Vol.2021-ARC-244 No.10
Vol.2021-SLDM-194 No.10
Vol.2021-EMB-56 No.10
2021/3/25

アプリケーションからのストレージアクセス分析手法

長谷川博紀[1]　松原豊[1]　加藤寿和[2]　山本整[2]　高田広章[1]

概要：近年，車載情報機器のような組込み機器においても，スマートフォンのように機器メーカが開発する基本機能に加え，第三者が開発したアプリケーションを導入可能とすることが検討されている．複数のアプリケーションを組込み機器に統合する場合，追加されたアプリケーションによる基本機能への悪影響を防止したいという要求がある．アプリケーション間での影響伝搬を防止するパーティショニング技術の研究では，CPU やメモリを対象とする技術は数多く存在するが，ストレージデバイスに対する研究は少ない．本論文では，複数アプリケーションが共有するストレージデバイスに対するパーティショニング技術の確立に向けて，アプリケーションごとのストレージデバイスへのアクセスを記録し，分析する手法を提案する．Linux カーネルベースの ftrace と，ブロック I/O スケジューラベースの blktrace を併用することで，プロセスごとの I/O アクセスをトレースし可視化する．動画再生とナビゲーションの２つのアプリケーションを対象に，それぞれのストレージデバイスへのアクセスの頻度や帯域，アクセス時間の内訳を詳細に分析できることを確認した．

1. はじめに

近年，車載情報機器を初めとする高性能組込み機器においても，スマートフォンのように機器メーカが開発する基本機能に加え，第三者が開発したアプリケーションを導入可能とすることが検討されている．複数のアプリケーションを組込み機器に統合して動作させる場合，追加されたアプリケーションによる基本機能への悪影響を防止したいという要求がある．アプリケーション間での影響伝搬を防止するパーティショニング技術の研究では，CPU やメモリを対象とする技術は数多く存在するが，ストレージデバイスに対する研究は少ない [1][2].

本論文では，複数アプリケーションが共有するストレージデバイスを対象として，アプリケーションの独立性とリアルタイム性の両立が可能なパーティショニング技術の確立に向けて，アプリケーションごとのストレージデバイスへのアクセスを記録（トレース）し，分析する手法を提案する．ストレージデバイスへのアクセスをトレースするツールとして，２つのツールの採用を検討した．具体的には，ストレージデバイスへのアクセスをトレースするツールとして，Linux カーネルベースの ftrace と，ブロック I/O スケジューラベースの blktrace を併用することで，プロセスごとの I/O アクセス状況を可視化し分析する．提案手法の評価のために，車載情報機器での利用が想定される動画再生とナビゲーションを対象に，ストレージへのアクセス特性を分析している．評価対象のストレージには，高性能・省電力であり，さまざまな組込み機器で搭載実績のある eMMC を利用した．

表 1　eMMC v5.1 の基本性能

最大容量	256GB
最高転送速度	400MB/s
接続方式	MMC カードスロット（シリアルバス）
ブロックサイズ	512 bytes
パッケージサイズ	SD card と同等（規定なし）

2. Linux I/O システム

2.1 eMMC

eMMC（embedded Multi Media Card）は JEDEC によって規定された規格である．組込み機器への搭載を前提に，MMC（Multi Media Card）を省電力化・小型化したものである．現在，eMMC の最新バージョンは，v5.1 である．基本性能を表 1 に示す．近年のモバイル機器の要求に対応するため，v5.0 から最大スループットが 400MB/s となり，SSD に近い性能が出るようになっている [3].

Linux において，eMMC は SD カードと同様の MMC サブシステムでサポートされている．MMC ドライバは，ハードウェアの初期化処理や接続されているデバイスを認識して，SD カードと eMMC を判断したり，ファイルシステム

[1] 名古屋大学 大学院情報学研究科
[2] 三菱電機株式会社 情報技術総合研究所

Vol.2021-ARC-244 No.10
Vol.2021-SLDM-194 No.10
Vol.2021-EMB-56 No.10
2021/3/25

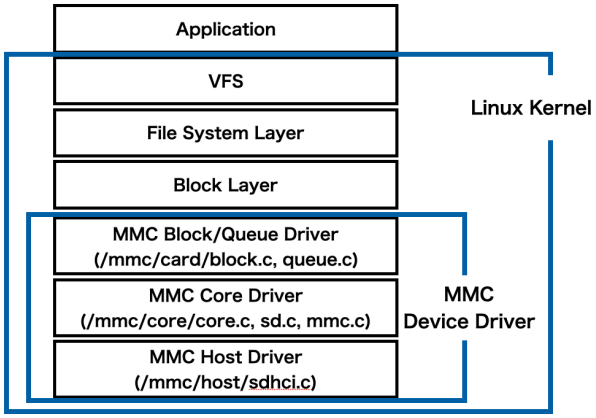

図 1 Linux における MMC デバイスドライバの位置付け

や上位層からのデータ転送要求の処理を担当している [4]. MMC デバイスドライバの構成を図 1 に示す.

