Vol.2020-GN-109 No.19
Vol.2020-CDS-27 No.19
Vol.2020-DCC-24 No.19
2020/1/24

# オンサイト利用を想定したIoTデータのための複数NAS統合型ファイルシステムの提案

岡本 祐樹[†1,a)]　荒井 研一[†2]　小林 透[†2]　藤橋 卓也[†1]　渡辺 尚[†1]　猿渡 俊介[†1,b)]

**概要**：Arduino，Raspberry Pi などのコンピューティングデバイス，BLE，LPWA などの通信技術の登場によって IoT が爆発的に広まってきている．IoT データを長時間取得する際、日々増加し続ける大量のセンサーデータを保存するためのストレージが問題となる。本稿では、IoT が駆動する現場で利用するための複数 Network Attached Storage（NAS）統合型ファイルシステム「Sensor Data File System（SDFS）」を提案する。SDFS はユーザー空間にファイルシステムを実装するためのインターフェイスである FUSE を使用して開発した。SDFS は複数の NAS が 1 つのファイルシステムとして扱えるだけでなく、容易に新しい NAS を追加できるなどの特徴を兼ね備えている。SDFS を aufs や unionfs-fuse などの既存のファイルシステムと比較したところ、同等の読み込み・書き込みスループットを達成することを確認した。

**キーワード**：IoT，NAS, 仮想ファイルシステム, union mount

## 1. はじめに

Arduino，Raspberry Pi などのコンピューティングデバイス，BLE，LPWA などの通信技術の登場によって IoT が爆発的に広まってきている．IoT の応用は工場，農業，建築，土木，エネルギー，物流，商業，教育など幅広く，さまざまな分野での導入が進められている．これら多種多様な応用の中で IoT のシステムが成功するかどうかは，現場の人たちがどれだけ自分の手を使って IoT サービスの構築と運用ができるかにかかっている．例えば愛知県碧南市の旭鉄工では，秋葉原で購入してきた部品を組み合わせて自前で工場のライン監視システムを構築して設備投資で 4 億円，労務管理費で 1 億円の削減に成功している [1].

IoT でのデータ取得が長期間続くと，日々増加し続ける大量のセンサデータを保管するためのストレージをどうするかが問題となってくる．大量のセンサデータを扱うことを考えると，低コストで拡張性が高いストレージシステムが望ましい．既存のストレージシステムとしては，クラウドで提供されているサービスを利用することや，大容量のストレージサーバを用意することが考えられる．しかしながら，クラウドサービスを利用した場合にはアップロードのための通信回線や月々の利用料などの固定費の存在が，大容量のストレージサーバを導入する場合には高価な初期導入コストなどの問題がある．現場駆動の IoT をシステムは小規模な実証から徐々に拡張していくため，上記のような既存のストレージシステムを利用するのには不向きである．既存のストレージシステムの問題点に関しては 2 節で詳細に議論する．

このような観点から，本稿ではオンサイトで低コストで拡張性が高い複数 Network Attached Storage (NAS) 統合型ファイルシステム「Sensor Data File System (SDFS)」を提案する．SDFS は日々増加し続けるセンサデータを効率的に蓄積・管理するファイルシステムである．SDFS の最大の特徴は「複数の NAS が 1 つのファイルシステムとして見える」ことである．その他の特徴としては，「拡張・管理が容易」であることや「データが時系列的に保存される」ことが挙げられる．SDFS を aufs や unionfs-fuse などの既存の複数ストレージを重ね合わせることのできる手法と実機比較を行った結果，定量的な性能に関しては既存手法と同等のスループットを達成していることが分かった．

本稿の構成は以下の通りである．２節ではセンサデータを蓄積する際の課題について整理する．３節で提案手法である SDFS の全体像と実装について述べ，４節でその SDFS の評価をまとめている．５節では関連研究について触れ，６節でまとめとする．

†1 現在，大阪大学大学院情報科学研究科
Presently with Information Science and Technology
†2 現在，長崎大学大学院工学研究科
Presently with Nagasaki University
a) okamoto.yuki@ist.osaka-u.ac.jp
b) saru@ist.osaka-u.ac.jp

ⓒ 2020 Information Processing Society of Japan

## 2. センサデータを蓄積する際の課題

ハードウェアの発達や新たな通信規格の登場によってセンサを設置してデータを集め始めること自体の敷居は下がって来ている．例えば，筆者らは，軍艦島において崩壊中の建築構造物の映像や加速度のデータを収集することで建築構造解析に貢献することを目指した軍艦島モニタリングプロジェクトを進めている [2–4,4–25]．2016 年よりセンサデータの収集を開始し，2019 年 11 月現在では約 8 TB のセンサデータを収集している．

センサでのデータの取得が長期に渡ってくると，日々増加し続ける大量のセンサデータをどのように保存するかが問題になる．具体的には，
(1) 安くて簡単に拡張・管理できること
(2) 時系列的に保存されること
(3) 既存のソフトウェアがそのまま利用できること
の 3 つの要件を満たしたストレージが求められる．

1 つ目の「安くて簡単に拡張・管理できること」はセンサデータの価値の観点から重要である．センサデータは，センシングを行うシステムの運用を開始した時から増え続ける．また，データ解析技術が後から進歩することもあるため，可能な限り過去のデータは全て蓄積しておくことが望ましい．例えば，軍艦島モニタリングシステムは，先ほども述べたように 2019 年 11 月現在で 8 TB のデータを蓄積しており，1 日あたり約 10 GB ずつ増え続けている．現在の軍艦島モニタリングシステムでは，複数の NAS にセンサデータを保存し，NAS の容量が足りなくなった際に手動で NAS を切り替えながら日々増え続けるセンサデータに対応している．また，今年度より過去に取得した画像データを用いて軍艦島の崩壊場所を抽出する研究も始めるなど，蓄積されているセンサデータの種類や量が増えれば増えるほど新たに「〜のような解析をしたい」という要求も生まれ続けている．

