情報処理学会論文誌 Vol.61 No.3 707–717 (Mar. 2020)

推薦論文

CoW機能を考慮したOFF2Fプログラムのページ例外処理の評価

谷口 秀夫[1,a)] 佐藤 将也[1,b)] 河辺 誠弥[1,c)] 横山 和俊[2,d)]

受付日 2019年5月23日, 採録日 2020年1月16日

概要：ハードウェア技術の進歩により，不揮発性メモリが登場している．この登場を受け，計算機は，実メモリとして，揮発性メモリと不揮発性メモリを混載する構成が考えられる．この環境を想定し，仮想記憶機構が2つのメモリの特徴を生かしたプログラムの実行をできるように，新しい実行プログラムのファイル形式（OFF2F：Object File Format consisting of 2 Files）が提案されている．本論文では，仮想記憶機構の要求時ページング（ODP：On Demand Paging）機能の下でOFF2Fプログラムを実行する際，CoW（Copy on Write）機能を考慮したページ例外（PF：Page Fault）処理について述べ，PF処理の時間を定式化し，OFF2Fプログラムを実行することによるPF処理時間の短縮効果を示す．また，OS初期化処理時間の短縮効果を述べる．

キーワード：不揮発性メモリ，実行プログラム形式，ページ例外処理，ODP，CoW

Evaluation of Page Fault Handling of OFF2F Program with CoW Function

Hideo Taniguchi[1,a)] Masaya Sato[1,b)] Seiya Kawabe[1,c)] Kazutoshi Yokoyama[2,d)]

Received: May 23, 2019, Accepted: January 16, 2020

Abstract: Non-volatile memory is appeared by evolving hardware technologies. With this, the main memory of a computer would be consisting of volatile and non-volatile memory-mixed environment. Assuming the environment, OFF2F (Object File Format consisting of 2 Files), a new file format for exploiting characteristics of the two memory types, is proposed. This paper shows the processing of page fault handling considering copy-on-write (CoW) when executing an OFF2F program under the function for on-demand paging (ODP) of a virtual memory system. We also formulated the time for page fault handling and show the effect of reduction of the time for page fault handling by executing OFF2F programs. Besides, we show the effect of the reduction of the initialization time of an operating system.

Keywords: non-volatile memory, format of executable program, page fault handling, ODP, CoW

1. はじめに

メモリが揮発性ではなく不揮発性であれば，データの操作や格納の方法が大きく変わることを想定し，メモリを不揮発性（Non-Volatile Memory：以降，NVメモリと略す）として仮想的に扱う研究 [1], [2], [3] がある．

一方，最近のハードウェア技術の進歩により，不揮発性

1 岡山大学大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University, Okayama 700–8530, Japan
2 高知工科大学情報学群
School of Information, Kochi University of Technology, Kami, Kochi 782–8502, Japan
a) tani@cs.okayama-u.ac.jp
b) sato@cs.okayama-u.ac.jp
c) kawabe@swlab.cs.okayama-u.ac.jp
d) yokoyama.kazutoshi@kochi-tech.ac.jp

本論文の内容は2018年10月の電気・情報関連学会中国支部連合大会にて報告され，支部長により情報処理学会論文誌ジャーナルへの掲載が推薦された論文である．

メモリが登場している．この登場を受け，不揮発性メモリを有効利用するソフトウェア技術も研究を開始している．ファイルシステムでの有効利用に関する研究として，揮発性の実メモリと外部記憶装置の利用に合わせてNVメモリの利用形態を変更させる方法 [4]，高速なNVメモリと低速なSSDの2つを階層型のストレージとして扱うATSMF [5]，NVメモリの容量を仮想拡張する手法VEMS [6]，ファイルシステムのメタデータにNVメモリを利用する研究 [7] がある．また，データの書き出しにNVメモリをキャッシュとして利用する研究 [8] がある．消費電力に着目し，HPCにおいてNVメモリと揮発性メモリの搭載比率が性能と省電力に与える影響に関する研究 [9] がある．メモリとして扱う研究として，NVメモリをメモリ階層の1つとして利用する研究 [10]，CPUのキャッシュとNVメモリ上のデータの一貫性を最適化する研究 [11] がある．さらに，不揮発性メモリも含め様々な特徴を保つメモリを言語記述として見せる研究 [12] がある．

また，プログラム実行に関する研究 [13], [14], [15] がある．NVメモリをログ構造化ファイルシステムの一部に組込むことで強い一貫性保証と高速化を実現する手法 [13] が提案されている．また，NVメモリを主記憶として利用する際に揮発性メモリを少量だけ混載し，データ構造の配置を工夫することで，インメモリ処理を行う応用プログラムの性能低下を大幅に抑える研究 [14] がある．さらに，文献 [15] では，揮発性メモリとNVメモリが混載された計算機において，仮想記憶機構が2つのメモリの特長を生かしたプログラムの実行をできるように，新しい実行プログラムのファイル形式を提案している．具体的には，プログラムをメモリ上に配置したときのアクセス形態に着目し，2つのファイルからなる実行ファイル形式（OFF2F：Object File Format consisting of 2 Files）である．文献 [13] は，ファイルシステムに大幅な改変を加える必要があるが，OFF2Fは，従来のファイルシステムを維持しつつ，ページ例外処理の一部を改変するだけでNVメモリに配置したファイルを実行可能にしている．また，文献 [14] は，読み込み遅延の大きい大容量低価格のNVメモリを想定しているが，OFF2Fは，読み込み遅延がDRAMと同等で少容量高価格のNVメモリを想定している．

本論文では，仮想記憶機構の要求時ページング（ODP：On Demand Paging）機能の下でOFF2Fプログラムを実行する際，CoW（Copy on Write）機能を考慮したページ例外（PF：Page Fault）処理について述べる．また，PF処理の時間を定式化し，OFF2Fプログラムを実行することによるPF処理時間の短縮効果を示す．さらに，OS初期化処理時間の短縮効果としてFreeBSDを取り上げ，OFF2Fプログラム実行の効果を述べる．

2. 揮発/不揮発メモリ混載環境

従来，計算機の実メモリは揮発性のメモリであり，電源OFFとともに保持データは消失する．これに対し，電源OFF後においてもデータを継続保持できるNVメモリが登場している．NVメモリは，アクセス速度の向上，大容量化，消費電力低減，耐久性の向上，および価格低下といった性能や機能の向上が著しい．このため，NVメモリの不揮発性という特徴を生かし，たとえば，NVメモリにファイルシステムを構築してPDAで利用されている．

揮発性メモリは，アクセス速度が高速であり，これまでの技術革新により大容量かつ低価格である．これに対し，ハードウェア構造やエネルギー効率の性質上，相対的に見ると，NVメモリは，アクセス速度（特に，書き込み速度）が低速であり，少容量かつ高価格である．したがって，実メモリがすべてNVメモリ搭載になる構成ではなく，揮発性メモリとNVメモリが共存する実メモリ構成のプロセッサ環境が，今後の主流になる．