MMC Block/Queue Driver

キュードライバは, 上位の Block Layer から受けた要求を，キューで管理する．ブロックドライバはキューの先頭から取り出した要求のタイプ (read，write，erase，など) を解析し，更に下の MMC Core Driver に要求を渡す.

MMC Core Driver

コアドライバは，上位から受け取った要求の対象が SD カードか eMMC かを判別し，SD カード/eMMC 処理部に要求を渡す．SD カードと eMMC ではカードの認識やスピードモードの設定に基づいて，下位の MMC Host Driver 内の SD ホストコントローラ共通処理部や機種依存部を呼び出し，ホストコントローラを制御する.

MMC Host Driver

ホストドライバは，受け取った要求をハードウェアコントローラのレジスタを使用してデバイスへ送信する．デバイスから完了割り込みがあれば，そこにエラーがあるか解析し，エラーがあれば Block Layer に I/O の完了割り込みをブロックレイヤーに送信する.

2.2 ext4 ファイルシステム

Linux では，ファイルシステムに ext4 が使用されている．ext4 の大きな特徴は，エクステントとジャーナリングシステムである.

2.2.1 エクステント

エクステントとは，従来のブロックマッピング方式に置き換わるデータ参照方法である．大きなファイルを記憶する際に，１つ１つのブロック位置を記憶するのではなく，先頭ブロックと連続ブロック数（オフセット）を一度に記憶することで効率化する．ext3 までは，バッファ書き込み時にディスク上のブロックを割り当てるが，ext4 ではディスク書き込み時にブロックを割り当てる．ext4 では複数の書き込みをまとめてブロックを割り当てるため，連続したブロックを割り当てやすくなり，結果として，フラグメンテーションが起こりにくくなっている.

2.2.2 jbd2

jbd2 は，ext4 に使用されているジャーナリングシステムである．初めにメタデータの更新内容をジャーナルログに書き込み，完了した後にディスクに書き込む．ディスクへの書き込みが完了した後，ジャーナルログを破棄する．もし，書き込み処理が途中でクラッシュした場合に，ジャーナルログを辿ってデータを再生してディスクに書き込むことができる．書き込みを保証するレベルは journal，order，writeback の 3 つがあり，この順番で保証レベルが高い．それぞれの保証レベルでの動作を以下に示す.

journal

全てのデータとメタデータをジャーナルに書き込む

order

ファイルシステムのメタデータのみジャーナルに書き込む

writeback

メタデータとデータの書き出し順が保証されない

Linux における，ジャーナリングシステムのデフォルト保証レベルは order である.

2.3 ブロックレイヤー

ブロックレイヤーでは, I/O スケジューラがデバイスへの読み書きを効率化するために，要求を並び替える．現在の Linux カーネルの I/O スケジューラは, none, mq-deadline, kyber，bfq の 4 種類である．概要を以下に示す.

noop

I/O 要求を要求順に処理する．スケジューリング負荷が小さい.

mq-deadline

デッドラインを設定し，もしデッドラインミスが起きそうな I/O があれば，その I/O 要求を優先する.

kyber

読み込みと同期書き込みに対してレイテンシを設定し，そのレイテンシに間に合うように並び替える.

bfq

デバイスによるレイテンシが少ないように，要求を並び替える.

Ubuntu20.04 では，noop と mq-deadline は初めから使用できるが，kyber と bfq を使用するには，カーネルコンフィグレーションし，再ビルドする必要がある．Ubuntu20.04 におけるデフォルトの I/O スケジューラは mq-deadline である.