2 つ目の「時系列的に保存されること」は，センサデータの時系列性の観点から重要である．センサデータは時刻と紐づいて初めて意味を持つ．センサデータを解析する際にも，「いついつからいついつまでのデータが欲しい」のように，時刻に紐づいた形でセンサデータを利用することが多い．膨大なデータの中からある期間のセンサデータを取得する際に，複数のストレージに別々に保存されているなどストレージの存在を意識することはデータの利活用の観点から非効率的となる．理想的には，1 つのストレージ内において時系列的に保存されることが望ましい．

3 つ目は，「既存のファイル管理用のソフトウェアがそのまま利用できること」である．軍艦島モニタリングシステムではデータ管理に ls，cp，mv，rm，rsync などの標準的なファイル管理ソフトウェアを利用している．このような既存のソフトウェアをそのまま利用できるストレージが必要となる．

**表 1** クラウドストレージの月額使用料の例

| クラウドサービス | 月額使用料 [$] |
|---|---|
| Microsoft Azure Files | 480.0 |
| Dropbox | 285.05 |
| Amazon S3 | 200.0 |
| IBM Cloud Object Storage (Cold Vault) | 72.0 |

センサデータを蓄積する手段として，クラウドで提供されているストレージサービスを利用することが考えられる．例えば，Amazon S3 などのクラウドストレージは取得したセンサデータの複数拠点での利用に有利である．しかしながら，大量のデータを保存するためには費用が高額になる．表 1 に主要なクラウドストレージの月額使用料を示す．8 TB のストレージを利用して，1 ヶ月あたり 300 GB のデータをアップロードする場合の料金の例である．実際に利用する場合はダウンロード等の転送料金が追加されるため使用料は増加する．また，データは増え続ける一方であり，ストレージ利用料は固定費として支払い続けなければならない．「センサデータを利活用して新しいサービスを創出する」という観点に立った場合，大企業でなければ運用し続けることが難しいようなシステムは避ける必要がある．

センサデータを蓄積する手段として，最初から大容量なストレージサーバを導入する事が考えられる．しかしながら，ストレージサーバは一般的に高価なものが多い．また，扱えるデータ量の上限が決まっているなど，増え続けるセンサデータに対しては不向きな要素が多い．

## 3. 提案手法：Sensor Data File System

### 3.1 SDFS の全体像

2 節の議論を踏まえて，安価な複数の NAS を統合して 1 つのストレージとして見えるような仮想ファイルシステムを構築することを考える．図 1 に複数の NAS を統合した場合のファイルへのアクセスを表す図を示す．図 1 のように，file.dat というファイルを操作したい場合は，ファイルの存在確認のためにそれぞれの NAS へ順にアクセスしていく必要がある．複数の NAS を 1 つに見せた場合，あるファイルにアクセスする際にそのファイルがどの NAS に蓄積されているかが分からないため，それぞれの NAS についてファイル・ディレクトリの存在を確認する必要が生じる．大量のデータを扱う際，ファイルの存在確認自体も数多く実行するため，オーバヘッドが問題となる．

以上の議論を踏まえて，2 節で示した 3 つの要件を満たした IoT データ向けのユーザ空間ファイルシステムである「Sensor Data File System (SDFS)」を提案する．SDFS は，時系列 IoT データを効率よく管理するために以下の 5 つの特徴を備えている．

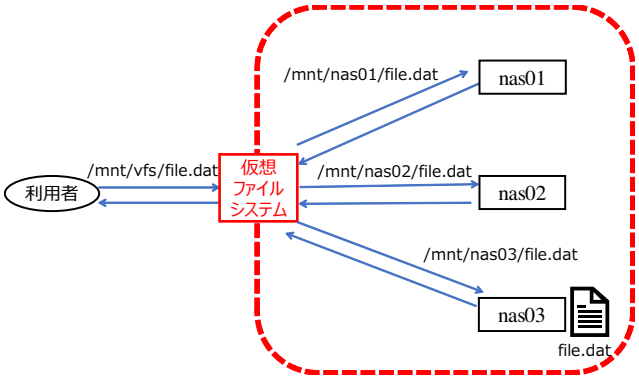


**図 1** 複数の NAS を統合する際のファイルへのアクセス

(1) 複数の NAS が 1 つのファイルシステムとして見える
(2) 設定ファイルの書き換えだけでサービスの再起動なしに新しい NAS を追加できる
(3) ある NAS の残り容量が少なくなった場合には自動的に残り容量が多い NAS に保存される
(4) 日付が近いファイルが同じ NAS に保存されやすくなる仕組みを提供している
(5) 1 つの NAS だけ取り出しても通常のファイル操作ができる

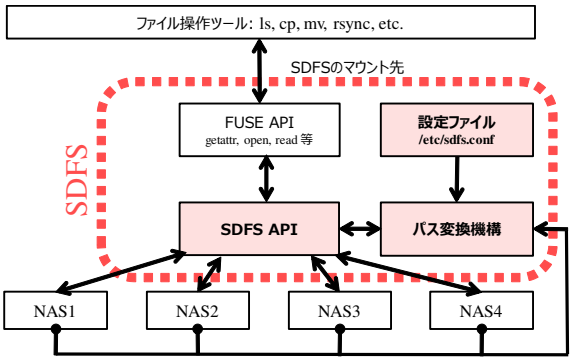


**図 2** SDFS の全体像

SDFS では，日付ごとにセンサデータを保存するためのフォーマットを SDFS パスとして統一することで，複数の NAS を統合しやすい仕組みを実現している．具体的には，/sdfs に SDFS をマウントした場合，/sdfs/[センサデータの種類]/[年]/[月]/[日]/[ファイル名] の形式で統一している．例えば，2019 年 11 月 13 日の加速度センサのファイル名が acc.csv だった場合，SDFS パスは/sdfs/acc/2019/11/13/acc.csv となる．

