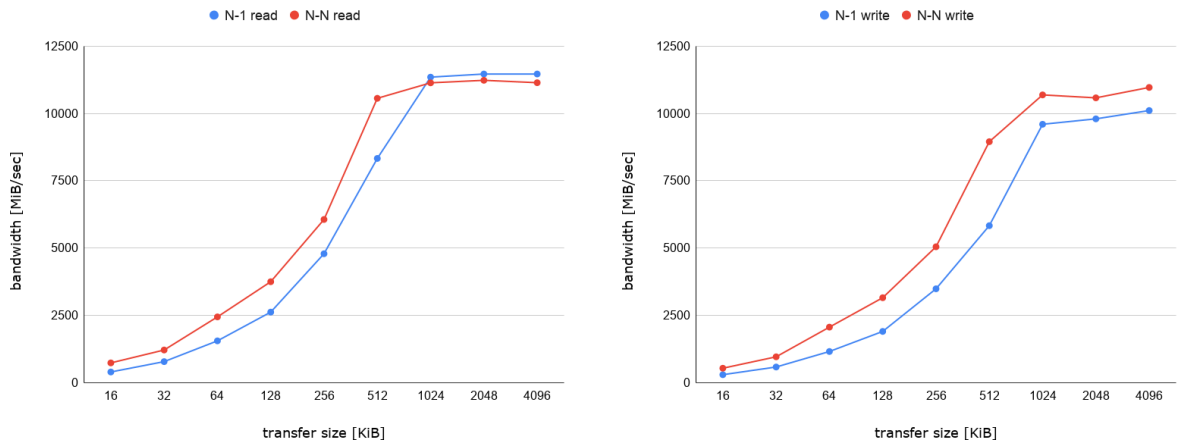


図 4: トランスファーサイズを変化させたときのバンド幅
(左) 読み込み (右) 書き込み

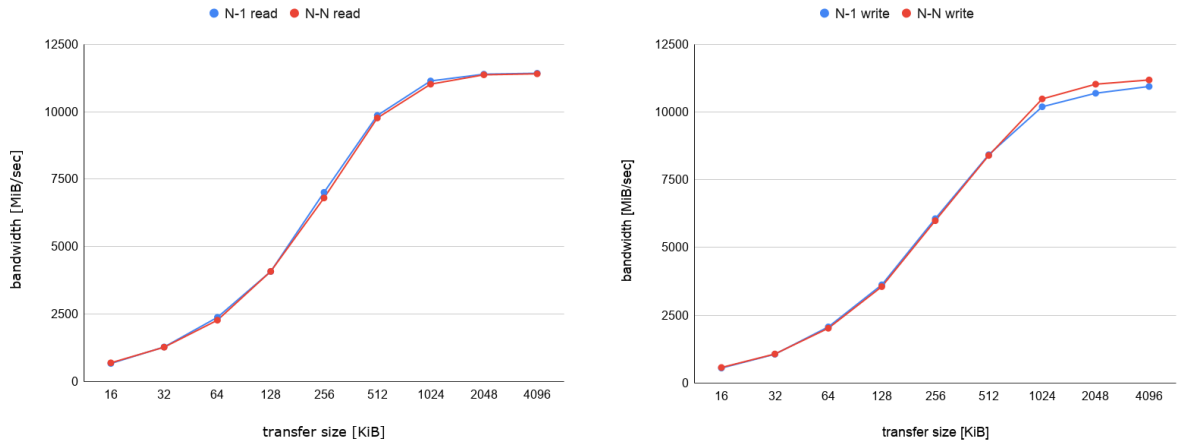


図 5: トランスファーサイズを変化させたときの読み込み性能 (シャーディングあり)
(左) 読み込み (右) 書き込み

と考えられる.

### 4.4 実験 2. データ格納領域ターゲット数

複数のターゲットを用意することによって, IO の並列性を高め, 同時に読み込みや書き込みを行うことが可能となる. ターゲット数による IO 処理の性能の変化を見ることで DAOS ファイルシステムの IO パフォーマンスの傾向を明らかにすることができる.

DAOS のストレージクラスタはターゲット数をそれぞれ指定して構築した上でバンド幅やメタデータ性能の計測を行った. クライアントプロセス数は 32 とし, 2 ノードからそれぞれ 16 プロセスずつ実行し, 合計で 8GiB の読み込み・書き込みを行った. チャンクサイズはデフォルトの 1MiB に設定した. またトランスファーサイズは一例として 64KiB とした.