Vol.2021-ARC-244 No.10
Vol.2021-SLDM-194 No.10
Vol.2021-EMB-56 No.10
2021/3/25

3. トレースツール

3.1 ftrace

ftrace は，Linux カーネルをベースとしたトレースツールである．Linux カーネルに組み込まれており，カーネルのイベントを時系列でログとして記録・確認できる．カーネル領域における関数呼び出しを記録できるため，アプリケーションが実際にどのような処理を呼び出しているかだけでなく，タイムスタンプも記録するため，I/O においてカーネル内のどのレイヤーにおける処理負荷が高いかも知ることができる [5]．トレース対象が定まっているのであれば，カーネルイベントのトレース対象を絞り込むことで，詳細な分析が可能である．

3.2 blktrace

blktrace は，ブロックレイヤーでの処理を詳細にトレースするツールである．ただし，トレースする対象を指定することはできず，実行中の全てのプロセスの実行を記録する．blktrace パッケージには，トレース結果のバイナリファイルを可読化する blkparse や，情報を整理して出力する btt コマンドも含まれる．Linux のブロックレイヤーは，図 2 のようになっており，I/O スケジューラ内の処理が表 2 と表 3 のようにイニシャル文字で表示される．トレース結果には，デバイス ID，CPU 番号，シーケンス番号，タイムスタンプ，PID，Action，RWBS（Read Write Barrier Synchronous），セクタ番号，プロセス名が含まれる．

btt を使用することで，図 2 右側に示す各区間の平均処理時間や最小処理時間，最大処理時間，I/O 数を整理できる．各区間の説明は以下のとおりである．

Q2Q
　要求がブロック層に送信される間隔

Q2G
　ブロック I/O キューに挿入されてから，要求が割り当てられるまで

G2I
　要求が割り当てられてから，リクエストキューに挿入されるまで

I2D
　リクエストキューに挿入されてから，デバイスに発行されるまで

Q2M
　ブロック I/O キューに挿入されてから，要求がマージされるまで

M2D
　要求がマージされてから，デバイスに発行されるまで

D2C
　デバイスによるサービス時間

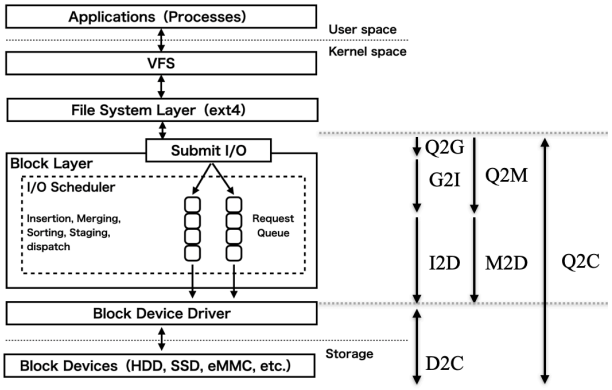

図 2　Linux ブロックレイヤー

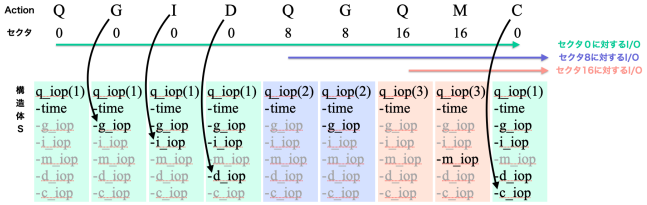

図 3　btt における処理時間計算方法

Q2C
　ブロックレイヤー全体で消費した総時間

これらの区間の測定方法は，I/O の始まりである Q において，各 Action の情報を保管できる構造体 S を新しく作成する．その後，各 Action が発行されれば，G，I，M に関してはセクタ番号が一致している構造体 S に保管する．その際に，計算できる区間は計算する．例えば，図 3 に示す I/O があった場合，Q の Action で新しい構造体を作成し，Q のタイムスタンプを保管する．次に G の Action が発生した時に，セクタが同じ Q が保管されている構造体内の g_iop に G のタイムスタンプを保管する．Q と G のタイムスタンプの差をとることで Q2G の処理時間を計算し，最小処理時間，平均処理時間，最大処理時間，I/O 数を更新する．D，C に関しては，構造体 S のリストで先頭にあるものが対応していると判断している．図 3 では，リストの先頭にある q_iop(1) に保管してあるデータを使用して，計算可能な区間の処理時間を計算する．そして，C の計算が終了した時に先頭の構造体を削除する．

アプリケーション毎にブロックレイヤーでの処理特性が異なるので，blktrace を用いることで，単体動作だけでなく，複数のアプリケーションを同時実行した際のアプリケーションのアクセス特性の分析に役立つ [6]．加えて，B オプションによって，I/O 毎のブロック数，セクタ番号，トレース開始からの経過時間を集めた dat ファイルを作成し，bno_plot で可視化することもできる．

