

コンシューマ・システム論文

DIY-SDFS：オンサイト利用を想定した IoT データ向け複数NAS統合型ファイルシステム

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)}

受付日 2020年6月30日, 採録日 2020年11月4日

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「Do It Yourself-Sensor Data File System (DIY-SDFS)」を提案する. DIY-SDFS はユーザ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 (Filesystem in USErspace) を使用して開発した. DIY-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. DIY-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, 仮想ファイルシステム, Network Attached Storage

DIY-SDFS: A NAS Interated File System for On-site IoT Data Storage

YUKI OKAMOTO^{1,a)} KENICHI ARAI² TORU KOBAYASHI² TAKUYA HUJHASHI¹
TAKASHI WATANABE¹ SYUNSUKE SARUWATARI^{1,b)}

Received: June 30, 2020, Accepted: November 4, 2020

Abstract: In the case of IoT data acquisition, long-term storage of large amounts of sensor data is a significant challenge. When a cloud service is used, fixed costs such as a communication line for uploading and a monthly fee are unavoidable. In addition, the use of large-capacity storage servers incurs high initial installation costs. In this paper, we propose a Network Attached Storage (NAS) integrated file system called the “Do It Yourself-Sensor Data File System (DIY-SDFS)”, which has advantages of being on-site, low-cost, and highly scalable. We developed the DIY-SDFS using FUSE (Filesystem in USErspace), which is an interface for implementing file systems in user-space. DIY-SDFS not only allows multiple NAS to be treated as a single file system but also has functions that facilitate the ease of the addition of storage. In this paper, we compared DIY-SDFS with existing integrated storage modalities such as aufs and unionfs-fuse. The results indicate that DIY-SDFS achieves an equivalent throughput compared with existing file systems in terms of read/write performance.

Keywords: IoT, virtual file system, Network Attached Storage

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 Internet of Things (IoT) が爆発的に広まってきている. IoT の応用 [1], [2] 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

¹ 大阪大学大学院情報科学研究科
Information Science and Technology, Osaka University,
Suita, Osaka 565-0871, Japan

² 長崎大学大学院工学研究科
Graduate School of Enginnering, Nagasaki University,
Nagasaki 852-8521, Japan

^{a)} okamoto.yuki@ist.osaka-u.ac.jp

^{b)} saru@ist.osaka-u.ac.jp

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にかかっている。たと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ることで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 [3]。

IoT サービス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 章で詳細に議論する。

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「Do It Yourself-Sensor Data File System (DIY-SDFS)」を提案する [4], [5]。DIY-SDFS はオンサイトで日々増え続ける IoT データを効率的に蓄積・管理するためのファイルシステムとして、最小限の機能を備えている。そのファイルシステムは現場の人が自ら実装し、必要があれば独自の要望を満たすように拡張していくソフトウェアにしたいという意味をこめて DIY-SDFS と名付けている。DIY-SDFS の最大の特徴は「システム稼働状態のまま容量を拡張できる」ことである。その他の特徴としては、「複数の NAS が 1 つのファイルシステムとして見える」「トラブル発生時にファイルシステムを止めずにトラブルシューティングが可能である」ことがあげられる。DIY-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で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。

また、本稿が提案する DIY-SDFS は増え続けるデータを効率的に保存するため、すでに実運用されている。図 1 はミラーリングされた 4TB の HDD を積んだ NAS を 2 台、DIY-SDFS を用いて運用している様子である。NAS は Synology DS218j という約 2 万円程度のものを使用している。筆者が属している研究室で使われているネットワークのトラフィックを tcpdump でキャプチャし、DIY-SDFS に保管している。ここまで、約 30 日間の安定動作をしている。



図 1 DIY-SDFS を実運用する様子

Fig. 1 DIY-SDFS of operation.

本稿の構成は以下のとおりである。2 章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 章で提案手法である DIY-SDFS の全体像と実装について述べ、4 章でその DIY-SDFS の評価をまとめている。5 章では関連研究について触れ、6 章でまとめとする。

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収集始めること自体の敷居は下がって来ている。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 [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28], [29]。2016 年よりセンサデータの収集を開始し、2020 年 6 月現在では約 10 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わたってくると、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) 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
- (4) データの保存が安価にできること

の 4 つの要件を満たしたストレージシステムが求められる。

1 つ目の「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ときから増え続ける。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すべて蓄積しておくことが望ましい。たとえば、軍艦島モニタリングシステムは、2020 年 6 月現在で 10 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。このような場合においても NAS を簡単に管理できるためには、運用中に NAS の容量が足りなくなった場合に自動的に保存する NAS を切り替えることができれば望ましい。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となる。時系列的に保存するには 1 つのストレージに保存することが望ましい。

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システムが必要となる。

4 つ目は、「データの保存が安価にできる」ことである。IoT システムはスモールスタートで予算がないところから始まることも多く、データを保存する手段も導入の難易度を下げるために安価であることが望ましい。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。たとえば、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。8TB のストレージを利用して、1 カ月あたり 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など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。

表 1 クラウドストレージの月額使用料の例

Table 1 Example of monthly usage amounts for cloud storage.