一方，不揮発性という共通の特徴を持つ外部記憶装置とNVメモリを比較すると，アクセス単位の面で大きな違いがある．外部記憶装置はブロック単位であるのに対し，NVメモリはバイト単位である．また，磁気ディスク装置（DK）やソリッドステートドライブ（SSD）といった外部記憶装置は，大容量，低価格および高耐久性である．

したがって，今後の計算機として，実メモリは揮発性メモリとNVメモリの混載した構成になる．なお，揮発性メモリとNVメモリは，物理アドレス空間上で連続したアドレスに配置されている必要はない．これは，揮発性メモリとNVメモリを混載したとしても，利用法はOSから制御できるためである．また，外部記憶装置は，バス接続され，入出力装置の位置づけである．

3. OFF2F と ODP 機能

3.1 OFF2F

仮想記憶機構のODP機能を利用し，実行ファイルを実行する際，外部記憶装置（たとえばDK）とメモリ間の入出力時間が長いという問題がある．DKではなくSSDを用いた場合も，メモリ間複写に比べるとその入出力時間は長い．NVメモリは，揮発性メモリと同様にバイト単位アクセスが可能であり，読み出しは高速である．一方で，書き込みは低速である．そこで，NVメモリ上にファイルシステムを構築し利用することで，NVメモリから揮発性メモリへのメモリ間複写により，ページ読み出し時間（ページイン処理時間）を高速化できる．しかし，実行ファイル全体をNVメモリ上にファイルとして格納する必要があるため，大きなNVメモリを必要とする．したがって，読み出しのみ行われるデータをNVメモリに格納し，仮想記憶を利用してそのまま仮想空間にマッピングできれば，ODP

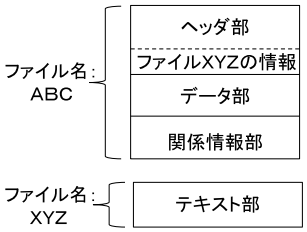


図 1 OFF2F の形式
Fig. 1 OFF2F format.

処理の入出力時間を短縮しつつ大きな NV メモリを必要としない．

実行ファイルは，4 つの内容（テキスト部，データ部，関係情報部，ヘッダ部）から構成される．それぞれの内容は，アクセス形態から大きく 2 つに分類できる．1 つは，読み出しのみ行われる部分であり，テキスト部，関係情報部，およびヘッダ部である．なお，ヘッダ部の読み出しは，主にプログラムをプロセスとして実行するときに発生し，このとき，関係情報部の読み出しは発生しない．一方，テキスト部の読み出しは，プログラム実行時により頻発する．もう 1 つは，頻繁に読み書きされる部分であり，データ部である．

そこで，実行ファイルの内容のアクセス形態に着目し，実行ファイルを 2 つのファイルに分割格納する実行ファイル形式（OFF2F：Object File Format consisting of 2 Files）が提案されている [15]．OFF2F の形式を**図 1** に示す．実行ファイルは 4 つの内容から構成されているため，最大 4 つのファイルに分割格納できる．しかし，OFF2F では，構造の複雑化を防ぎ，またファイルシステムでの占有領域を抑制するため，構成するファイル数を 2 個にしている．

図 1 の形式では，ファイル ABC を外部記憶装置上に置き，ファイル XYZ を NV メモリ上に置く．ファイル ABC は，プログラム実行時の仮想メモリ空間を構築する処理において，既存の処理流れを利用できる．また，ファイル XYZ は，必ずしも NV メモリ上に存在する必要はないため，NV メモリの有無の影響を受けない．

なお，種類数の増加を続ける実行プログラムすべてを NV メモリに配置するのは難しい．このため，実行プログラムの利用頻度やテキスト部の大きさ，および NV メモリの容量に応じて，NV メモリに配置するプログラムを決定する．たとえば，OS 起動処理時に実行されるプログラムや頻繁に利用する OS 提供コマンド，および Web クライアントのようにプログラム起動時間の短縮化が望まれるプログラムである．

3.2 OFF2F プログラムを用いた ODP 機能

仮想記憶機構の ODP 機能において，OFF2F を利用する場合を以下に述べる．ファイル ABC は外部記憶装置上に存在し，ファイル XYZ は NV メモリ上に存在する．このとき，仮想メモリ空間の構築処理について説明する．

(1) 外部記憶装置上に存在するファイル ABC のヘッダ部を読み込む．
(2) ヘッダ部の情報より，テキスト部が NV メモリ上に存在することを認識し，テキスト部の各ページに対応するページテーブル（もしくは，OS が管理する対応ページ管理表）に，NV メモリのアドレスを設定する．
(3) ヘッダ部の情報より，データ部が外部記憶装置上に存在することを認識し，データ部の各ページに対応するページテーブル（もしくは，OS が管理する対応ページ管理表）に，外部記憶装置のアドレスを設定する．

上記処理により，プログラムを実行するとページ例外が発生し，処理が行われる．

4. CoW を考慮したページ例外処理

4.1 処理流れ

CoW は，複製を行う際に，書き換えが必要ない場合は原本を参照させておき，書き換えが必要になった際に初めて複製を作成することで，不要な複写処理を抑制する手法である．ODP 機能において PF 処理を行う際には，CoW 機能により不要なメモリ間複写等の処理を省略できる．OFF2F における ODP 機能および CoW 機能を有する際の PF 処理を**図 2** に示し，以下に説明する．

(1) 実メモリが存在せず，かつ NV メモリ上データに対する例外でない場合，通常の ODP 処理 (A) を行う．このとき，「実メモリ確保」において実メモリ不足が発生すればページアウト処理を行う．その後，CoW 処理の要否を確認し，必要であれば CoW 処理 (C) を行う．
(2) 実メモリが存在せず，かつ NV メモリ上データに対する例外の場合，NV メモリ対応処理 (B) として，当該の NV メモリのページをマッピング表に登録する．
(3) 実メモリが存在する場合，CoW 処理が必要であれば，CoW 処理 (C) を行う．なお，CoW 処理が必要なければ「プログラムのバグ」（パニック処理）である．ここでは，正常な処理内容を中心に記したので，図では省略している．

4.2 ページ例外処理時間の定式化

図 2 に基づき，PF 処理の時間を定式化する．

以下のように各処理の時間を定義する．

- t_1：実メモリ確保（1 ページ：4 KB）
- t_2：外部記憶装置からデータ（4 KB）を読み込む
- t_3：実メモリをマッピング表に登録
- t_4：NV メモリのページをマッピング表に登録
- t_5：メモリ間のデータ複写（4 KB）