4. 提案手法

ストレージアクセスにおいて，どの部分がボトルネック

表 2 Actions（blktrace）

C	complete
D	issued（dispatch）
I	inserted
Q	queued
B	bounced
M	back merge
F	front merge
G	get request
S	sleep
P	plug
U	unplug
T	unplug due to timer
X	split
A	remap
m	message

表 3 RWBS（blktrace）

R	Read
W	Write
M	Meta-data
S	Synchronous
A	Read-Ahead
F	Flush or Force Unit Access
D	Discard
E	Erase
N	No data

となっているのかを分析するために，カーネル空間全体をトレースできるという観点で ftrace，アプリケーション毎のアクセス特性が現れやすいブロックレイヤーを詳細にトレースできるという観点で blktrace を利用した．

4.1 レイヤー毎での分割

ftrace は，カーネル空間における関数呼び出しを記録できるため，広範囲かつ詳細に分析できる．レイヤー毎の処理時間を計測するために，レイヤー間の移動となる関数を目印に，レイヤー毎の実行時間を計測した．計測するレイヤーの分割は，ファイルシステム，ブロックレイヤー，デバイスドライバ，ハードウェア，ハードウェア割り込み（I/O の完了）である．レイヤー間の境界と定義した関数を表 5 に示す．

4.2 アプリケーション毎の分割

blktrace は，btt コマンドを併用することでブロックレイヤーにおける区間毎の実行時間を整理して出力する．しかし，実行中のプロセスを全てトレースしてまとめたアクセス特性を出力するため，アプリケーション毎のアクセス特性を分析することができない．そこで，対象アプリケーションに限定してアクセス特性を分析するために，btt のソースコードを修正する．

対象アプリケーションのプロセス名による I/O のみを抽出することで, 分離することを目的とした．プロセス名での分離では十分でない部分として, D（issue）と C（complete）の Action を実行するプロセスがあり，D は kworker プロセス，C は pid 0（swapper または sched）のプロセスによって実行されている．これによって，対象アプリケーションのプロセスのみを抽出することが難しくなっている．既存の btt の計算方法では順番通りに I/O Action が発生することが前提となっており，単純に I や M Action の次に発生する D，C Action を含めるようにすると，他プロセスの I/O に対する D，C を対応付けてしまう．このような誤った I/O を btt での計算に含めず，対象アプリケーションによる I/O のみを計算に使用するよう変更した．

対象アプリケーションによる I/O に対応する D，C のみを計算に加えるために，D，C が実行されたセクタ番号と I/O ブロックサイズを利用した．I/O ブロックサイズを利用した理由は，マージ操作が発生した場合，Q と D，C のセクタ番号が一致しないが，対応している I/O であり，考慮する必要があるためである．マージされる場合，D，C のセクタ番号はマージされる I/O の先頭のセクタ番号である．マージされる Q のセクタ番号は，I/O ブロックサイズの分だけ増加していくため，これを計算対象に含めるようにすればよい．これを考慮すると，計算する Q に対応する D，C は I/O の式 (1) を満たすものになる．

$$\begin{aligned}&\text{セクタ } (D,C)\\ &\quad \leq \text{セクタ } (Q)\\ &\qquad < \text{セクタ } (D,C)+I/O \text{ ブロックサイズ } (D,C)\end{aligned} \quad \cdots (1)$$

セクタ (Q) は，D，C と未計算の I/O の Q のセクタ番号であり，セクタ (D,C) は，全ての I/O による D と C のセクタ番号である．D，C のセクタ番号と I/O ブロックサイズから，式 (1) を満たす対象プロセスの I/O による Q が含まれていれば，その D，C が対象アプリケーションによる I/O に対応しているため，計算に含める．結果として，各 Action の I/O 数がログデータと一致するようになり，全ての I/O をトレースできていることを確認した．

表 4 評価環境

CPU	Intel Pentium (4cores)
OS	Ubuntu 20.04 LTS
RAM	8GB
Storage	eMMC(64GB)

5. ストレージアクセス分析

5.1 分析対象

I/O アクセス特性を分析し，どこで負荷が大きいかを明らかにするために，カーネル空間におけるレイヤーという観点で ftrace を，アプリケーション毎という観点で blktrace と改良版 btt を利用した．評価環境を表 4 に示す．また，jbd2 の設定は order であり，I/O スケジューラの設定は mq-deadline である．btt では，実行時間以外にも呼び出し関係や I/O スケジューラにおける要求のマージ数などが記録できるが，アクセス特性による影響が顕著に現れる実行時間のみを分析対象にした．