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

総量 8TB, 1 カ月あたり 300 GB のデータアップロードする場合

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、低予算で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。また、クラウドサービスの利用料金にはバックアップを作成するといったような保守費用も含まれている。しかし、クラウドサービスではバックアップを作成できる機能が存在していたとしても一時的な停止や大規模なデータロスが発生する可能性もあり、データが完全に利用できなくなることも考えられるため、この点においてもデータを自由にバックアップしていつでも利用できる本提案システムは有利である。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ことが考えられる。しかしながら、ストレージサーバは一般的には約 50~100 万円と高価な物が多い。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：DIY-SDFS

3.1 DIY-SDFS の全体像

2 章で述べた要件へ対応するため、安価な NAS を複数統合して扱うことのできる 1 つのストレージシステムとする仮想ファイルシステムを構築することを考える。ユーザが統合したい任意の NAS についての情報を記述するための設定ファイルを用いることとする。設定ファイルに記述された NAS が構築するファイルシステムへと 1 つに統合されることになる。仮想ファイルシステムは設定ファイルの情報を参照しながら、複数の NAS に散らばったディレクトリ・ファイルを束ねる役割を担う。このとき、仮想ファイルシステムは統合するそれぞれの NAS について順番にファイルの存在を確認する必要がある。図 2 は仮想ファイルシステムで複数の NAS を統合する場合に、file.dat というファイル进行操作したいときのファイルアクセスを表している。図 2 のように、複数の NAS を仮想ファイルシステムによって 1 つに見せることを考えたとき、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

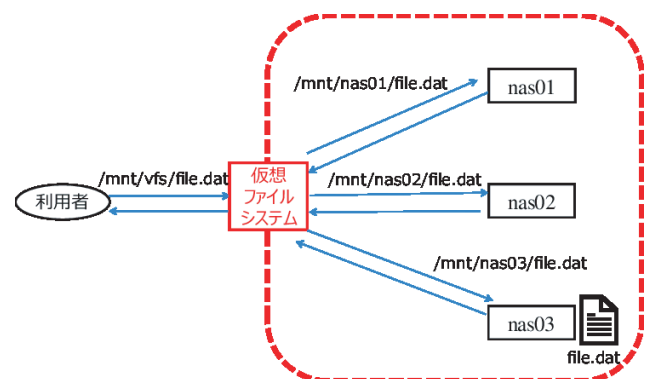


図 2 複数の NAS を統合する際のファイルへのアクセス

Fig. 2 Access to files in multiple NAS integrated VFS.

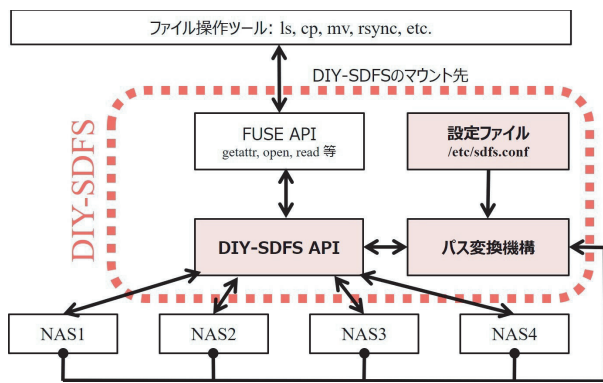


図 3 DIY-SDFS の全体像
Fig. 3 Overview of DIY-SDFS.

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、オーバーヘッドが問題となる。

以上の議論をふまえて、2章で示した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「Do It Yourself-Sensor Data File System (DIY-SDFS)」を提案する。増え続けるIoTデータを効率的に管理するための最小な機能を提供するという意味でDIY-SDFSと名付けている。DIY-SDFSは、時系列IoTデータを効率良く管理するために以下の5つの特徴を備えている。

- (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。

統合するNASの中の自由なディレクトリにファイルを保存しては、データの集中的な管理が難しくなり複数のNASが統合されにくくなる。DIY-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DIY-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DIY-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。たとえば、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、DIY-SDFSパスは/sdfs/acc/2019/11/13/acc.csvとなる。

図3にNASを4台統合した場合のDIY-SDFSの全体像を示す。DIY-SDFSでは、既存のファイル操作ツールであるcp, mv, rsyncなどをそのまま用いることができ、利用に際してはDIY-SDFSをマウントしたパス、すなわち

DIY-SDFSパスへと操作を行う。DIY-SDFSはLinux上にFUSE (Filesystem in Userspace) [30], [31]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DIY-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DIY-SDFS APIが呼ばれる。DIY-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DIY-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、DIY-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DIY-SDFSにNASを新規に追加したり、データを保管したいNASを指定したり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ごと・時系列ごとにデータを保管するNASを細かく指定することができる。さらに、DIY-SDFSは稼働中に設定ファイルの内容を定期的に読み込み、システムに反映させる機能を持っているので、サービスの再起動なしに新しいNASを追加することができる。