さらに，

- P：プログラムにおける（テキスト部＋データ部）が占めるページ数

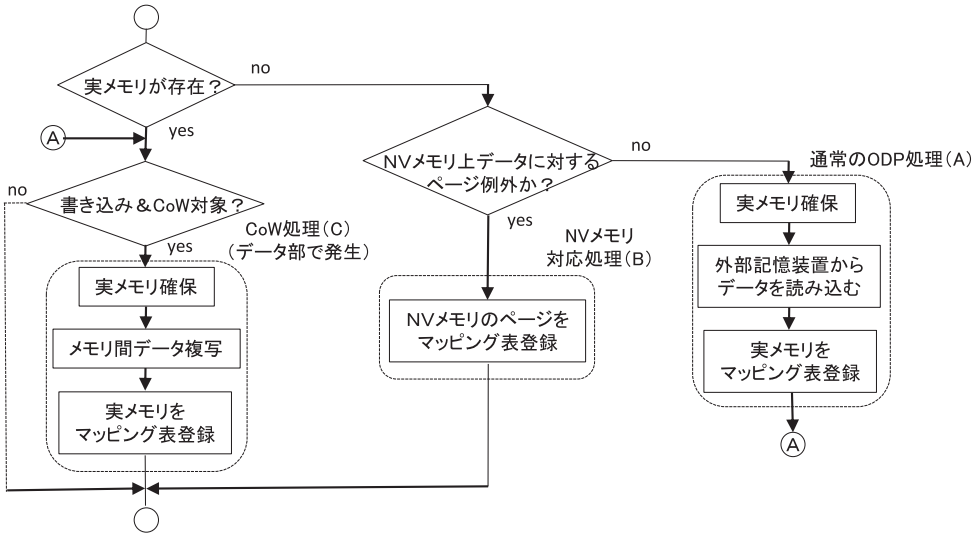


図 2 ページ例外処理の流れ
Fig. 2 Page fault handling.

- S：プログラムに占めるテキスト部の割合
- C_i：i 番目に生成されたプロセスのデータ部に占める CoW 対象ページ数の割合（C_1 は 0 とすることができる）

とする．

次に，プログラムがすべて外部記憶装置に格納されている場合（従来），およびテキスト部は NV メモリ上，他は外部記憶装置に格納されている場合（OFF2F）について定式化する．なお，定式化の条件として，プログラム全体がページ例外によりメモリ上にロードされ，1 つのプログラムから n 個のプロセスを生成し実行する．

（従来）の場合，以下の式 (1) となる．

$$(t_1+t_2+t_3)P+\sum_{i=2}^{n}(t_1+t_5+t_3)P(1-S)C_i \qquad (1)$$

第 1 項はプログラム全体の PF 処理時間であり，第 2 項は 2 個目以降のプロセスで行われる CoW 処理時間の和である．当然ながら，CoW 処理はデータ部のページ例外で発生する．

また，（OFF2F）の場合，以下の式 (2) となる．

$$t_4PS+(t_1+t_2+t_3)P(1-S)+\sum_{i=2}^{n}(t_1+t_5+t_3)P(1-S)C_i \qquad (2)$$

第 1 項はテキスト部の PF 処理時間であり，第 2 項はデータ部の PF 処理時間であり，第 3 項は 2 個目以降のプロセスで行われる CoW 処理時間の和である．

なお，式 (1)，(2) のいずれも，CoW なしの場合は $C_i=0$ とすればよい．また，式 (1)，(2) の差分，つまり（従来）に対する（OFF2F）の時間短縮効果は，以下の式 (3) で表すことができる．

$$(t_1+t_2+t_3-t_4)PS \qquad (3)$$

式 (1)，(2) および (3) から，以下のことが分かる．

(A) 式 (1) の第 2 項および式 (2) の第 3 項は，CoW 処理時間であり，同じ内容である．また，CoW を考慮しない場合 $C_i=0$ であり，t_2 が他の t_i に比べ大きいため OFF2F による PF 処理時間短縮の効果は大きいものの，CoW 機能の処理を考慮すると，効果の割合は小さくなる．

(B) 式 (1)，(2) は，S に関し，1 次関数であり，かつ S の係数は負であり，S の係数の絶対値は式 (2) > 式 (1) である．したがって，プログラムに占めるテキスト部の割合（S）が大きいほど，OFF2F の PF 処理時間の短縮効果は大きくなる．

(C) 式 (1)，(2) は，C_i に関し，1 次関数であり，かつ C_i の係数は正であり，両式とも同じ値である．したがって，データ部に占める CoW 対象ページ数の割合（C_i）に PF 処理時間は比例する．

(D) 式 (3) より，（従来）と（OFF2F）の差は C_i に依存せず，プログラムのテキスト部が占めるページ数に比例して一定である．

5. 評価

5.1 基本性能

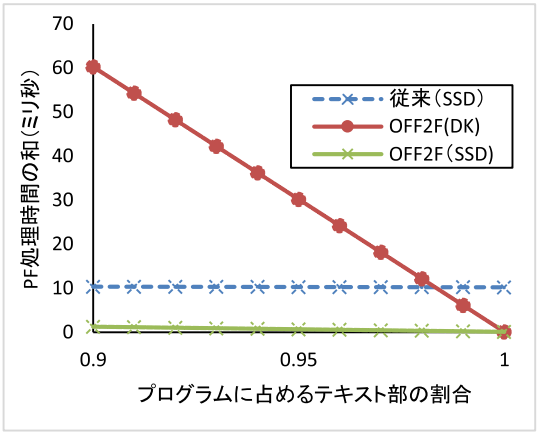
式 (1)，(2) を用い，PF 処理における OFF2F プログラム実行の効果を述べる．

次の環境で，1 GB データについて 4 KB 単位のランダム読み込み処理時間を測定した．

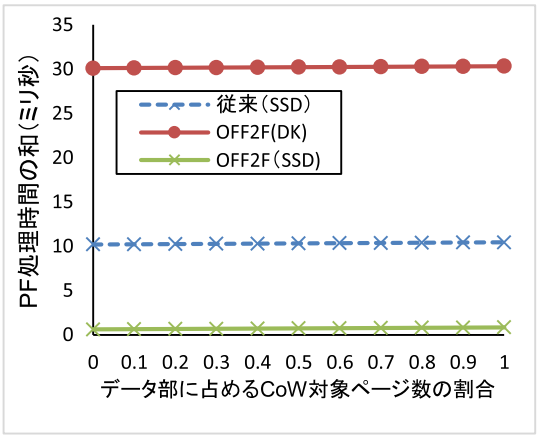
- OS：FreeBSD 6.3-R
- プロセッサ：Intel Core i7-2600（3.4 GHz）
- DK：Seagate ST500DM002（7,200 RPM）
- SSD：Intel SSD 540s Series

その結果，平均読み込み時間は，DK の場合 6.22 ミリ秒，SSD の場合 93.70 マイクロ秒であった．

そこで，外部記憶装置からデータ（1 ページ：4 KB）を読

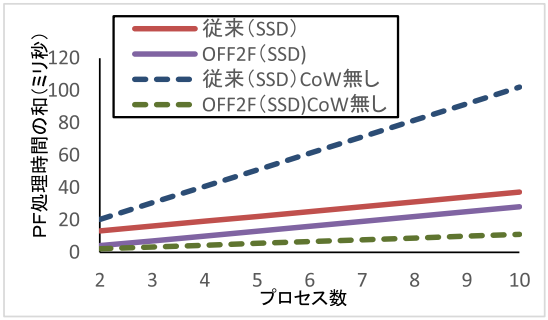


(A)データ部に占めるCoW対象ページ数の割合0.3

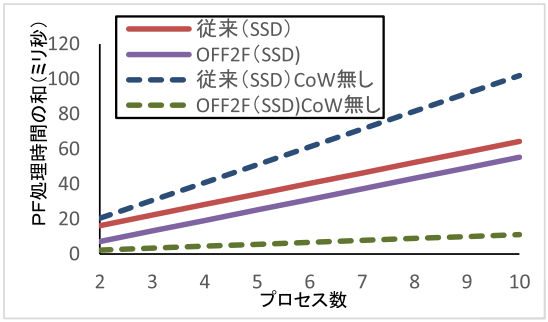


(B)プログラムに占めるテキスト部の割合0.95

図 3 プログラム全体のページ例外処理時間（プロセス数 5）
Fig. 3 Time for page fault handling of whole programs (the number of processes: 5).