図 2 に NAS を 4 台統合した場合の SDFS の全体像を示す．SDFS では，既存のファイル操作ツールである cp, mv, rsync 等をそのまま用いることができ，利用に際しては SDFS をマウントしたパス，すなわち SDFS パスへと操作を行う．SDFS は Linux 上に FUSE [26, 27] を用いて実装されている．FUSE はユーザ空間のプログラムにファイルシステムを実装するためのインタフェースである．FUSE を用いたファイルシステムの実装では，各ファイル操作に関連するシステムコールに対応するフック関数をオーバーライドすることで，独自の機能を実装することができる．ファイル操作ツールから SDFS で提供されているファイルやディレクトリに対して操作が行われると，FUSE API を介して SDFS API が呼ばれる．SDFS API は，アプリケーションから受け取った特定のファイルやディレクトリに対する要求をパス変換機構と連携しながら実際のパスに対して処理を行う．パス変換機構は SDFS パスと実際のパスへと変換を行う機構である．パス変換機構は，操作対象のパス，設定ファイルの内容，NAS が保持しているファイル，NAS のディスク残量に応じて，SDFS API に対して操作すべき NAS への実際のパスを与える．設定ファイルは SDFS に NAS を新規に追加したり，データを保管したい NAS を指定するために利用者が編集するファイルである．設定ファイルによって利用者はセンサ毎・時系列毎にデータを保管する NAS を細かく指定することができる．

### 3.2 パス変換機構：設定ファイルの読み込み

パス変換機構の設定ファイルの読み込みはスレッドとして実現されており，定期的に設定ファイルの中身の読み込みを行っている．スレッドでは設定ファイルに記述されている NAS のマウントポイントの情報を取得，マウントしている各 NAS の残り容量をチェックしている．

設定ファイルは SDFS 上のディレクトリのパスパターンと対応先の NAS のマウントポイントのペアで表される．パスパターンは，マウントポイントをルートとした絶対パスで記述する．ワイルドカードを使用することもできる．SDFS が/sdfs に，3 台の NAS がそれぞれの NAS が/mnt/nas01，/mnt/nas02，/mnt/nas03 としてマウントされていたとする．このとき，設定ファイル/etc/sdfs.conf には以下のように記述すると記載されている順に容量が無くなるまで順番に保存されるようになる．

```
/␣/mnt/nas01
/␣/mnt/nas02
/␣/mnt/nas03
```

SDFS は取得したセンサデータの年や月によって，保存する NAS を指定することができる．設定ファイルに以下のように記述した場合は，2017 年 1 月から 9 月のセンサデータは/mnt/nas01 に，2017 年 10 月から 12 月と 2018 年のセンサデータは/mnt/nas02 に，2019 年のセンサデータは/mnt/nas03 に保存されるようになる．

```
/*/2017␣/mnt/nas01
/*/2017/10␣/mnt/nas02
/*/2017/11␣/mnt/nas02
/*/2017/12␣/mnt/nas02
/*/2018␣/mnt/nas02
/*/2019␣/mnt/nas03
```

### 3.3 パス変換機構：ファイルの存在確認

SDFS の特徴を満たす読み込みや書き込みは，パス変換機構のパス変換によって実現されている．読み込み・書き込みを行うファイルの存在確認については，仮想ファイルシステムが対象の NAS の実際のパスを指定してネットワーク越しにアクセスすることになる．

3.1 節にて，仮想ファイルシステムの複数の NAS をまとめる処理はオーバヘッドに繋がる可能性があることを議論した．解決方法として，SDFS のファイルの存在確認にネガティブキャッシュの機能を実装することが考えられる．ネガティブキャッシュとは「そのファイルが存在しない」という情報をキャッシュすることを意味する．ファイルの存在確認時にネットワーク越しに NAS 上のファイル情報を参照に行かずにキャッシュで返すことができれば，オーバーヘッドを抑えることができる．

Algorithm 1 に SDFS のパス変換機構でのファイルの存在確認関数を，表 2 に Algorithm 1 で使用する変数と関数を示す．

例えば，「/mov_b30SS/2019/09/」のディレクトリがとある NAS に存在しないことがキャッシュできていれば，このディレクトリ以下のファイル存在確認要求に対して全て「ファイルは存在しない」と返すことができる．つまり，「/mov_b30SS/2019/09/14/building_001.mov」や「/mov_b30SS/2019/09/14/building_002.mov」などのファイルは，その上位の階層でネガティブキャッシュにヒットするので，オーバーヘッドの増加を限定することができる．

**Algorithm 1** ファイルの存在確認関数

**Require:** $p$
**Ensure:** return **true** if $p$ exists, return **false** if $p$ not exists

```
A ←split(p, "/")
s = ""
for i = 1 to size(A) do
    s ←strcat(s, A[i])
    if checkNCache(s) == true then
        return false
    end if
end for
s = ""
for i = 1 to size(A) do
    s ←strcat(s, A[i])
    if lstat(s) == 1 then
        addNCache(s)
        return false
    end if
end for
return true
```