図 7 に, ターゲット数を 1, 2, 4, 8, 16 と変化させたときのバンド幅とメタデータ性能の計測結果を示す. 計測には N-N のアクセスパターンを用いた. 図 7 の縦軸はバンド幅, 横軸はターゲット数を表しており, 読み込みと書き込みのそれぞれの結果をグラフで示している. 結果はそれぞれのターゲット数につき 3 回計測した平均値を用いた.

バンド幅について, 書き込みと読み込みの両方でターゲット数を増やすとバンド幅も上がっていることが分かる. ターゲット数が 8 以降ではバンド幅が約 6000MiB/sec となった. DAOS ではターゲットごとにサーバープロセスが立ち上げられ, それぞれがクライアントからの IO の処理を行う. 各オブジェクトはオブジェクト ID により計算されるハッシュで分散されるため, ターゲットを増やすことで複数のターゲットにオブジェクトに対する IO を分散できると考えられる. よって, ターゲット数は十分に大きく取る必要があると言える.

### 4.5 実験 3. ディレクトリ内のメタデータ操作

ファイル・ディレクトリ操作における DAOS ファイルシ

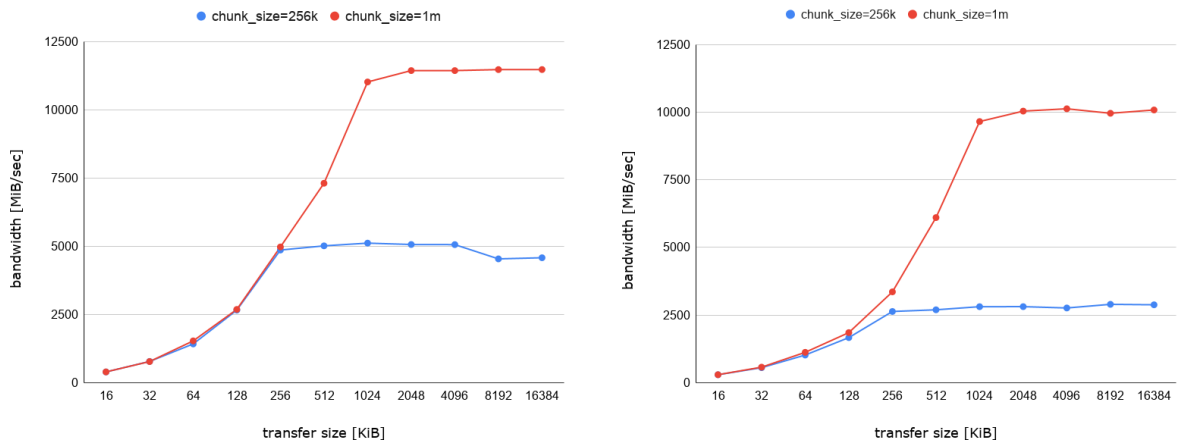


図 6: チャンクサイズを変化させたときの読み込みバンド幅の比較
(左) 読み込み (右) 書き込み

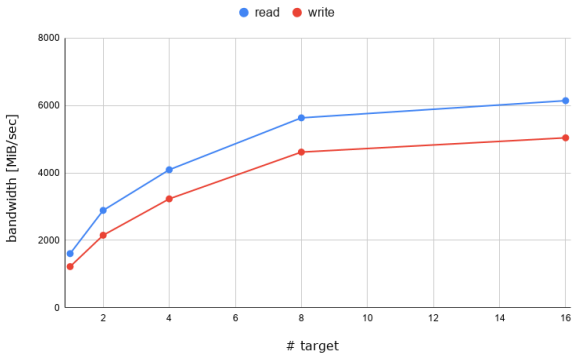


図 7: ターゲット数を変化させたときのバンド幅

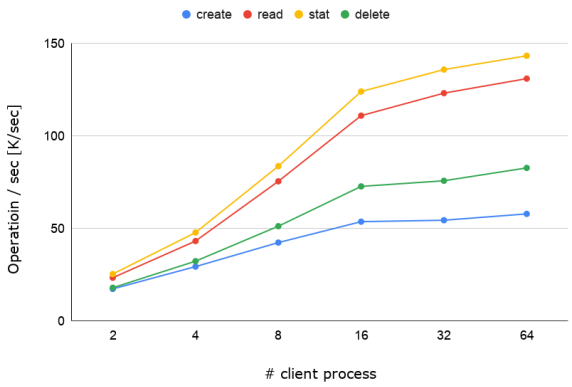


図 8: ファイルのメタデータ性能

ステムの性能の傾向を調べるために, MDTest によるメタデータ性能の計測を行う. 単一のディレクトリを用意してファイルの作成 (`create`)・取得 (`stat`)・読み取り (`read`)・削除 (`delete`) を行い, その時間を計測することで, 秒間の操作回数を評価する.