3.2 設定ファイル：NASの管理

DIY-SDFSはユーザが記述する設定ファイルの情報を用いてNASの管理を行っている。パス統合機構によってこの設定ファイルが読み込まれると、設定ファイルに記述されていたNASがDIY-SDFSに統合される。この読み込みをメイン処理とは別にスレッドとして定期的に行うことで、再起動なしにDIY-SDFSにNASを追加・除外することを実現している。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DIY-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。DIY-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なくなるまで順番に保存されるようになる。

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

DIY-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、2018 年 1 月から 9 月のセンサデータは/mnt/nas01 に、2018 年 10 月から 12 月と 2019 年のセンサデータは/mnt/nas02 に、2020 年のセンサデータは/mnt/nas03 に保存されるようになる。

```
/*/2018_/_mnt/nas01
/*/2018/10_/_mnt/nas02
/*/2018/11_/_mnt/nas02
/*/2018/12_/_mnt/nas02
/*/2019_/_mnt/nas02
/*/2020_/_mnt/nas03
```

DIY-SDFS では、ユーザが設定ファイルに書き込み終了するとシステムが設定ファイルを読みこめるようになる。設定ファイルの読み込みが完了するとすぐその内容が反映されるようになる。もし設定ファイルを正しいフォーマットで記述していなかった場合エラーとなる。この場合は、正しいフォーマットで記述し直し、DIY-SDFS を再起動することで内容が反映されるようになる。

3.3 パス変換機構：ファイルの存在確認

DIY-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 節で、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ー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、DIY-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 DIY-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。

ファイルの存在確認関数では、各 NAS に対して実際に操作するディレクトリ・ファイルの存在確認を行う。このとき、見つからなかったファイル・ディレクトリを見つからなかったタイミングでネガティブキャッシュへと登録している。たとえば、「/mov_b30SS/2020/06/」のディレクトリがあるとある 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すべて「ファイルは存在しない」と返すことができる。つまり、「/mov_b30SS/2020/06/23/building_001.mov」や「/mov_b30SS/2020/06/25/building_001.mov」などのファ

表 2 Algorithm 1 で使用する変数、関数
Table 2 Variables and functions used in Algorithm 1.

変数、関数	説明
p	存在確認対象のパス。
A	階層ごとに分解したパスを格納するリスト。
$\text{split}(s_1, s_2)$	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。
$\text{strcat}(s_1, s_2)$	s_1 と s_2 を結合した文字列を返す関数。
$\text{checkNCache}(s)$	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。
$\text{lstat}(s)$	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。
$\text{addNCache}(s)$	文字列 s をネガティブキャッシュに登録する関数。

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。

DIY-SDFS はこのネガティブキャッシュをファイルの存在確認の関数内に導入しており、DIY-SDFS パスのような階層的なディレクトリ構造に対して効率的にファイル操作を行うことができる。