(A)データ部に占めるCoW対象ページ数の割合0.3



(B)データ部に占めるCoW対象ページ数の割合0.6

図 4 プログラム全体のページ例外処理時間（プログラムに占めるテキスト部の割合 0.9）
Fig. 4 Time for page fault handling of whole programs (the text region account for program is 0.9).

み込む処理時間（t_2）として，DK は 6 ミリ秒，SSD は 0.1 ミリ秒と仮定する．つまり，$t_2 = 6\,\text{ms}$（DK），$0.1\,\text{ms}$（SSD）である．また，実測を基に，$t_5 = 0.01\,\text{ms}$，他は $t_i = 0.001\,\text{ms}$ とし，$P = 100$ ページでプロセス数 5（$n = 5$）の場合について，プログラム全体の PF 処理時間を**図 3** に示す．ここで t_i の値は，他プロセスとの競合等の影響で変動するが，OFF2F の効果検証のために静的な値として評価を行った．なお，テキスト部サイズはデータ部サイズの約 20 倍 [15] であるから，プログラムに占めるテキスト部の割合を 95%前後とした．

図 3 より，以下のことが分かる．

(1) OFF2F による PF 処理時間の短縮効果は非常に大きく，プログラムに占めるテキスト部の割合が多いほど効果は大きい．たとえば，OFF2F（SSD）は，従来（SSD）に比べ，プログラムに占めるテキスト部の割合に関係なく PF 処理時間が短い．また，OFF2F（DK）は，DK アクセス速度が SSD に比べ非常に遅いにもかかわらず，プログラムに占めるテキスト部の割合が 0.98 を超えると従来（SSD）より PF 処理時間が短い．

(2) PF 処理時間は，データ部に占める CoW 対象ページ数の割合が大きくなると増加する．しかし，その増加量は PF 処理時間の和に比べ非常に小さい．この傾向は，3 者とも同様である．したがって，データ部に占める CoW 対象ページ数の割合に関係なく，OFF2F による PF 処理時間は従来より短い．

(3) 従来（SSD）と OFF2F（SSD）を比較すると，(A)，(B) ともに式 (3) の計算結果と一致し，データ部に占める CoW 対象ページ数の割合によらず，一定の PF 処理時間短縮効果が確認できる．

また，CoW 機能でデータを共有するプロセス数と PF 処理時間の関係を**図 4** に示す．図 4 より，以下のことが分かる．

(4) (OFF2F) は，(従来) に比べ，データ部を共有するプロセス数に関係なく PF 処理時間が短い．

(5) CoW 機能を有すると，(従来) と (OFF2F) の両方において，その傾きは，(A) より (B) が大きい．これは，データ部に占める CoW 対象ページ数の割合が大きくなるため，CoW 処理が多く発生するためであり，データ部を共有するプロセス数の増加にともなう PF 処理時間の増加量は大きくなる．

(6) CoW 機能により（従来）の PF 処理時間は大幅に減少しているものの，データ部を共有するプロセス数が少ない場合には OFF2F（SSD）CoW なしの処理時間が短い．

また，OFF2F の基本性能として，NV メモリを高速読み出し可能なディスクとして実行ファイルを分割せずにそのまま格納した場合と比較する．この場合，式 (1) において t_2 の値はメモリ間のデータ複写（t_5）と同等といえる．したがって，式 (3) は，以下のようになる．

$$(t_4 - t_1 - t_2 - t_3)PS + (t_2 - t_5)P \qquad (4)$$

ここで，本節で仮定した各値を式 (4) に代入すると，DK を用いる場合は式 (4) の値が正の値になり OFF2F が 28.9 ms 低速であり，SSD を用いる場合は式 (4) の値が負の値になり OFF2F が 0.6 ms 高速である．DK を用いる場合に OFF2F が低速となるのは，データ部へのアクセス時間が大きいためである．一方，SSD を用いる場合は，データ部へのアクセス時間が小さいため，OFF2F が高速になる．さらに，OFF2F は，SSD を用いた場合に高速とできることに加えて，NV メモリの使用量を削減できる効果があり，NV メモリを高速読み出し可能なディスクとして利用する場合よりも有用であるといえる．

5.2 OS 初期化処理時間の短縮効果

5.2.1 観点

OS 初期化処理を取り上げ，OFF2F プログラムの有効性を述べる．具体的には，FreeBSD の OS 初期化処理において起動されるプログラムを明らかにし，それらのプログラムが OFF2F プログラムであるときの OS 初期化処理時間短縮効果を示す．

5.2.2 初期化処理の流れ

FreeBSD の初期化処理について，init プログラムの起動からログインプログラムの起動までを述べる．この初期化処理において，生成から終了までを行うプロセスは 589 個（pid25 から pid613）である．この様子を**図 5** に示す．pid25 は，init プログラム起動後からログインプログラム起動前まで走行している．多くの場合，pid25 のプロセスが fork() により他のプロセスを生成し，生成されたプロセスが exec() で新たなプログラムを起動する．この例として，0.735 秒から 0.739 秒までの拡大を**図 6** (A) に示す．また，pid25 以外のプロセスが fork() する例として，4.662 秒から 4.666 秒までの拡大を図 6 (B) に示す．図 6 (A) では，pid25 の sh が fork() により pid50 から pid53 までのプロセスを生成する．pid52 の sh は複製された後，exec() で swapon を起動する．図 6 (B) では，pid25 の sh が fork() により pid347 と pid351 のプロセスを生成する．また，pid347 の sh は fork() により pid348 から pid350 までのプロセスを生成する．pid348 から pid350 までの sh は複製された後，それぞれ exec() で mount を起動する．したがって，図 5 と図 6 より，以下のことが分かる．