`create`, `stat`, `read`, `delete` のそれぞれについてクライアントプロセス数を増やした際の性能を計測する. プロセス数は 2, 4, 8, 16, 32, 64 とし, 32, 64 プロセスで実験する時はクライアントノードを 2 ノード用意してノードあたりのプロセス数はそれぞれ 16, 32 とした. また全体で 1,000,000 のファイルを作成し, それぞれの操作を行った. ターゲット数は一例として 4 に設定し, チャンクサイズはデフォルトの 1MiB とした.

図 8 はファイル及びディレクトリに対してそれぞれのメタデータ操作を行ったときの性能を示している. 縦軸は OPS(Operation/second) を表しており, 横軸は実行したクライアントプロセス数を表している. 結果はそれぞれのターゲット数につき 3 回計測した平均値を用いた.

ファイルの `stat` については 2 プロセスで 25.4Kops, 4 プロセスで 47.8Kops とクライアントプロセスの増加に伴い, 時間あたりの `stat` 回数も大きくなっていき, 64 プロセスでは 130.9Kops となった. MDTest における `read` は主にファイルの `open` や `close` を主に行うワークロードであるが, `stat` と同様の傾向が見られ, 64 プロセスでは 143.2Kops に到達した. また, `create` と `delete` でもプロセス数の増加に伴って OPS も増加している傾向が見られ, 64 プロセスで `create` は 57.8Kops, `delete` は 82.7Kops であった.

それぞれの操作を特徴と共に評価する.

`delete` はオブジェクト及びオブジェクトの dkey のパンチ操作である. パンチ操作では実際にオブジェクトなどの削除が行われるのではなく, ゼロ値の書き込みやパンチされたログを追加することで, オブジェクトが削除されたことにする. パンチされたオブジェクトや dkey の削除はバックグラウンドで実行される. したがって, `create` のようにオブジェクトの作成や作成されたエントリの登録を行う操作よりも比較的軽量な操作であったため, `create` よりも OSP が高かったと考えられる.

IOR や MDTest における `stat` 操作では, `create` で作

成されたハッシュ表を用いている．このハッシュ表はファイルシステムのエントリの絶対パスを用いて対応するオブジェクト ID を取得するためのキャッシュとしての役割を担っている．これを用いることでディレクトリのルックアップを減らし, 検索の性能を向上されることができると考えられる．本稿ではこのハッシュ表を利用した `stat` の評価を行ったが, ルックアップのみの評価を行う必要があると考えられる．

### 4.6 実験 4. DAOS へのアクセス方法

第 3.2 節にて, DAOS を POSIX ファイルシステムとして使用する際のアクセス方法について紹介した．それぞれの手法について, ユーザーのアプリケーションから発行される IO が経由するソフトウェアスタックが異なることから, 実際のアプリケーションにおけるパフォーマンスも異なると考えられる．これらの手法のパフォーマンスを比較することで, アプリケーションの規模や利用状況によって, どの手法を用いるかの判断の指標となる．本研究では DFS と `dfuse` を用いた POSIX アクセス, `dfuse` とインターセプトライブラリを用いた POSIX アクセスについて性能評価を行い, これらを比較する．

DAOS ファイルシステムを利用する 3 通りの方法について, 読み込み・書き込み性能およびメタデータ性能を調査する．DAOS のストレージクラスタは, ターゲット数を 1 と指定して構築した上でバンド幅やメタデータ性能の計測を行った．クライアントプロセス数は 4 とし, チャンクサイズはデフォルトの 1MiB に設定した．それぞれのプロセスで 5GiB のデータの読み書きを行った．

図 9 はバンド幅について, アクセス方法それぞれについて実験を行った結果である．それぞれの図の縦軸はバンド幅を表し, 横軸はトランスファーサイズを表す．結果はそれぞれのトランスファーサイズにつき 3 回計測した平均値を用いた．