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
1:  $A \leftarrow \text{split}(p, "/")$ 
2:  $s = ""$ 
3: for  $i = 1$  to  $\text{size}(A)$  do
4:    $s \leftarrow \text{strcat}(s, A[i])$ 
5:   if  $\text{checkNCache}(s) == \text{true}$  then
6:     return false
7:   end if
8: end for
9:  $s = ""$ 
10: for  $i = 1$  to  $\text{size}(A)$  do
11:    $s \leftarrow \text{strcat}(s, A[i])$ 
12:   if  $\text{lstat}(s) == 1$  then
13:      $\text{addNCache}(s)$ 
14:     return false
15:   end if
16: end for
17: return true
```

3.4 DIY-SDFS API：読み込み

DIY-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。たとえば、複数の NAS においてそれぞれの 2020 年 6 月 23 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には、そのディレクトリの 1 つのファイルとして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。

3.5 DIY-SDFS API：書き込み

DIY-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 DIY-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。したがって、同パス同名のファイルが複数あっても書き込みにより上書きを行ったファイルを必ず読み込むことが可能である。これにより、書き込み対象にならなかった同パス同名のファイルは設定ファイルの優先順位が変わらない限り DIY-SDFS から扱われることはない。

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

DIY-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) DIY-SDFS

本稿で提案する、大量のセンサデータを効率良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。

表 3 DIY-SDFS と既存手法の比較

Table 3 Comparison of DIY-SDFS and existing filesystem.

	DIY-SDFS	unionfs-fuse	aufs
Samba に対応	○	○	×
カーネル再構築の必要なし	○	○	×
ファイルシステムを止めずに 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	○	×	×
ファイルシステムを止めずにトラブルシューティング可能	○	×	×

(2) unionfs-fuse [32]

複数の異なるファイルシステムのディレクトリどうしを透過的に重ねることができるファイルシステム UnionFS [33], [34] の FUSE ベースの実装。

(3) aufs [35]

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) [36], [37] がある。いずれもローカルに接続されたストレージに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などに利用されているプロトコルである。

評価に用いた計算機のスペックとして、CPU は Intel Core i7-4600U 2.10 GHz、メモリは 8 GB、OS は Ubuntu 18.04 である。評価に用いた NAS は Synology DS218j の HDD 台を RAID1 で構築している。

4.2 DIY-SDFS と既存手法の比較

オンサイトで利用されるストレージシステムとしての DIY-SDFS の実用性を評価するために、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 数を増やした際の性能
- (2) 読み書きの性能
- (3) Samba に対応しているかどうか
- (4) カーネルの再構築が必要かどうか
- (5) ファイルシステムを止めずに NAS の追加が可能
- (6) センサデータの日付に応じて書き込む NAS を指定可能
- (7) ストレージ残量に応じた書き込みが可能
- (8) ファイルシステムを止めずにトラブルシューティング可能

上記の評価基準 (3) から (8) における比較結果を、表 3 に示す。

NAS 数を増やした際の性能について、IoT システムのオンサイトで利用されるストレージシステムとして比較が必要である。IoT システムの稼働時間が長くなれば取得したセンサデータの量も増え、NAS の数も増えていくことが予想される。そのため、NAS の数が増えていくことがファイルシステムのパフォーマンスにどの程度影響が及ぼすのかの検証が必要である。そこで、DIY-SDFS、unionfs-fuse、aufs で統合する NAS を増やしていくとファイルシステムの性能にどの程度影響が及ぼすのかを 4.3 節の実験で検証している。また、DIY-SDFS の性能に関して、3.3 節で取り上げたファイルの存在確認時に使われているネガティブキャッシュに関する評価も行う必要がある。ネガティブキャッシュの有用性を評価するため、生存時間を変えていった場合に rsync のパフォーマンスにどの程度影響するのかを計測した。このネガティブキャッシュに関する具体的な実験と結果は 4.5 節で取り上げている。

読み書きの性能による評価とは、単純な読み込み・書き込みのスループットである。4.3 節では rsync という同期に用いるアプリケーションソフトウェアを用いてパフォーマンスを比較しているが、読み込み・書き込みのスループットについても実験・既存手法との比較を行った。読み込み・書き込みスループットについての評価は 4.4 節で扱っている。