### 3.4 SDFS API：読み込み

SDFS への読み込みは，複数の NAS に分散されたファイルが 1 つのディレクトリの中にあるかのようにアクセスすることができる．例えば，複数の NAS においてそれぞれの 2019 年 11 月 13 日のディレクトリに 1 つずつ名前の同じファイルを持っていた場合でも，SDFS では 3 つのディレクトリに存在するかのようにアクセスできる．読み込み時の NAS のアクセスは設定ファイルの最長プレフィックスマッチング順に行われる．読み込み対象のファイルのパスが，設定ファイルの記述されているパスパターンと長くプレフィックスマッチングする NAS から順にファイルの存在確認を行い，ファイルが存在した場合に読み込みを行う．

同じ長さでプレフィックスマッチングする NAS が複数存在した場合には，設定ファイルに記述されている順番にファイルの存在確認を行う．異なる NAS に同じパスの同じファイルが存在した場合でも，上記の方法で最も早く見つかったファイルのみが読み込まれる．

**表 2 algorithim 1 で使用する変数，関数**

| 変数，関数 | 説明 |
|---|---|
| $p$ | 存在確認対象のパス． |
| $A$ | 階層毎に分解したパスを格納するリスト． |
| split($s_1$, $s_2$) | 文字列 $s_1$ を $s_2$ の文字列で区切って配列として返す関数． |
| strcat($s_1, s_2$) | $s_1$ と $s_2$ を結合した文字列を返す関数． |
| checkNCache($s$) | $s$ がネガティブキャッシュに存在するかどうかを確認する関数． |
| lstat($s$) | ファイル $s$ の状態確認をする関数．存在していれば 0，しなければ 1 を返す． |
| addNCache($s$) | 文字列 $s$ をネガティブキャッシュに登録する関数． |

### 3.5 SDFS API：書き込み

SDFS への書き込みの手順は以下の 3 つの段階から構成される．第 1 段階は，書き込み対象ファイルの存在確認である．対象ファイルの存在確認は前節に記載した読み込みと同じ手順で行う．書き込み対象のファイルが SDFS 内に存在する場合は，その NAS が書き込み先として選択されて上書きされる．存在しない場合は，第 2 段階に進む．また，複数の NAS の同じパスに同名のファイルが存在する場合は，設定ファイルに書かれている順序が最も上位の NAS のファイルが上書きされる．

第 2 段階は，設定ファイル内のパスパターンとのマッチングである．書き込み対象のファイルのパスが設定ファイルの記述されているパスパターンと一致し，かつ残り容量が十分な NAS に書き込まれる．パスパターンにマッチングした NAS の残り容量が不十分な場合は，第 3 段階に進む．一致するパスパターンが複数ある場合は最も長くプレフィックスマッチングしたパスパターンに，同じ長さでプレフィックスマッチングしたパスパターンが複数存在する場合は設定ファイルに先に記述された NAS が優先される．

第 3 段階は，残り容量が十分にあって，設定ファイルの上位に記載されている NAS が選択される．設定ファイルの上位に記載されている順番にファイルを保存することで，生成された日時が近いセンサデータが同じ NAS に保存されて時系列性をなるべく維持するように工夫している．

## 4. 評価

### 4.1 評価環境

SDFS の有用性を相対的に評価するため，以下の 3 つのファイルシステムを比較する．

(1) SDFS
本稿で提案する，大量のセンサデータを効率よく管理するためのファイルシステム．複数の NAS でセンサデータをまとめて管理することを想定している．

(2) unionfs-fuse [28]
複数の異なるファイルシステムのディレクトリ同士を透過的に重ねることができるファイルシステム UnionFS [29,30] の FUSE ベースの実装．

(3) aufs [31]
UnionFS の信頼性とパフォーマンスを改良する目的で開発されたファイルシステム．

NAS のネットワークを介した共有方法は Samba と Network File System(NFS) [32] がある．いずれもローカルに接続されたストレージ対してネットワークを介したリモートマウントを可能にするプロトコルである．Samba は，Server Message Block(SMB) の Linux サポートとして知られている．NFS はデータ共有の手段として，仮想基盤のストレージ等に利用されているプロトコルである．評価に用いた NAS は Synology DS218j に 4TB の HDD2 枚を RAID1 で構築した．

### 4.2 SDFS と既存手法の比較

SDFS の実用性を定性的に評価するために，以下の評価基準を用いて既存手法との比較を行いたい．

(1) NAS の数を増やすと性能が低下するか
(2) Samba に対応しているかどうか
(3) カーネルの再構築が必要かどうか
(4) ファイルシステムの再構築無しで NAS の追加が可能
(5) センサデータの日付に応じて書き込む NAS を指定可能
(6) ストレージ残量に応じた書き込みが可能

上記の評価基準における比較結果を，表 3 に示す．

aufs は Samba で共有したストレージには対応しておらず，性能評価の際は NFS でストレージを共有した．SDFS, unionfs-fuse，aufs のファイルシステムはいずれも重ね合わせる NAS の数を増やしても性能がほぼ低下しない．SDFS と unionfs-fuse は FUSE ベースの実装であるため，カーネルを再構築する必要はない．一方で aufs はカーネルモジュールが提供されていて，利用にはカーネルを再構築する必要がある．オンサイトにおいて，導入のし易さは重要であるため表のような結果になっている．SDFS は設定ファイルの書き換えのみで新しい NAS を追加することができるが，unionfs-fuse と aufs ではファイルシステムの再構築が必要となる．NAS を追加する度にファイルシステムを止めていては，毎分・毎秒と増え続けるセンサデータに対応できない．SDFS はセンサデータの種類・日付やストレージ残量に応じて書き込む NAS を指定することができる．unionfs-fuse や aufs もディレクトリ名に応じて書き込む NAS を指定できるが，3.2 節にあるような詳細な指定はできない．