(1) pid25 が常時走行し，fork() により多数のプロセスを生成する．

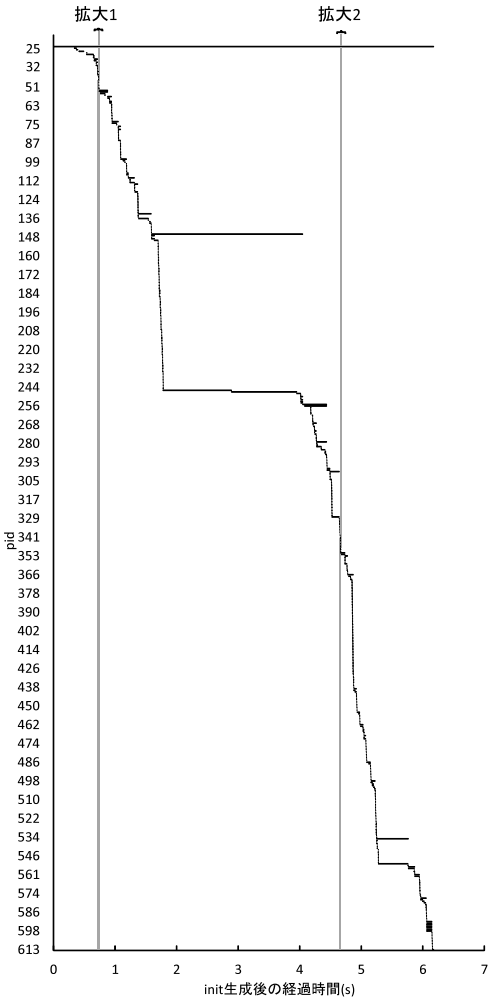


図 5 初期化処理における生成または終了するプロセス
Fig. 5 Processes created or terminated on initialization.

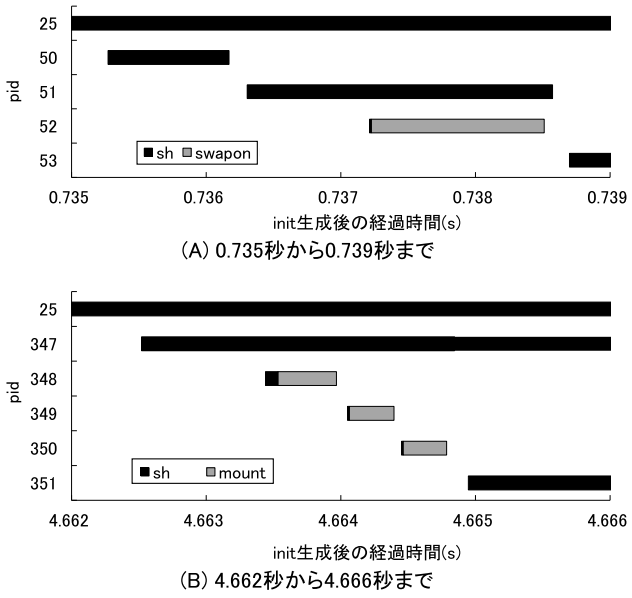


図 6 初期化処理におけるプロセスの遷移
Fig. 6 Transition of processes on initialization.

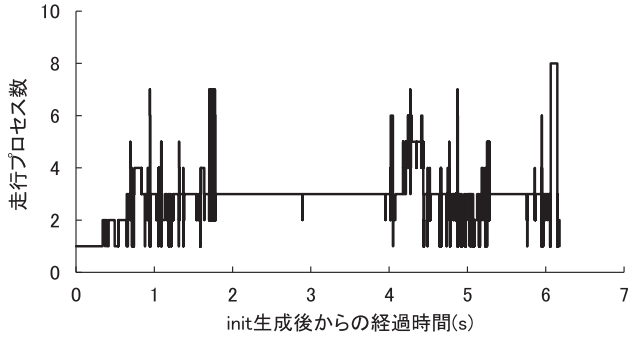


図 7 初期化処理におけるプロセス数の遷移
Fig. 7 Transition of the number of processes on initialization.

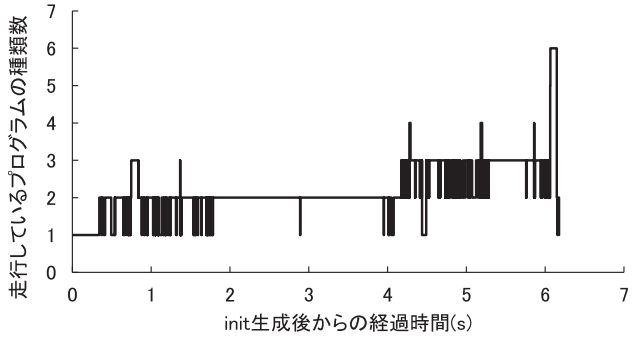


図 8 初期化処理において実行しているプログラムの種類数の遷移
Fig. 8 Transition of the number of programs on initialization.

(2) pid25 の sh から複製されたプロセスによって，sh 以外のプログラムの多くが起動される．

また，走行プロセス数を**図 7** に示し，実行しているプログラムの種類数を**図 8** に示す．図 7 と図 8 より，以下のことが分かる．

(3) 走行プロセス数と実行しているプログラムの種類数は，数個である．なお，走行プロセス数と実行しているプログラムの種類数の最低値が 1 であるのは，pid25 のプロセスが長時間走行しているためである．

(4) 走行プロセス数の増加とともに，実行しているプログラムの種類数が増加している．これは，pid25 のプロセスが fork() によりプロセス数を増加させ，その後，生成されたプロセスが exec() で sh 以外のプログラムを起動し，処理実行後にプロセス終了するためである．

次に，初期化処理で実行されるプログラムのテキスト部とデータ部の大きさ，および各プログラムが fork() する回数と exec() で起動される回数を**表 1** に示す．表 1 より，以下のことが分かる．

(1) プログラムは 61 種類である．

(2) sh は fork() する回数が多く，sh 以外の多くのプログラムは fork() しない．

(3) 多くのプログラムにおいて，テキスト部のサイズはデータ部のサイズに比べ 3 倍以上大きい．

なお，exec() でテキストを共有する場合は exec() で sh を起動する際のみであり，10 回であった．

表 1 初期化処理における実行プログラム
Table 1 Executable programs on initialization.