Samba に対応しているかどうかについては、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。

カーネルの再構築が必要かどうかについては、IoT システムが導入される現場において、カーネルの再構築をする必要がないということは、導入のしやすさという点で重要である。DIY-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。

ファイルシステムを止めずに NAS の追加ができることについては、DIY-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。ファイルシステムを止めずに NAS を追加できるようにするためには、3.2 節で取り上げた設定ファイルによる NAS の管理方法が関係している。そのため、ネガティブキャッシュの読み込み間隔を変えた場合のパフォーマンスへの影響や設定ファイルの反映までにかかる時間などを計測した。この設定ファイルの読み込み間隔・反映に関する具体的な実験と結果は 4.7 節で取り上げている。

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について、DIY-

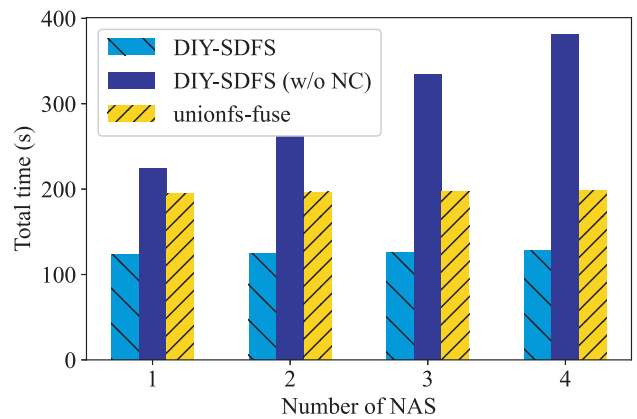


図 4 rsync の性能 (Samba)

Fig. 4 rsync performance (Samba).

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。

ファイルシステムを止めずにトラブルシューティング可能について、運用中に HDD などの故障でストレージシステムが使えなくなると困るため、保守の面で重要である。

たとえば、高価なストレージサーバを使用していると RAID のディスク 1 つだけを取り出したとしてもそこからファイルを復元するのは極めて難しい。DIY-SDFS では何かトラブルがあったときでも NAS 単体での動作もできるのでトラブルシューティングがしやすい。上記から、DIY-SDFS はオンサイト利用において他の手法に比べて耐障害性についても利点があることが分かる。

4.3 rsync の性能による評価

DIY-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。

複数の NAS を DIY-SDFS、unionfs-fuse、aufs で統合する前提で、統合する NAS の台数を 1 台から 4 台まで変えながら、以下の実験を行った。256 B のファイル 1 万個が入ったディレクトリを rsync で転送する。ファイル同期元はファイル 1 万個が入ったディレクトリで、同期先は DIY-SDFS と unionfs-fuse のマウント先である。同期先はファイル数が 0 であり、新規にすべてのファイルが作成されるようになっている。IoT ではデータの発生頻度が多く、まとまった 1 つのデータよりもファイルサイズの小さなデータをたくさん処理することが多い。よって、データサイズが 256 B のファイルを 1 万個用いて実験を行った。

図 4 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。

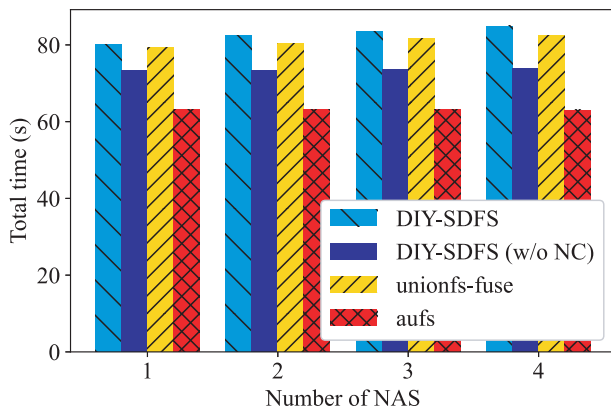


図 5 rsync の性能 (NFS)

Fig. 5 rsync performance (NFS).

図 4 中の DIY-SDFS (w/o NC) とは、ネガティブキャッシュ機能をオフにしたときの DIY-SDFS を指している

図 4 より、DIY-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、DIY-SDFS はネガティブキャッシュの機能なしでは、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分かる。

図 5 に NFS で NAS を共有した場合の実験結果を載せる。図 5 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、DIY-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ない。rsync では、ファイルが存在するかどうか、ディレクトリが存在するかどうか、更新する必要があるかどうかなど、さまざまなタイミングで数多くファイルの存在確認をする必要がある。この結果の違いは、NFS は Samba と違ってプロトコル自体にネガティブキャッシュの機能が実装されているからであると考えられる。

4.4 読み込み・書き込み性能の評価

DIY-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 1kB から 10 倍刻みで 1GB 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した。図 6 に Samba で、図 7 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 6、図 7 中の NAS とは、単に NAS をそのままマウントして計測した結果を指している。

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 8 に Samba で、図 9 に NFS でマウントした際のスループットを載せる。図 6～図 9 より、DIY-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。

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

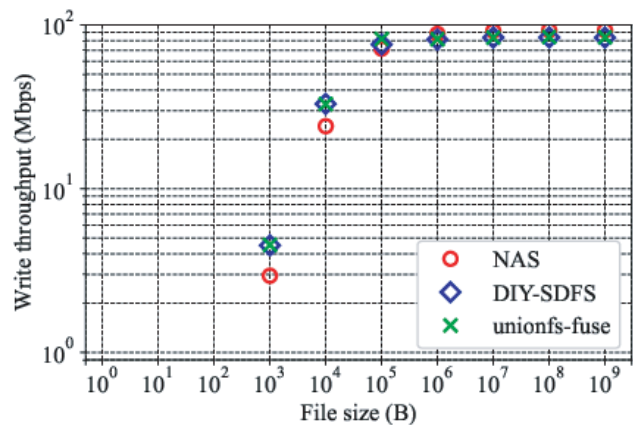


図 6 書き込みスループットの性能 (Samba)

Fig. 6 Write throughput performance (Samba).

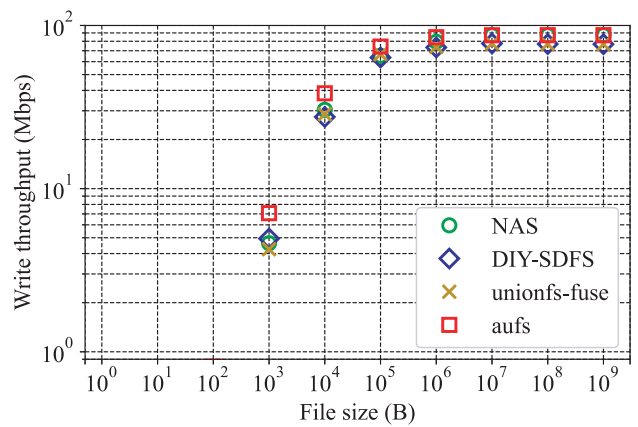


図 7 書き込みスループットの性能 (NFS)

Fig. 7 Read throughput performance (NFS).

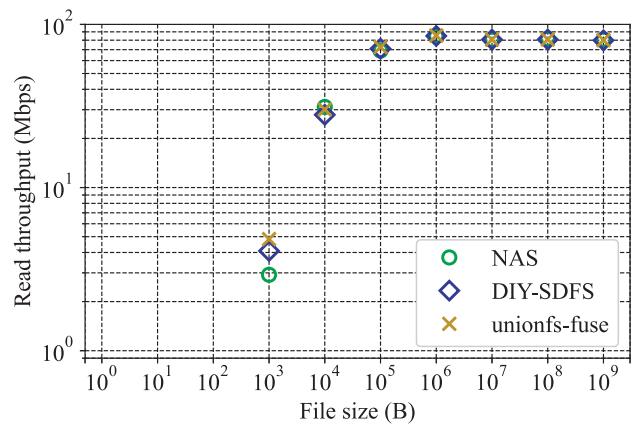


図 8 読み込みスループットの性能 (Samba)

Fig. 8 Write throughput performance (Samba).