読み込み及び書き込みの両方の性能について共通しているのは, DFS API を直接利用する方法がどのトランスファーサイズについても高い性能を発揮しているという点である．`dfuse` や `dfuse` とインターセプトライブラリを用いる場合では, DFS API を発行してサーバーで実行されるまでに様々なソフトウェアスタックを通過しなければならないことから, DFS API を直接利用する場合と比較して性能が低くなっていると考えられる．

読み込み性能に関して, サイズが大きくなるにつれて DFS API を直接利用する場合と `dfuse` 経由で POSIX API を発行する場合とのバンド幅の差が大きくなっていることも分かる．また, `dfuse` を利用した場合と `dfuse` とインターセプトライブラリを利用した場合についてトランスファーサイズが 256KiB より大きくなると DFS API を直接利用する場合の半分ほどのバンド幅となっている．インターセプトライブラリを利用した方法について, 読み込み性能に限っては `dfuse` を用いる場合と比較してトランスファーサイズ間での差はあまり見られなかった．

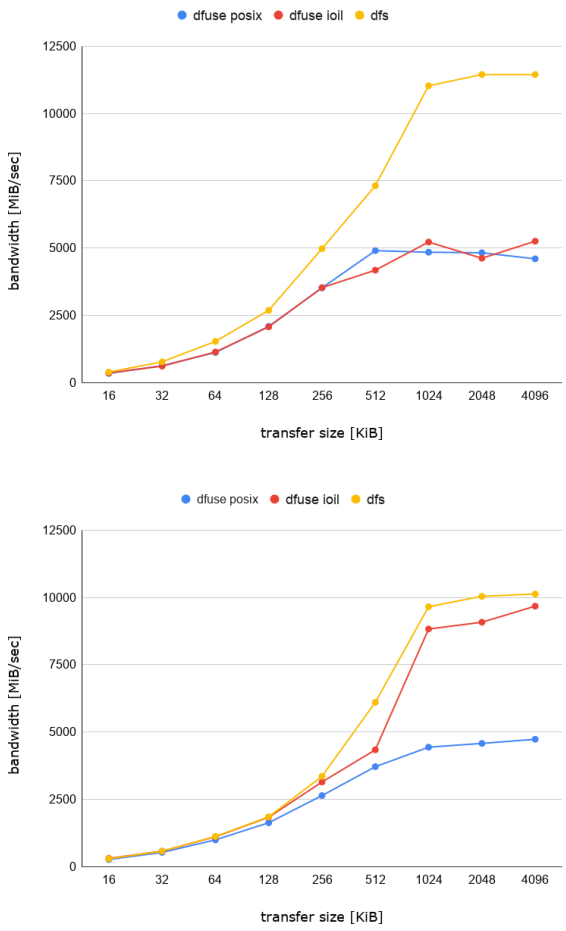


図 9: DAOS ファイルシステムの各利用方法におけるバンド幅
(左) 読み込み (右) 書き込み

書き込み性能に関して, トランスファーサイズが小さい場合はアクセス方法との間にバンド幅の差は殆どない．トランスファーサイズが大きくなると DFS API を直接利用する場合と `dfuse` を利用する場合の差が大きくなっていき, `dfuse` を用いる場合は最大で DFS API を直接利用する場合の半分のバンド幅となった．また, インターセプトライブラリを用いる場合は, トランスファーサイズが大きくなると DFS のバンド幅に近い結果となり, POSIX API を経由することによるレイテンシの一部が排除できていると考えられる．特にトランスファーサイズがチャンクサイズ以上である場合について DFS API を直接利用した場合の結果に近い書き込み性能を発揮できることが分かった．

## 5. まとめ

本研究では DAOS から構成される DAOS ファイルシステムについて, そのアクセス方法や IO パターンに着目して

性能評価を行った. DAOS ファイルシステムの評価で以下のことが明らかになった.