通番	プログラム名	テキスト部 (byte)	データ部 (byte)	fork() する回数	exec() で起動される回数	テキスト部の全 PF 数	データ部の全 PF 数
1	adjkerntz	6,073	681	0	1	2	1
2	atmconfig	56,462	1,144	0	1	14	1
3	awk	192,681	3,528	0	3	144	3
4	basename	3,757	617	0	2	2	2
5	cat	8,615	817	0	5	15	5
6	cmp	8,521	737	0	1	3	1
7	cron	37,279	1,728	0	1	10	1
8	date	16,533	1,248	0	1	5	1
9	dd	20,557	824	0	3	18	3
10	devd	1,119,770	14,160	4	1	274	20
11	devfs	12,196	801	0	55	165	55
12	dmesg	5,129	744	0	1	2	1
13	dumpon	4,133	633	0	1	2	1
14	egrep	94,331	1,356	0	2	48	2
15	expr	16,114	713	0	8	32	8
16	find	48,966	1,256	0	1	12	1
17	fsck	15,380	897	2	1	4	3
18	fsck_ufs	106,206	1,300	0	2	52	2
19	gpart	22,796	4,352	0	1	6	2
20	grep	94,331	1,356	0	1	24	1
21	hostname	2,694	593	0	1	1	1
22	id	8,744	801	0	2	6	2
23	ifconfig	154,722	27,152	0	12	456	84
24	init	1,012,833	15,075	1	0	0	4
25	ip6addrctl	7,763	761	0	2	4	2
26	kbdcontrol	34,843	900	0	7	63	7
27	kenv	4,180	641	0	3	6	3
28	ldconfig	17,799	944	0	2	10	2
29	limits	15,929	849	0	6	24	6
30	ln	6,736	729	0	2	4	2
31	logger	8,886	761	0	2	6	2
32	ls	27,928	1,232	0	1	7	1
33	mailwrapper	4,789	713	0	2	4	2
34	md5	19,723	1,720	0	1	5	1
35	mixer	8,558	689	0	3	9	3
36	mkdir	3,898	641	0	2	2	2
37	mktemp	3,882	649	0	2	2	2
38	mount	19,602	1,544	0	6	30	6
39	newsyslog	38,428	1,296	0	1	10	1
40	protect	3,395	609	0	1	1	1
41	ps	32,033	10,202	0	10	80	30
42	rcorder	8,769	737	0	2	6	2
43	realpath	2,311	569	0	1	1	1
44	rm	10,903	881	0	21	63	21
45	rmdir	2,727	585	0	2	2	2
46	route	26,438	1,025	0	7	49	7
47	savecore	15,415	985	0	1	4	1
48	sed	32,915	1,108	0	3	27	3
49	sendmail	4,789	713	2	2	4	4
50	sh	146,263	1,748	570	13	108	295
51	sleep	2,883	593	0	2	2	2
52	sshd	277,942	6,800	0	2	136	4
53	stty	18,054	1,680	0	1	5	1
54	swapon	14,569	984	0	2	8	2
55	sysctl	17,327	857	0	49	245	49
56	syslogd	37,812	1,556	1	1	10	2
57	tr	14,369	2,904	0	5	20	5
58	umount	14,390	929	0	1	4	1
59	uname	6,416	617	0	1	2	1
60	utx	3,880	625	0	1	1	1
61	vidcontrol	21,642	841	0	1	6	1
合計		4,007,009	134,130	580	278	2,267	683

表 2 実行プログラムにリンクされるライブラリ
Table 2 Libraries linked to the executable programs.

通番	ライブラリ名	テキスト部 (byte)	データ部 (byte)	リンクするプログラム数	テキスト部の全 PF 数	データ部の全 PF 数
1	lib80211.so.1	14,400	1,401	1	48	12
2	libbsdxml.so.4	151,200	7,977	2	481	26
3	libc.so.7	1,630,004	48,672	59	106,266	6,900
4	libcrypt.so.5	50,234	1,272	1	26	2
5	libcrypto.so.8	2,354,308	167,384	2	1,725	123
6	libedit.so.7	210,640	8,143	1	156	590
7	libelf.so.2	89,816	2,060	2	242	11
8	libgeom.so.5	18,139	1,056	1	5	1
9	libjail.so.1	16,955	888	2	110	22
10	libkvm.so.7	54,427	1,992	2	154	11
11	libm.so.5	169,847	1,776	3	1,050	25
12	libmd.so.6	90,148	1,344	1	23	1
13	libncursesw.so.8	355,179	17,160	2	348	1,480
14	libsbuf.so.6	7,859	689	2	26	13
15	libthr.so.3	107,906	2,964	1	54	2
16	libufs.so.6	13,791	1,480	1	8	2
17	libutil.so.9	66,358	3,008	13	629	40
18	libxo.so.0	104,896	3,171	3	312	12
19	libz.so.6	90,999	1,328	4	138	6
20	libasn1.so.11	637,392	14,720	1	312	8
21	libbsm.so.3	102,532	4,153	1	52	4
22	libbsnmp.so.6	70,847	1,776	1	18	1
23	libbz2.so.4	77,518	4,112	2	57	6
24	libcom_err.so.5	4,287	688	1	4	2
25	libgnuregex.so.5	84,145	809	2	63	3
26	libgssapi.so.10	35,831	1,552	1	18	2
27	libgssapi_krb5.so.10	118,640	5,968	1	58	4
28	libheimbase.so.11	11,572	1,336	1	6	2
29	libhx509.so.11	296,494	13,512	1	146	8
30	libkrb5.so.11	488,483	18,416	1	240	10
31	libpam.so.6	47,206	2,320	2	36	3
32	libprivateheimipcc.so.11	6,923	848	1	4	2
33	libprivateldns.so.5	358,156	21,552	1	176	12
34	libprivatessh.so.5	632,945	11,296	1	310	6
35	libroken.so.11	70,258	2,688	1	36	2
36	libwind.so.11	160,519	1,016	1	80	2
37	libwrap.so.6	31,902	2,625	1	16	2
合計		8,832,756	383,152	124	113,433	9,358

最後に，実行プログラムにリンクされているライブラリのテキスト部とデータ部の大きさ，および各ライブラリがリンクするプログラム数を**表 2** に示す．表 2 から以下のことが分かる．

(1) プログラムにリンクされたライブラリは 37 種類である．
(2) libc.so.7 が最も多くリンクされ，それ以外は数種類のプログラムにしかリンクされていない．
(3) 多くのライブラリにおいて，テキスト部のサイズはデータ部のサイズに比べ 6 倍以上大きい．

なお，実行プログラムは 61 種類であるが，ライブラリをリンクしたプログラムは 59 種類であった．つまり，2 つの実行プログラム（devd と init）はライブラリをリンクしていない．

5.2.3 ページ例外の発生回数

プログラムおよびライブラリにおいて，ページ例外が発生する回数は，プログラムの fork() 回数や exec() 回数，およびライブラリをリンクしているプログラムの fork() 回数や exec() 回数に依存する．具体的には，以下の回数である．

(1) プログラムのテキスト部：ページ数 × exec() で起動される回数
(2) プログラムのデータ部：ページ数 ×（fork() する回数と exec() で起動される回数の和）
(3) ライブラリのテキスト部：ページ数 × リンクしているプログラムが exec() で起動される回数
(4) ライブラリのデータ部：ページ数 ×（リンクしているプログラムが fork() する回数と exec() で起動される回数の和）

ここで，回数とは，そのプログラムまたはライブラリの全体が 1 度だけ実行や参照/更新されたときに発生する PF 回数（以降，全 PF 数と略す）である．なお，テキスト部については，exec() で起動される際にテキストを共有しない場合の回数である．つまり，先に述べたように，exec() でテキストを共有する場合は exec() で sh を起動する際のみであり，この場合の回数（10 回）は含めない．また，データ部については，fork() 直後に exec() する場合，fork() 直後のデータ部は CoW 機能により処理されるため PF 対象にならない．このため，この場合の回数を fork() する回数に含めない．