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 DIY-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、DIY-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 256 B のファイル 1 万個を rsync を使って同期を行った。

図 10 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。

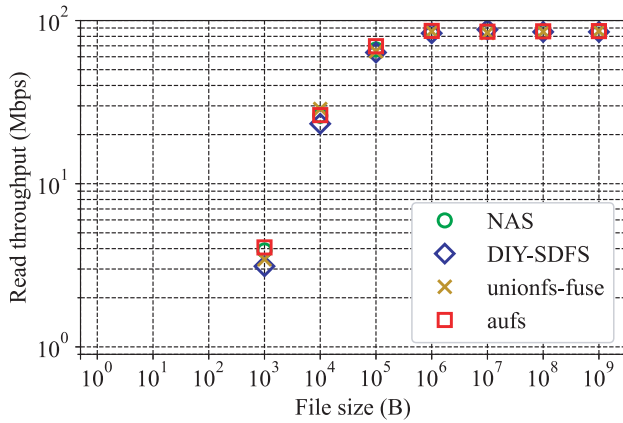


図 9 読み込みスループットの性能 (NFS)

Fig. 9 Write throughput performance (NFS).

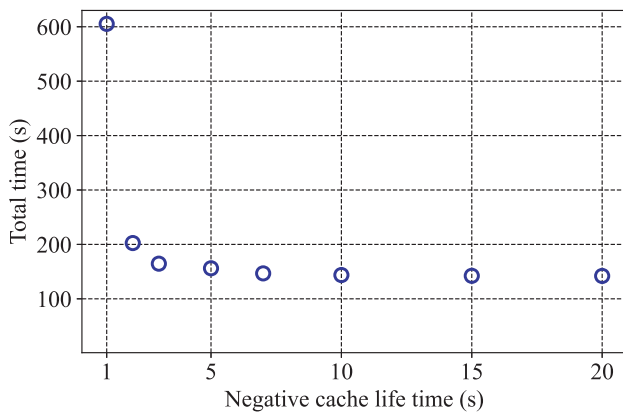


図 10 rsync の性能 (Samba)

Fig. 10 rsync performance (Samba).

図 10 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 秒以上のときは、グラフが平行線をたどっていることから rsync の性能はほとんど変わらないことが分かる。rsync はアプリケーションの特性上、ファイルの同期を順々に行っていく。そのため、ネガティブキャッシュとして情報を保持している期間が短くても rsync の性能改善に繋がったと考えられる。

4.6 設定ファイルの読み込み間隔と反映時間による評価

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 DIY-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。10kB というファイルサイズは実際の IoT データとして妥当な大きさであり、今回はこのファイルサイズを用いて実験を行った。

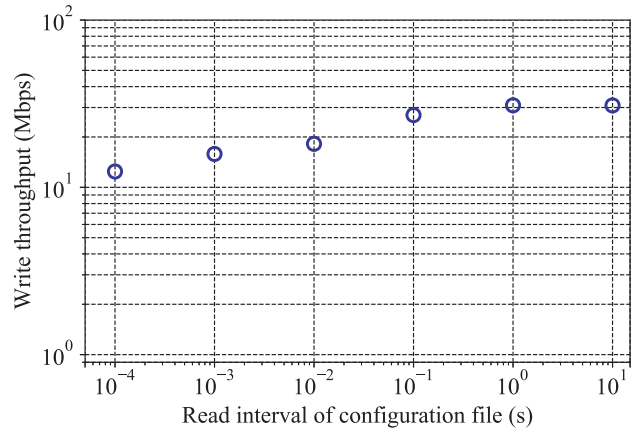


図 11 書き込みスループットの性能 (Samba)

Fig. 11 Write throughput performance (Samba).

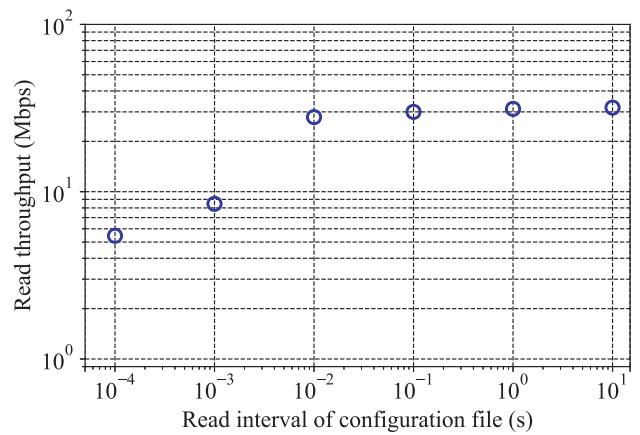


図 12 読み込みスループットの性能 (Samba)

Fig. 12 Read throughput performance (Samba).

図 11、図 12 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。

また、設定ファイルを変更してから、NAS の追加・除外がシステムに反映されるまでの時間を計測する実験を行った。ここで、DIY-SDFS の設定ファイルの読み込み間隔は 60 秒としている。前提として DIY-SDFS にはすでに 1 台の NAS が運用されており、そこに新しく NAS をシステムに追加することで実験を行った。除外に関しては、DIY-SDFS に複数台の NAS が統合されている状態から 1 台になるまで NAS を除外することで実験を行った。この追加・除外する NAS の数を徐々に変えながら、システムに反映されるまでの時間を計測した。図 13 に DIY-SDFS に NAS に追加・除外したときの実験結果を示す。横軸がシステムに新しく追加する NAS の数で、縦軸がシステムに反映されるまでの時間を表している。

4.7 実験結果の考察

図 4 より、DIY-SDFS は unionfs-fuse よりも rsync の性能が優れていることが分かる。また、unionfs-fuse において

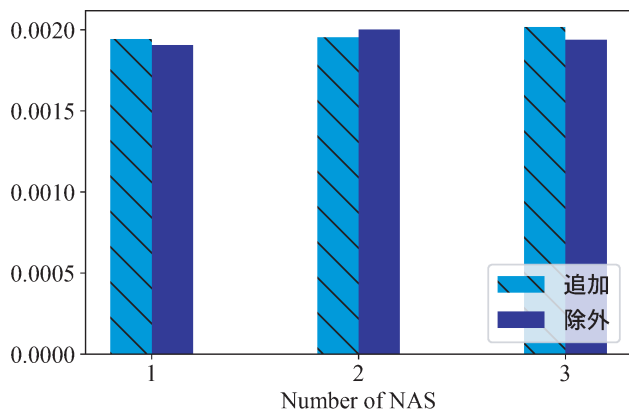


図 13 NAS の追加・除外による実験結果

Fig. 13 Experimental results by adding / excluding NAS.

も NAS の台数を増やしたときに性能が変わらないような工夫がされていることが分かる。これは、3.3 節で取り上げている本提案手法におけるネガティブキャッシュの実装が関係していると考えられる。DIY-SDFS は DIY-SDFS バスにあてはまるディレクトリ構成のデータを上位のディレクトリからネガティブキャッシュに取り込む。そのため、本実験において DIY-SDFS はより上位の階層でネガティブキャッシュにヒットさせることができ、unionfs-fuse を性能面で追い抜いたと考えられる。

図 5 において、aufs が他の手法よりも優れた結果を出している。この理由は、DIY-SDFS と unionfs-fuse はユーザ空間でも動作しているのに比べて、aufs はカーネル空間のみで動作するからと考えられる。また、図 5 では DIY-SDFS (w/o NC) の結果が DIY-SDFS よりも優れていることが分かる。これについては、NFS にすでにネガティブキャッシュの機能が実装されているので、DIY-SDFS に実装したことがオーバーヘッドとなったと考えられる。

図 5 の NFS における DIY-SDFS と unionfs-fuse との性能差は、図 4 の Samba のときに比べて小さくなっている。この理由として、元々 DIY-SDFS が備えていたネガティブキャッシュの機能を NFS が使ってくれているため、パフォーマンスに差が出ることがなくなったと考えられる。

図 11, 図 12 より、設定ファイルの読み込み間隔が 1 秒以上の場合、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 13 より、設定ファイルを読み込んでシステムに反映するまでの処理時間は 0.002 秒程度である。これらの結果から、DIY-SDFS のファイルの読み書きに影響を与えるのは、ファイルの読み込み間隔が 1 秒のときやそれよりも小さいときであることが分かる。

5. 関連研究

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どうしを透過的に重ねることができる、Union Mount [38] という技術がある。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あげられる。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 [33] や aufs [35], OverlayFS [39] 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 [40]。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。Overlayfs は aufs の後継として開発されたファイルシステムである。