**表 3** SDFS と既存手法の比較

| | SDFS | unionfs -fuse | aufs |
|---|---|---|---|
| NAS 数を増やした際の性能 | ○ | ○ | ○ |
| Samba に対応 | ○ | ○ | × |
| カーネル再構築の必要無し | ○ | ○ | × |
| ファイルシステムを止めること無く NAS の追加が可能 | ○ | × | × |
| センサデータの日付に応じて書き込む NAS を指定可能 | ○ | △ | △ |
| ストレージ残量に応じた書き込みが可能 | ○ | × | × |

### 4.3 rsync の性能による評価

SDFS の実用性を定量的に評価するため，rsync の性能を調査する．センサにより大量のデータが蓄積・保管・利用されるような現場では，センサデータのバックアップなどに rsync を利用することが考えられる．NAS を 1 台から 4 台まで統合する台数を変えながら，ファイルサイズ 256B のファイル 1 万個を rsync を使って同期を行った．図 3 に Samba で NAS を共有した場合の実験結果を載せる．縦軸が rsync が完了した時間で，横軸が統合した NAS の台数である．図 3 より，SDFS は unionfs-fuse よりも rsync の性能が優れていることが分かる．しかしながら，SDFS はネガティブキャッシュの機能無しでは，NAS の台数が増えるにつれて性能が大幅に低下していることがわかる．

図 4 に NFS で NAS を共有した場合の実験結果を載せる．図 4 より，NFS での共有において aufs が rsync の性能的に最も良かったことが分かる．一方で，SDFS と unionfs-fuse は Samba で共有した場合と違って性能にほとんど差が無い．

### 4.4 読み込み・書き込み性能の評価

SDFS の実用性を定量的に評価するため，読み込み・書き込みスループットを計測した．書き込みスループットは書き込むファイルサイズを 1 B から 10 倍刻みで 1 GB まで変化させ，10 回ファイル書き込みを実行した平均を算出

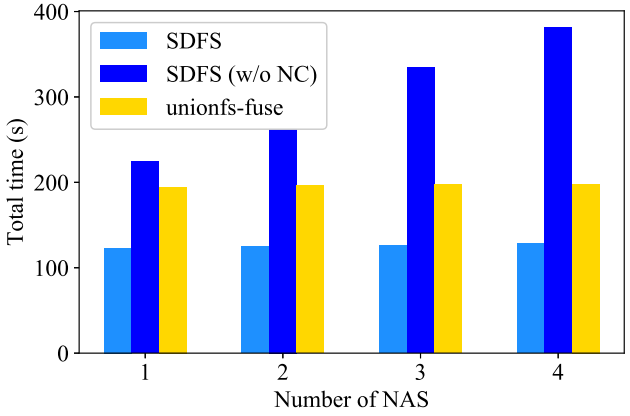


**図 3** rsync の性能（Samba）

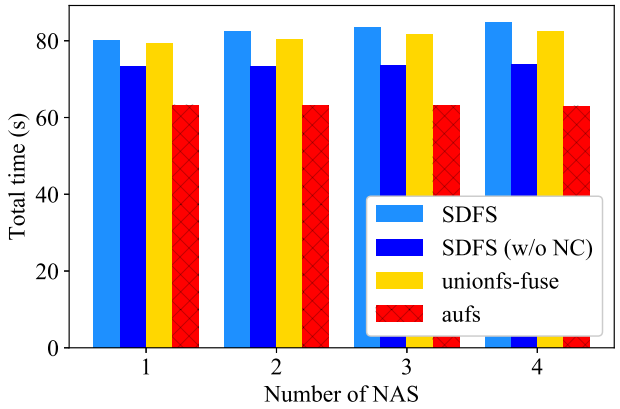


**図 4** rsync の性能（NFS）

した．図 5 に Samba で，図 6 に NFS でマウントした際のスループットを示す．縦軸がスループット (Mbps)，横軸が書き込みの際のファイルサイズ (B) である．

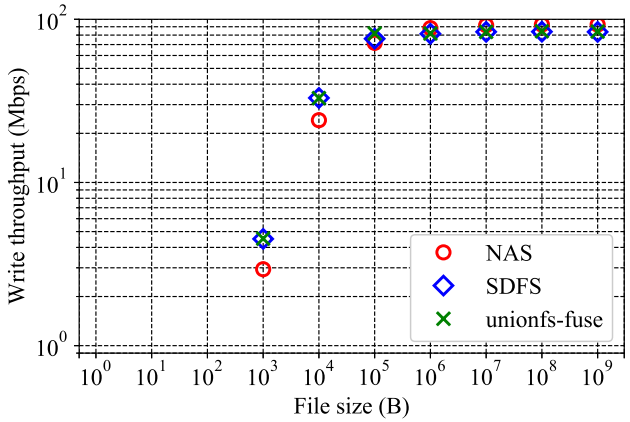


**図 5** 書き込みスループットの性能（Samba）

読み込みスループットに関しても，同様の実験を行った．図 7 に Samba で，図 8 に NFS でマウントした際のスループットを載せる．図 5,6,7,8 より，SDFS は既存手法とほぼ同等の読み書きスループットを達成していることが分かる．