- IO サイズがチャンクサイズになるとスループットが一定となることが分かった.
- シャーディングなしの場合, N-N のアクセスパターンのほうが N-1 よりも良いスループットとなり, シャーディングを有効にすることで両アクセスパターンで同程度の性能が発揮できることが分かった.
- ターゲット数を十分に取ることで, スループットを向上させることができることを確認した. 実験ではターゲット数を増加させることで 3 倍以上のスループットを得ることができた.
- 各メタデータ操作についてプロセス数を増やすことで性能がスケールすることを確認した.
- DAOS ファイルシステムに対するアクセス方法について, DFS API を直接用いる方法と `dfuse` を使って POSIX アクセスを可能にする方法, インターセプトライブラリと `dfuse` を用いて部分的に POSIX IO をバイパスさせる方法という三種類の方法について評価を行った. DFS API を直接利用する方法が最もスループットが高く, インターセプトライブラリについては書き込み性能において, `dfuse` よりも高い性能を発揮することを確認した.

**謝辞** 本研究の一部は, 筑波大学計算科学研究センターの学際共同利用プログラム（Cygnus）, 国立研究開発法人新エネルギー・産業技術総合開発機構（NEDO）および富士通研究所との共同研究の助成を受けたものです.

## 参考文献

[1] Xu, J., Zhang, L., Memaripour, A., Gangadharaiah, A., Borase, A., Da Silva, T. B., Swanson, S. and Rudoff, A.: NOVA-Fortis: A fault-tolerant non-volatile main memory file system, *Proceedings of the 26th Symposium on Operating Systems Principles*, pp. 478–496 (2017).

[2] Yang, J., Izraelevitz, J. and Swanson, S.: Orion: A distributed file system for non-volatile main memory and RDMA-capable networks, *17th {USENIX} Conference on File and Storage Technologies ({FAST} 19)*, pp. 221–234 (2019).

[3] Lu, Y., Shu, J., Chen, Y. and Li, T.: Octopus: an RDMA-enabled Distributed Persistent Memory File System, *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, Santa Clara, CA, USENIX Association, pp. 773–785 (online), available from ⟨https://www.usenix.org/conference/atc17/technical-sessions/presentation/lu⟩ (2017).

[4] Liang, Z., Lombardi, J., Chaarawi, M. and Hennecke, M.: DAOS: A Scale-Out High Performance Storage Stack for Storage Class Memory, *Asian Conference on Supercomputing Frontiers*, Springer, pp. 40–54 (2020).

[5] Lustre Working Group: Lustre: A Scalable, High-Performance File System, Lustre Working Group (online), available from ⟨`https://www.lustre.org/`⟩ (accessed 2021-01-22).

[6] Lustre Working Group: Lustre Wiki Links Main Page, Lustre Working Group (online), available from ⟨`https://wiki.lustre.org/Main_Page`⟩ (accessed 2021-01-22).

[7] Virtual Institute for I/O: IO500 Full list 2020-11, Virtual Institute for I/O (online), available from ⟨`https://www.vi4io.org/io500/list/20-11/start`⟩ (accessed 2021-01-22).

[8] Hennecke, M.: Designing DAOS Storage Solutions with Lenovo ThinkSystem SR630 Servers, Technical report (2020).

[9] Virtual Institute for I/O: IO500 Submission Rules, Virtual Institute for I/O (online), available from ⟨`https://www.vi4io.org/io500/rules/submission`⟩ (accessed 2021-01-22).

[10] Breitenfeld, M., Fortner, N., Henderson, J., Soumagne, J., Chaarawi, M., Lombardi, J. and Koziol, Q.: DAOS for Extreme-scale Systems in Scientific Applications, *ArXiv*, Vol. abs/1712.00423 (2017).

[11] daos-stack: DAOS Github reposity, daos-stack (online), available from ⟨`https://github.com/daos-stack/daos`⟩ (accessed 2021-01-22).

[12] daos-stack: DAOS documentation, daos-stack (online), available from ⟨`https://daos-stack.github.io/`⟩ (accessed 2021-01-22).

[13] Intel Corporation: Optane DC Persistent memory Brief, Intel Corporation (online), available from ⟨`https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/optane-dc-persistent-memory-brief.pdf`⟩ (accessed 2021-01-22).

[14] Margie.venes, Felix.legros: Technology Trends 3D Xpoint, GCwiki (online), available from ⟨`https://wiki.gccollab.ca/images/2/2e/EN_-_Technology_Trends_-_3D_Xpoint.pdf`⟩ (accessed 2021-01-22).

[15] OpenFabrics Interfaces Working Group: OpenFabrics Interface - Libfabric, OFI Working Group (online), available from ⟨`https://ofiwg.github.io/libfabric/`⟩ (accessed 2021-01-22).