本稿で提案している DIY-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。DIY-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「DIY-SDFS」を提案した。DIY-SDFS の実用性を定性的・定量的評価をもって確認した。DIY-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。

また、本稿では DIY-SDFS の rsync・読み書きスループットの定量評価について取り上げた。今後の課題として、rsync 以外のソフトウェアなど、単純な読み書き以外の手法での評価を行い、DIY-SDFS のパフォーマンスを裏付けることがあげられる。

謝辞 本研究は JSPS 科研費 JP17KT0042 および 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。

参考文献

- [1] Lee, I. and Lee, K.: The Internet of Things (IoT): Applications, investments, and challenges for enterprises, *Business Horizons*, Vol.58 (2015).
- [2] Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M. and Ayyash, M.: Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications, *IEEE Communications Surveys Tutorials*, Vol.17, No.4, pp.2347–2376 (2015).
- [3] アキバで買ったセンサで見事に成功！中小製造業，入手先 (<https://news.aperza.jp>).
- [4] Okamoto, Y., Arai, K., Kobayashi, T., Fujihashi, T., Watanabe, T. and Saruwatari, S.: A NAS Integrated File System for On-site IoT Data Storage, *4th International Workshop on Mobile and Pervasive Internet of*

- Things (PerIoT'20 IEEE PerCom 2020 Workshops)*, pp.1–6 (2020).
- [5] Okamoto, Y.: DIY-SDFS, available from <https://github.com/watalabo/DIY-SDFS>.
- [6] 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).
- [7] 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).
- [8] 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).
- [9] 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).
- [10] 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).
- [11] 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).
- [12] 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).
- [13] 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シブシステム (MBL) 研究会, Vol.20, pp.1–8 (2017).
- [14] 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).
- [15] 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).
- [16] 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).
- [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 2) 長期振動計測システム, 2016 年度日本建築学会大会 (2016).
- [18] 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).
- [19] 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).
- [20] 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).
- [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 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).
- [22] 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).
- [23] 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).
- [24] 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).
- [25] 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).
- [26] 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).
- [27] 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).
- [28] 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).