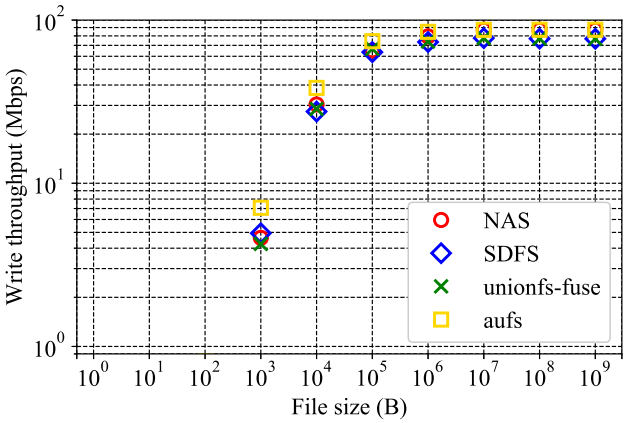


**図 6** 書き込みスループットの性能（NFS）

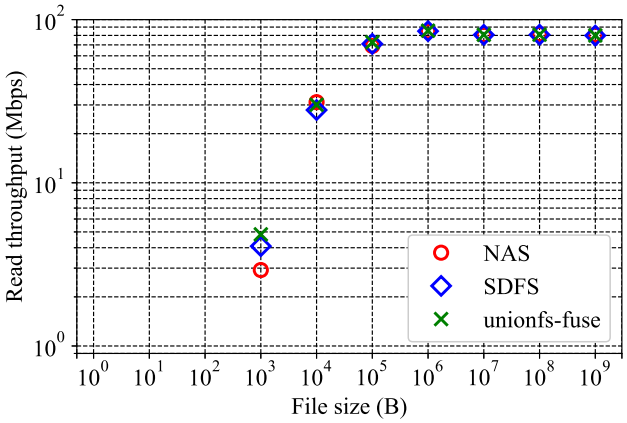


**図 7** 読み込みスループットの性能（Samba）

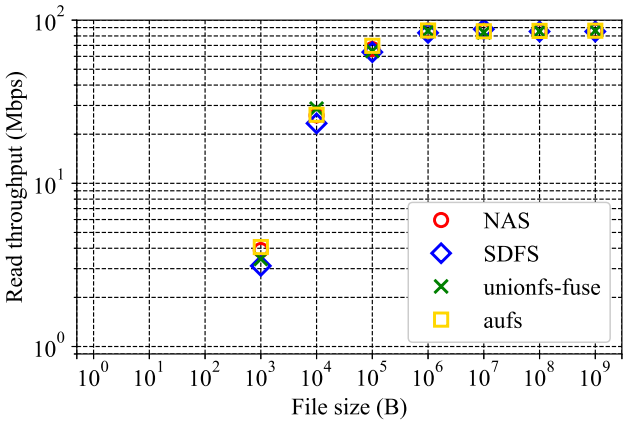


**図 8** 読み込みスループットの性能（NFS）

### 4.5 ネガティブキャッシュの生存時間による評価

これまでの評価では，パス変換機構でのネガティブキャッシュの生存期間を 60 秒としていた．ネガティブキャッシュの生存時間が SDFS の性能に与える影響を検証する．具体的には，NAS4 台を Samba で共有し，SDFS のネガティブキャッシュの生存時間を変えながらファイルサイズ 256B のファイル 1 万個を rsync を使って同期を行った．

図 9 に実験結果を載せる．縦軸が rsync が完了した時間で，横軸がネガティブキャッシュの生存時間である．図 9

より，生存時間が 1 秒のときを除いてネガティブキャッシュは有効に作用したといえる．さらに，生存時間が 10 秒以上のとき，rsync の性能はほとんど変わらないことが分かる．

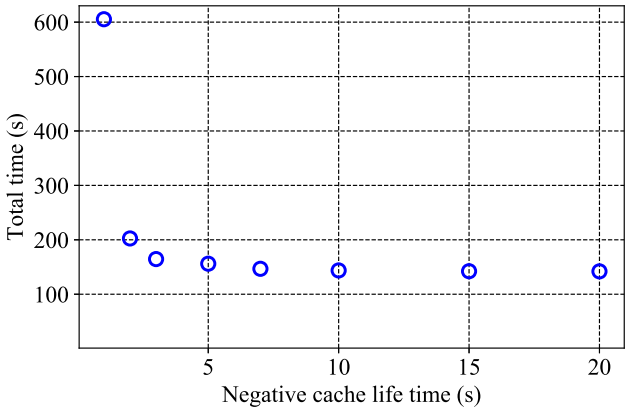


**図 9** rsync の性能（Samba）

### 4.6 設定ファイルの読み込み間隔による評価

これまでの評価では，パス変換機構での設定ファイルの読み込み間隔を 60 秒としていた．設定ファイルの読み込み間隔が SDFS の性能に与える影響を検証する．具体的には，NAS4 台を Samba で共有し，設定ファイルの読み込み間隔を $10^{-4}$ 秒から 10 秒まで 10 倍刻みで変化させた場合の書き込みと読み込みのスループットを計測した．ファイルサイズが 10kB のファイルの書き込み・読み込みを 10 回実行した際のスループットの平均を算出した．

図 10，図 11 に読み込み間隔を変化させた場合の書き込み・読み込みのスループットを示す．縦軸がスループット [Mbps]，横軸が設定ファイルの読み込み間隔 [s] である．図 10, 11 より，設定ファイルの読み込み間隔が 1 秒以上の場合は，読み書きスループット性能に影響を与えないことが分かった．