各プログラムの全 PF 数を表 1 に示す．各全 PF 数は，上記 (1)，(2) で算出した．なお，sh については，fork() 直後に exec() している場合を考慮し，fork() する回数は 292 回（570 − 278）とし，exec() で起動される回数は 3 回（13 − 10）として全 PF 数を算出した．また，ライブラリの全 PF 数を表 2 に示す．各全 PF 数は，各ライブラリをリンクしているプログラムを調査し，その結果と表 1 の「fork() する回数」や「exec() で起動される回数」を利用して上記 (3)，(4) に基づき算出した．たとえば，lib80211.so.1 の場合，リンクしているプログラムは ifconfig であり，ifconfig の fork() する回数は 0 であり exec() で起動される回数は 12 であることから，当該値を算出した．なお，sh は libc.so.7，libedit.so.7，libncursesw.so.8 をリンクしており，全 PF 数の算出の際には exec() でのテキスト共有や fork() & exec() での CoW 機能を考慮している．表 1 と表 2 より，プログラムの全 PF 数はテキスト部が 2,267 回，データ部が 683 回であり，ライブラリの全 PF 数はテキスト部が 113,433 回，データ部が 9,358 回である．

一方，実測した初期化処理のページ例外回数は 51,610 回であった．プロセスの仮想アドレス空間情報とページ例外の仮想アドレスにより分類すると，

- プログラムのテキスト部：45 回
- プログラムのデータ部：3,271 回
- ライブラリ：45,175 回
- その他（たとえば，スタック）：3,119 回

であった．なお，FreeBSD では，起動するプログラムのテキスト部の先頭部分（64 KB）は，PF 処理ではなく，exec() によるプロセス起動時に読み込まれる．この処理も PF 処理と同様に OFF2F の効果が期待できるため，以降ではページ例外として扱う．

5.2.4 ページ例外処理の短縮効果

ページ例外の発生回数から，OFF2F プログラム実行による PF 処理の短縮効果を述べる．

ライブラリのテキスト部とデータ部の PF 数を次の方法で見積もる．実 PF 数（45,175）をテキスト部とデータ部の全 PF 数の割合で決定する．つまり，テキスト部は 41,732（$45{,}175 \times (113{,}433/(113{,}433 + 9{,}358))$）となり，データ部は 3,443（$45{,}175 \times (9{,}358/(113{,}433 + 9{,}358))$）となる．

したがって，実測した初期化処理のページ例外回数 51,610 を以下の分類とする．

- テキスト部としての実 PF 数：41,777（45 + 41,732）
- データ部としての実 PF 数：9,833（3,271 + 3,443 + 3,119）

上記と式 (1)，(2) を利用して，PF 処理時間の和とデータ部 PF 数に占める CoW 処理の割合の関係を**図 9** に示す．5.1 節と同じ計算機環境において，FreeBSD カーネルのファイルを 4 KB 単位で連続読み込みする処理を実測したところ，DK では 0.0919 ミリ秒/4 KB，SSD では 0.01696 ミリ秒/4 KB であった．このため，DK の読み込み処理時間を 0.092 ミリ秒/4 KB，つまり $t_2 = 0.092$ ms とし，SSD の読み込み処理時間を 0.017 ミリ秒/4 KB，つまり $t_2 = 0.017$ ms とした．連続読み込みする処理を測定したのは，起動処理で用いられるプログラムは外部記憶装置の連続領域に格納されているためである．その他の t_i は 5.1 節と同じとした．図 9 より，以下のことが分かる．

(1) OFF2F による PF 処理時間の短縮効果は非常に大きい．データ部 PF 数に占める CoW 処理の割合に関係なく，DK では約 4 秒，SSD では約 750 ミリ秒短縮できる．

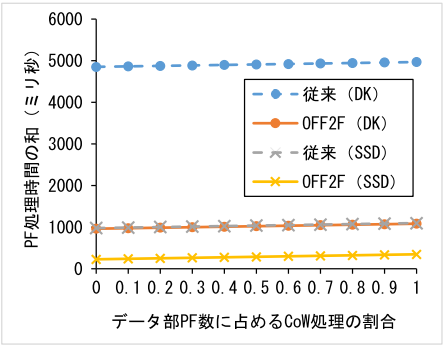


図 9 PF 処理時間の和とデータ部 PF 数に占める CoW 処理の割合
Fig. 9 Rate of CoW processing account for the sum of page fault processing time and the number of page faults on data region.

(2) PF 処理時間は，データ部 PF 数に占める CoW 処理の割合が大きくなると増加する．しかし，その増加量は，PF 処理時間の和に比べ非常に小さい．

5.3 応用プログラム実行時の処理時間の短縮効果

5.3.1 観点

応用プログラム実行時の処理時間の短縮効果を明らかにする．具体的には，データ圧縮を行う gzip コマンド実行時のページ例外発生回数をもとに，OFF2F による処理時間の短縮効果を明らかにする．

5.3.2 ページ例外の発生回数

gzip コマンドにより FreeBSD のカーネルプログラム（/boot/kernel/kernel）を圧縮した際のページ例外発生回数を**表 3** に示す．テキスト部とデータ部の PF 数は，実行プログラムのみでなくリンクされているライブラリも含む．実測したページ例外回数の合計は 268 回（9 + 259）であった．表 3 より，gzip コマンドでは gzip と sh が実行されることが分かる．

5.3.3 ページ例外処理の短縮効果

式 (1)，(2) において，t_i の値は 5.2.4 項で述べた値と同じとした．また，P はページ例外により実際にメモリ上へロードされたページ数（268）とし，S はメモリ上へロードされたページのうちテキスト部の割合（$0.03 = 9/259$）とした．上記と式 (1)，(2) を利用して，PF 処理時間の和とデータ部 PF 数に占める CoW 処理の割合の関係を**図 10** に示す．図 10 より，以下のことが分かる．

表 3 gzip コマンド実行時の PF 回数
Table 3 The number of page faults on the execution of gzip command.

通番	プログラム名	プログラムのサイズ(KB)	テキスト部 (KB)	テキスト部の全 PF 数	テキスト部以外の全 PF 数
1	gzip	37.5	32.8	9	177
2	sh	150.1	142.9	0	82
合計				9	259

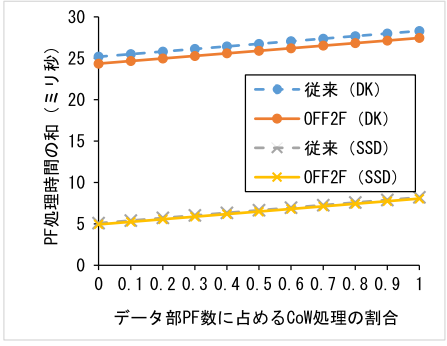


図 10 gzip コマンド実行時の PF 処理時間の和とデータ部 PF 数に占める CoW 処理の割合
Fig. 10 Rate of CoW processing account for the sum of page fault processing time and the number of page faults on data region for the execution of gzip command.