- [29] 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).
- [30] Szeredi, M.: Filesystem in Userspace, available from <http://fuse.sourceforge.net> (2005).
- [31] 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)*, pp.59–72, USENIX Association (2017) (online), available from <https://www.usenix.org/conference/fast17/technical-sessions/presentation/vangoor>.
- [32] Podgorny, R.: unionfs-fuse, available from <https://github.com/rpodgorny/unionfs-fuse>.
- [33] Wright, C.P. and Zadok, E.: Kernel Korner: Unionfs: Bringing Filesystems Together, *Linux J.*, Vol.2004, No.128, p.8 (2004).
- [34] Pendry, J.S. and McKusick, M.K.: Union mounts in 4.4BSD-Lite, *Proc. USENIX Technical Conference on UNIX and Advanced Computing Systems*, pp.25–33 (1995).
- [35] aufs, available from <http://aufs.sourceforge.net/aufs.html>.
- [36] Radkov, P., Yin, L., Goyal, P., Sarkar, P. and Shenoy, P.: A Performance Comparison of NFS and iSCSI for IP-Networked Storage, pp.101–114 (2004).
- [37] nfs, available from <https://github.com/torvalds/linux/tree/master/fs/nfs>.
- [38] Unionmounts, available from <http://valerieaurora.org/union/>.
- [39] Torvalds, L.: OverlayFS, available from <https://github.com/torvalds/linux/tree/master/fs/overlayfs>.
- [40] 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 (online), DOI: 10.1145/1149976.1149979 (2006).



岡本 祐樹 (学生会員)

1996 年生。2019 年神戸大学情報知能工学科卒業。同年大阪大学大学院修士課程入学。制約プログラミング, IoT, ストレージシステムに関する研究に従事。



荒井 研一 (正会員)

2004 年信州大学工学部情報工学科卒業。2006 年同大学大学院工学系研究科博士前期課程修了。2010 年同大学院総合工学系研究科博士課程修了。博士 (工学)。以来、情報セキュリティに関する研究に従事。2011 年東京理科大学理工学部嘱託助教。2015 年長崎大学大学院工学研究科助教。2020 年より同大学大学院工学研究科准教授。電子情報通信学会、日本応用数理学会各会員。



小林 透 (正会員)

1985 年東北大学工学部精密機械工学科卒業。1987 年同大学大学院工学研究科修士課程修了。博士 (工学)。同年 NTT 入社。以来、ソフトウェア生産技術、情報セキュリティ、データマイニング、Web 技術等の研究開発に従事。1998~2002 年までドイツ、デュッセルドルフに駐在し欧州研究機関と Web 技術、セキュリティ技術に関する共同研究開発、およびスマートカードに関する標準化活動に従事。2013 年長崎大学大学院工学研究科教授。2017 年から情報担当副学長。IEEE (シニア)、電子情報通信学会 (シニア) 各会員。本会シニア会員。



藤橋 卓也 (正会員)

2012 年静岡大学情報学部情報科学科卒業。2013 年同大学大学院情報学研究科情報学専攻修了。2016 年大阪大学大学院情報科学研究科情報ネットワーク学専攻修了。博士 (情報科学)。2014~2016 年日本学術振興会特別研究員。2014~2015 年 Mitsubishi Electric Research Laboratories 滞在。2017~2019 年愛媛大学大学院理工学研究科助教。2019 年より大阪大学大学院情報科学研究科助教。映像伝送、無線ネットワークに関する研究に従事。2014 年電気通信普及財団テレコムシステム技術賞奨励賞。電子情報通信学会、IEEE 各会員。



渡辺 尚 (正会員)

1982 年大阪大学工学部通信工学科卒業。1984 年同大学大学院博士前期課程修了。1987 年同大学院博士後期課程修了。工学博士。同年徳島大学工学部情報工学科助手。1990 年静岡大学工学部情報知識工学科助教授。1996 年静岡大学情報学部情報科学科教授。2008 年同大学創造科学技術大学院教授。2013 年大阪大学大学院情報科学研究科教授。1995 年文部省在外研究員 (カリフォルニア大学アーバイン校)。計算機ネットワーク、分散システムに関する研究に従事。情報処理学会理事、電子情報通信学会アドホックネットワーク研究会副委員長、等。訳書『計算機設計技法』、『802.11 無線ネットワーク管理』等。IEEE 会員。



猿渡 俊介 (正会員)

2007 年東京大学大学院博士課程修了。博士 (科学)。2003~2004 年 IPA 未踏ソフトウェア創造事業。2006~2008 年日本学術振興会学振特別研究員。2007~2008 年イリノイ大学客員研究員。2008~2012 年東京大学先端科学技術研究センター助教。2012~2015 年静岡大学大学院情報学研究科助教 (テニュアトラック)。2015~2016 年静岡大学大学院情報学研究科講師。2016 年より大阪大学大学院情報科学研究科准教授。専門はワイヤレスネットワーク、センサネットワーク、システムソフトウェア等。2009 年電子情報通信学会論文賞。2010 年情報処理学会山下記念研究賞。電子情報通信学会、IEEE、ACM 各会員。