表 5　レイヤー間の関数呼び出し

From	To	Function
User App	File System	ksys_write
File System	Block Layer	generic_make_request
Block Layer	Device Driver	blk_mq_sched_insert_requests
Device Driver	Hardware	blk_mq_run_hw_queue
Hardware	IRQ	mmc_cqe_request_done
IRQ	Finish	bio_endio

5.2　ftrace による 1 回の I/O のアクセス分析

5.2.1　分析方法

ftrace を操作するインターフェースとして，trace-cmd コマンドがある．トレースするためには

```
# trace-cmd record -p function_graph -F (Command)
```

のように実行する．p オプションでトレーサの指定ができ，function_graph の設定では関数の呼び出しと戻るタイムスタンプを記録する．F オプションで指定したコマンドのみをトレースできる．今回の評価で使用したコマンドは，

```
# trace-cmd record -p function_graph
                    -F (Write / Read Program)
```

である．ファイル書き込みのプログラムでは fopen と fprintf を，ファイル読み込みのプログラムでは fopen と fgets を使用した．

5.2.2　分析結果

分析結果を図 4 に示す．表 5 に示した関数をレイヤーの境界として，ファイルシステム，ブロックレイヤー＋デバイスドライバ，デバイス，I/O の完了割り込みの処理時間を計測し，グラフ化した．キャッシュへのマッピングやファイルマッピングの処理をするファイルシステムレイヤーでの処理が最も長い理由として，プログラム実行前にキャッシュを全て削除してから実行しているため，全てのマッピング処理が必要なことも影響していると考えられる．次に処理時間が長いのはデバイスであり，eMMC の処理時間がボトルネックとなることがわかった．

5.3　動画再生アプリの分析

5.3.1　分析方法

動画再生のアプリケーションとして，Linux で主流である VLC メディアプレイヤーを使用して mp4 形式の動画を再生した．blktrace による分析をするために，以下の (1)〜(4) のコマンドを使用した．

(1) blktrace -d /dev/mmcblk0 (-w Time[s])
　コア毎に mmcblk0 への I/O をトレースし，mmcblk0 の出力ファイルを生成

(2) blkparse -i mmcblk0 -d video.bin
　トレース結果のバイナリファイル mmcblk0 を読めるように変換

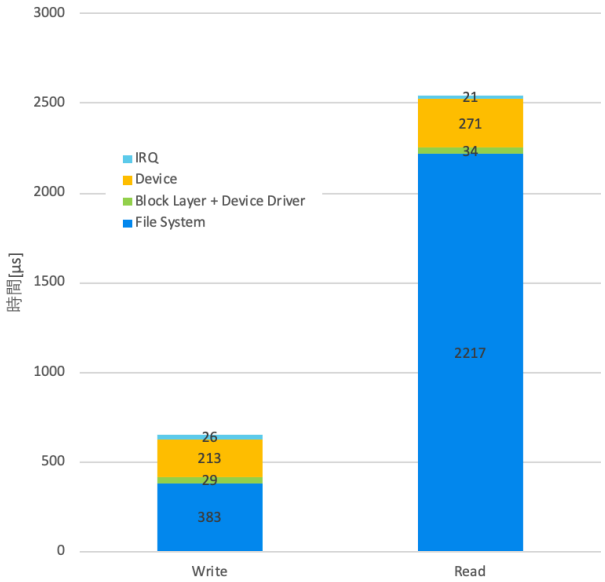

図 4　Write と Read 操作の実行時間の内訳

　コア毎のトレース結果を集約して video.bin を生成

(3) btt -i video.bin -B Video -N vlc
　video.bin からデータを集計
　Video_r.dat，Video_w.dat，Video_c.dat に I/O 結果を出力
　N オプションは，今回コードに追加したものであり，対象プロセスのみを測定対象にする

(4) bno_plot Video_r.dat Video_w.dat
　Video_r.dat と Video_w.dat にある I/O データを可視化

キャッシュにデータが残っていると，ストレージへのアクセスがうまく記録できないため，blktrace 実行前にキャッシュを削除した．加えて，マルチコアからの I/O が並列に発生して，正しくトレースできなくなる可能性を下げるために，taskset コマンドでデバイスの割り込みコアに，対象のプロセスを固定させて動作させた．これにより，割り込みが他のコアで実行されている間にプロセスが進むことがなくなるため，btt での集計の誤動作を抑制できる．動画再生時の I/O を分析するために，2 通りの方法で計測した．