(1) プロセス数が少なく，テキスト部の割合が少ないため，OFF2F による処理時間の短縮効果は小さい．データ部 PF 数に占める CoW 処理の割合に関係なく，DK では約 0.8 ミリ秒，SSD では約 0.2 ミリ秒短縮できる．

6. まとめ

仮想記憶機構の ODP 機能の下で OFF2F プログラムを実行する際，CoW 機能を考慮したページ例外処理について述べ，その処理時間を定式化し，OFF2F プログラムを実行することによる PF 処理時間の短縮効果を明らかにした．評価により，OFF2F による PF 処理時間の短縮効果は，非常に大きく，プログラムに占めるテキスト部の割合が多いほど効果は大きいことを述べた．ただし，PF 処理時間は，データ部に占める CoW 対象ページ数の割合が大きくなると増加してしまう．しかし，この増加量は PF 処理時間に比べ非常に小さい．また，OS 初期化処理時間の短縮効果として FreeBSD を取り上げ，OFF2F プログラム実行の効果が非常に大きいことを述べた．

残された課題として，式 (1)，(2) の t_i を変化させたときの評価，および様々な応用プログラムでの評価がある．

謝辞 本研究の一部は，JSPS KAKENHI 18K11244，および共同研究（株式会社富士通研究所）による．

参考文献

[1] 谷口秀夫：分散指向永続オペレーティングシステム ***Tender***，情報処理学会コンピュータシステムシンポジウム論文集，Vol.95, No.7, pp.47–54 (1995).
[2] 谷口秀夫，市川正也：***Tender*** オペレーティングシステムにおける資源の永続化機構，情報処理学会研究報告，Vol.1999, No.32, pp.7–12 (1999).
[3] Yamauchi, T., Yamamoto, Y., Nagai, K., Matono, T., Inamoto, S., Ichikawa, M., Goto, M. and Taniguchi, H.: Plate: Persistent Memory Management for Nonvolatile Main Memory, *Proc. 31st ACM Symposium on Applied Computing* (*SAC 2016*), pp.1885–1892 (2016).
[4] 追川修一：Non-Volatile メインメモリとファイルシステムの融合，情報処理学会論文誌，Vol.54, No.3, pp.1153–1164 (2013).
[5] Oe, K., Sato, M. and Nanri, T.: Automated Tiered Storage System Consisting of Memory and Flash Storage to Improve Response Time with Input-Output (IO) Concentration Workloads, *2017 5th International Symposium on Computing and Networking* (*CANDAR*), pp.311–317 (2017).
[6] 追川修一：ブロックストレージとの組み合わせによるメモリストレージ容量拡張手法，情報処理学会論文誌コンピューティングシステム，Vol.8, No.2, pp.15–24 (2015).
[7] Wei, Q., Chen, J. and Chen, C.: Accelerating File System Metadata Access with Byte-Addressable Nonvolatile Memory, *ACM Trans. Storage* (*TOS*), Vol.11 (2015).
[8] Lee, E., Kim, J., Bahn, H., Lee, S. and Noh, S.H.: Reducing Write Amplification of Flash Storage through Cooperative Data Management with NVM, *ACM Trans. Storage* (*TOS*) – *Special Issue on MSST 2016*, Vol.13 (2017).
[9] Poremba, M., Akgun, I., Yin, J., Kayiran, O., Xie, Y. and Loh, G.H.: There and Back Again: Optimizing the Interconnect in Networks of Memory Cubes, *Proc. 44th Annual International Symposium on Computer Architecture* (2017).
[10] Yoon, D.H., Gonzalez, T., Ranganathan, P. and Schreiber, R.S.: Exploring latency-power tradeoffs in deep nonvolatile memory hierarchies, *Proc. 9th Conference on Computing Frontiers* (2012).
[11] Zhang, Y. and Swanson, S.: A study of Application Performance with Non-Volatile Main Memory, *2015 31st Symposium on Mass Storage Systems and Technologies* (*MSST*), pp.1–10 (2015).
[12] Guo, X., Shrivastava, A., Spear, M. and Tan, G.: Languages Must Expose Memory Heterogeneity, *Proc. 2nd International Symposium on Memory Systems* (2016).
[13] Xu, J. and Swanson, S.: NOVA: A Log-structured File System for Hybrid Volatile/Non-volatile Main Memories, *14th USENIX Conference on File and Storage Technologies* (*FAST 16*), pp.323–338, USENIX Association (2016) (online), available from ⟨https://www.usenix.org/conference/fast16/technical-sessions/presentation/xu⟩.
[14] Dulloor, S.R., Roy, A., Zhao, Z., Sundaram, N., Satish, N., Sankaran, R., Jackson, J. and Schwan, K.: Data Tiering in Heterogeneous Memory Systems, *Proc. Eleventh European Conference on Computer Systems*, *EuroSys '16*, pp.15:1–15:16, ACM (online), DOI: 10.1145/2901318.2901344 (2016).
[15] Sato, M. and Taniguchi, H.: OFF2F: A New Object File Format for Virtual Memory Systems to Support Volatile/Non-volatile Memory-Mixed Environment, *International Journal of Machine Learning and Computing*, Vol.9, No.4, pp.387–392 (2019).

推薦文

情報処理学会中国支部表彰規定に則り，平成 30 年度（第 69 回）電気・情報関連学会中国支部連合大会で発表された中から，特に優秀であることが認められた優秀論文発表賞を授賞した論文である．

（情報処理学会中国支部支部長　金田和文）

谷口 秀夫 （正会員）

1978 年九州大学工学部電子工学科卒業．1980 年同大学大学院修士課程修了．同年日本電信電話公社電気通信研究所入所．1987 年同所主任研究員．1988 年 NTT データ通信株式会社開発本部移籍．1992 年同本部主幹技師．1993 年九州大学工学部助教授．2003 年岡山大学工学部教授．2010 年同大学工学部長．2014 年同大学理事・副学長．博士（工学）．オペレーティングシステム，実時間処理，分散処理に興味を持つ．著書『並列分散処理』（コロナ社）等．電子情報通信学会，ACM 各会員．本会フェロー．

佐藤 将也 （正会員）

2010 年岡山大学工学情報工学科卒業．2012 年同大学大学院自然科学研究科博士前期課程修了．2014 年同大学同研究科博士後期課程修了．2013 年日本学術振興会特別研究員（DC2）．現在，岡山大学大学院自然科学研究科助教．博士（工学）．コンピュータセキュリティ，仮想化技術に興味を持つ．2012 年度情報処理学会論文賞受賞．電子情報通信学会会員．

河辺 誠弥 （正会員）

2017 年岡山大学工学部情報系学科卒業．2019 年同大学大学院自然科学研究科博士前期課程修了．

横山 和俊 （正会員）

1988 年広島大学工学部第二類（電気系）卒業．1990 年同大学大学院工学研究科博士課程前期修了．2006 年岡山大学大学院自然科学研究科博士後期課程修了．1990 年 NTT データ通信株式会社（現，株式会社 NTT データ）入社後，オペレーティングシステム，分散処理の研究開発に従事．2012 年高知工科大情報学群教授．博士（工学）．電子情報通信学会会員．