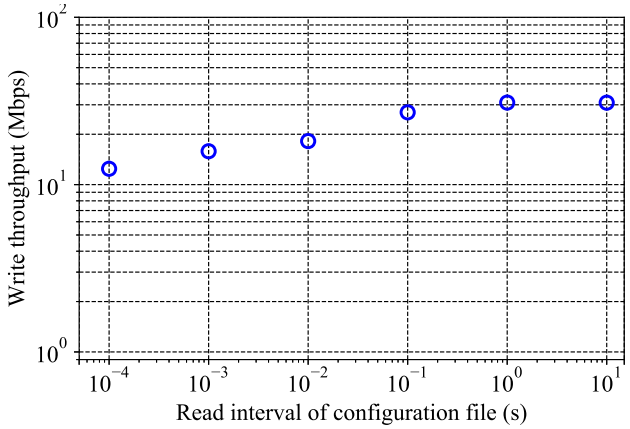


**図 10** 書き込みスループットの性能（Samba）

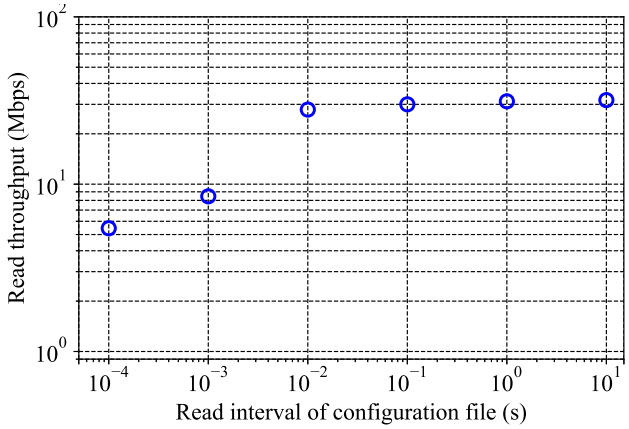


**図 11** 読み込みスループットの性能（Samba）

## 5. 関連研究

Union mount [33] は複数の異なるファイルシステムのファイルやディレクトリ同士を透過的に重ねることができる技術のことである．Union Mount の使用例として，CD や DVD などの光学メディアから起動するライブ版の Linux ディストリビューションが挙げられる．このようなディストリビューションでは，光学メディアへの書き込みをメモリに保存して，Union Mount により光学メディアとメモリを透過的に重ねることで，擬似的に光学メディアに書き込むことを実現している．Union mount の技術を利用したファイルシステムの代表的なものには，UnionFS [29] や aufs [31]，OverlayFS [34] などがある．これらの積み重ねが可能なファイルシステムには一定の需要があり，これまで熱心に開発されてきた [35]．aufs は UnionFS から派生したファイルシステムで，安定性やパフォーマンスの改善から多くのサービスで使用されている．は aufs の後継として開発されたファイルシステムが Overlayfs である．

本稿で提案している SDFS もディレクトリを重ねることができるという点で，Union Mount の一種であると考えられる．SDFS は既存の Union Mount に比べ増え続ける IoT データを扱う際に利点がある、複数の NAS を統合することを想定したファイルシステムである，

## 6. おわりに

本稿では大量のセンサデータを蓄積する際の課題を明らかにし，増加し続ける IoT データを効率的に蓄積・管理する複数 NAS 統合型ファイルシステム「SDFS」を提案した．SDFS の実用性を定性的・定量的評価をもって確認した．SDFS は複数の NAS が 1 つのファイルシステムとして見えるだけでなく，センサデータを管理しやすい特徴を兼ね備えていて，ストレージの追加なども容易に行える．

**謝辞** 本研究は JSPS 科研費 JP17KT0042 及び NTT アクセスサービスシステム研究所の支援の下で行った．

ⓒ 2020 Information Processing Society of Japan

## 参考文献

[1] アキバで買ったセンサで見事に成功！中小製造業：https://news.aperza.jp.
[2] 黒木琴海，小寺志保，倉田成人，濱本卓司，猿渡俊介：環境発電型センサシステムのためのデータ中心型タスクスケジューリング方式，情報処理学会論文誌，Vol. 57, No. 11, pp. 2475–2488 (2016).
[3] 黒木琴海，富岡昭浩，猿渡俊介，倉田成人，濱本卓司：軍艦島モニタリングプロジェクト，技術報告 20，静岡大学大学院, 日本航空電子, 大阪大学, 筑波技術大学, 東京都市大学 (2016).
[4] 守屋広汰，小寺志保，倉田成人，濱本卓司，猿渡俊介：軍艦島モニタリングに向けた首振りカメラによる映像取得方式の検討，第 78 回全国大会講演論文集，No. 1, pp. 153–154 (2016).
[5] 黒木琴海，小寺志保，倉田成人，濱本卓司，猿渡俊介：軍艦島センサネットワークのためのタスクスケジューリングの設計と評価，技術報告 21，静岡大学情報学部, 静岡大学大学院情報学研究科, 筑波技術大学, 東京都市大学, 静岡大学大学院情報学研究科 (2015).
[6] 黒木琴海，佐藤　匠，小寺志保，倉田成人，濱本卓司，猿渡俊介：軍艦島モニタリングに向けたクロスレイヤ型タスクスケジューリング方式，第 77 回全国大会講演論文集，Vol. 2015, No. 1, pp. 261–262 (2015).
[7] 小寺志保，倉田成人，濱本卓司，渡辺　尚，猿渡俊介：軍艦島モニタリングに向けた映像処理方式の提案, 電子情報通信学会ソサイエティ大会 (2015).
[8] 倉田成人，猿渡俊介：軍艦島モニタリングに期待されるICT，電子情報通信学会，Vol. 100, No. 11, pp. 1176–1181 (2017).
[9] 岡田隆三，黒木琴海，倉田成人，濱本卓司，富岡昭浩，大胡拓矢，田村博規，河本　満，大島　純，渡辺　尚，猿渡俊介：軍艦島モニタリングシステムの実装とその運用，情報処理学会モバイルコンピューティングとパーベイシブシステム (MBL) 研究会，Vol. 20, pp. 1–8 (2017).
[10] 岡田隆三，小寺志保，富岡昭浩，倉田成人，濱本卓司，猿渡俊介：軍艦島全域センサネットワーク構築に向けた検討，情報処理学会第 78 回全国大会， Vol. 1, pp. 229–230 (2016).
[11] 岡田隆三, 黒木琴海, 倉田成人, 濱本卓司, 猿渡俊介：Contiki OS を用いた複数シンク対応型 RPL の実装, 電子情報通信学会ソサイエティ大会 (2016).
[12] 濱本卓司，倉田成人，猿渡俊介，富岡昭浩：軍艦島モニタリングプロジェクト（その 1）研究計画と予備計測/長期計測，2015 年度日本建築学会大会 (2015).
[13] 富岡昭浩，濱本卓司，倉田成人，猿渡俊介：軍艦島モニタリングプロジェクト（その 2）長期振動計測システム，2016 年度日本建築学会大会 (2016).
[14] 関根明日香，濱本卓司，富岡昭浩，倉田成人，猿渡俊介：軍艦島モニタリングプロジェクト（その 3）長期モニタリングに基づく軍艦島 70 号棟の動的挙動に関する考察，2016 年度日本建築学会大会 (2016).
[15] 倉田成人，濱本卓司，猿渡俊介，富岡昭浩：軍艦島モニタリングプロジェクト（その 4）日本最古の鉄筋コンクリート造集合住宅 30 号棟の画像モニタリング，2016 年度日本建築学会大会 (2016).
[16] 鶴岡　湧，崔　井圭，濱本卓司：軍艦島モニタリングプロジェクト（その 5）ウェアラブルカメラとドローンを用いた軍艦島 30 号棟の劣化調査，2016 年度日本建築学会大会 (2016).
[17] 富岡昭浩，濱本卓司，倉田成人，猿渡俊介：軍艦島モニタリングプロジェクト（その 6）MEMS 加速度センサネットワークの構成，2017 年度日本建築学会大会 (2017).
[18] 関根明日香，鶴岡　湧，濱本卓司，倉田成人，猿渡俊介，富岡昭浩：軍艦島モニタリングプロジェクト（その 7）30 号棟の振動計測と劣化調査，2017 年度日本建築学会大会 (2017).
[19] 鶴岡　湧，関根明日香，濱本卓司，倉田成人，猿渡俊介，富岡昭浩：軍艦島モニタリングプロジェクト（その 8）日給社宅と 65 号棟の振動計測と劣化調査，2017 年度日本建築学会大会 (2017).
[20] 濱本卓司，倉田成人，猿渡俊介，富岡昭浩：軍艦島モニタリングプロジェクト（その 9）視覚センシングと聴覚センシングとの融合，2017 年度日本建築学会大会 (2017).
[21] 富岡昭浩，濱本卓司，倉田成人，猿渡俊介：軍艦島モニタリングプロジェクト（その 10）MEMS 加速度センサネットワークの構成，2018 年度日本建築学会大会 (2018).
[22] 倉田成人，佐々木謙二，猿渡俊介，濱本卓司，富岡昭浩：軍艦島モニタリングプロジェクト（その 11）3 号棟に設置した自律型気象センサシステム，2018 年度日本建築学会大会 (2018).
[23] 関根明日香，濱本卓司，富岡昭浩，大胡拓矢，倉田成人，猿渡俊介：軍艦島モニタリングプロジェクト（その 12）MEMS 加速度センサネットワークの構成，2018 年度日本建築学会大会 (2018).
[24] 小林正純，濱本卓司：軍艦島モニタリングプロジェクト（その 13）移動画像検査による軍艦島 30 号棟の崩壊危険度評価，2019 年度日本建築学会大会 (2019).
[25] 小林正純，濱本卓司：軍艦島モニタリングプロジェクト（その 14）移動画像検査による軍艦島日給社宅の崩壊危険度予測，2019 年度日本建築学会大会 (2019).
[26] M.Szeredi: Filesystem in Userspace, http://fuse.sourceforce.net (2005).
[27] Vangoor, B. K. R., Tarasov, V. and Zadok, E.: To FUSE or Not to FUSE: Performance of User-Space File Systems, *15th USENIX Conference on File and Storage Technologies (FAST 17)*, Santa Clara, CA, USENIX Association, pp. 59–72 (online), available from ⟨https://www.usenix.org/conference/fast17/technical-sessions/presentation/vangoor⟩ (2017).
[28] Podgorny, R.: unionfs-fuse, https://github.com/rpodgorny/unionfs-fuse.
[29] Wright, C. P. and Zadok, E.: Kernel Korner: Unionfs: Bringing Filesystems Together, *Linux J.*, Vol. 2004, No. 128, pp. 8– (2004).
[30] Pendry, J. S. and McKusick, M. K.: Union mounts in 4.4BSD-Lite, *In Proceedings of the USENIX Technical Conference on UNIX and Advanced Computing Systems*, pp. 25–33 (1995).
[31] aufs: http://aufs.sourceforge.net/aufs.html.
[32] Radkov, P., Yin, L., Goyal, P., Sarkar, P. and Shenoy, P.: A Performance Comparison of NFS and iSCSI for IP-Networked Storage, *Proceedings of the 3rd USENIX Conference on File and Storage Technologies*, FAST '04, Vol. 1, Berkeley, CA, USA, USENIX Association, pp. 101–114 (2004).
[33] Unionmounts: http://valerieaurora.org/union/.
[34] Torvalds, L.: OverlayFS, https://github.com/torvalds/linux/tree/master/fs/overlayfs.
[35] Zadok, E., Iyer, R., Joukov, N., Sivathanu, G. and Wright, C. P.: On Incremental File System Development, *Trans. Storage*, Vol. 2, No. 2, pp. 161–196 (2006).