1 つ目の計測は，1 つの動画を再生して I/O をトレースした．VLC を予め起動しておき再生ボタンを押す直前の状態にしてから，blktrace を 1 分半の間動作させ，その間に動画を再生し，トレース終了まで再生し続けた．

2 つ目の計測は，VLC による I/O と動画再生による I/O を分離するために，プレイリストを作成して複数の動画を連続再生した時の I/O をトレースした．今回の評価では，2 分半〜3 分半のそれぞれ異なる 5 つの動画（720p，30fps）からプレイリストを作成した．blktrace は，プレイリスト全体を含むように 15 分間実行した．

Vol.2021-ARC-244 No.10
Vol.2021-SLDM-194 No.10
Vol.2021-EMB-56 No.10
2021/3/25

表 6　動画再生時の I/O 時間

	平均時間 [μs]	I/O 数 [N]	割合 [%]
Q2Q	1387.4953	60	
Q2G	0.0083	61	0.02
G2I	0.0040	61	0.01
I2D	28.9116	61	83.80
D2C	5.5765	61	16.16
Q2C	34.5004	61	

5.3.2　分析結果

1 つの動画を再生した時の分析結果を表 6 に示す．動画の連続データに対するマージ操作とデバイスへの負荷がかかっていることが確認できる．bno_plot による 1 回毎の I/O をプロットした結果を図 5 に示す．動画再生処理では，基本的に書き込みが発生しないので，書き込みによるプロットデータは存在しないか，VLC 本体による書き込みが少し見られる程度と予想したが，この測定では見られなかった．連続するブロック番号への同サイズの連続的な読み込みがあることから，動画の読み込みだと考えられる．

プレイリストにして動画を再生した時の分析結果を，表 7 と図 6 に示す．図 6 の右側の連続した読み込みでは，切れ目が 4 つあり，動画読み込みによる I/O であると考えられる．切れ目で書き込みが発生することもわかった．動画再生の結果と比較すると，I/O 毎のブロックサイズが一定ではなかった．ランダム読み込みは一時的にしか見られず，VLC 自体の読み込みであると考えられる．

どちらの分析結果においても，I2D>D2C であり，デバイスよりも I/O Scheduler での時間が長かった．このことから，デバイスでの処理が長く，I2D でデバイスへの発行待ちキューに要求が長期間置かれていると考えられる．ftrace でトレースした 1 回の読み込み処理の計測結果では，ブロックレイヤーよりもデバイスでの負荷が高かったことからも，I/O スケジューラでの処理は終了しており，キュー待ちになっていることが考えられる．また，プレイリスト再生時にブロックサイズが一定でないことを考えると，I/O Scheduler の mq-deadline の設定によって，デッドラインをミスする直前にデバイスに発行する状況が頻繁に発生していると考えられる．Q2G と G2I の負荷は小さいため，この部分におけるキューの中で処理待ちとなっている I/O はほとんどなく，ボトルネックとはなっていない．

5.4　ナビゲーションアプリの分析

5.4.1　分析方法

ナビゲーションのアプリケーションとしては，ローカルに保存したマップを使用する Navit を使用した．Navit とは，オープンソースマップを使用可能な，無料のオープンソースカーナビゲーションシステムである．blktrace によるトレースは，動画再生の時と同様のコマンドを使用した．また，動画再生と同様にキャッシュの削除とコアの固定を

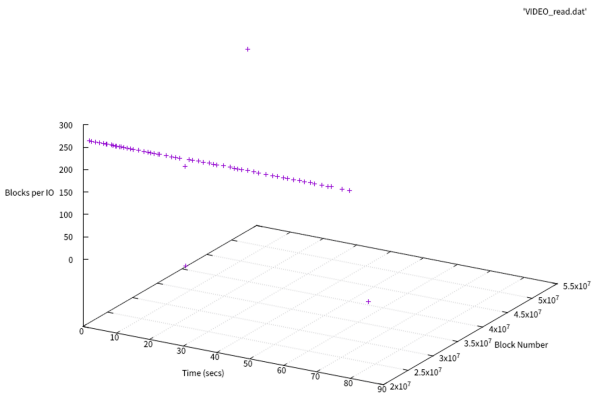

図 5　動画再生時の I/O 分布

表 7　プレイリスト再生時の I/O 時間

	平均時間 [μs]	I/O 数 [N]	割合 [%]
Q2Q	245.9818	3526	
Q2G	0.0064	2882	0.0215
G2I	0.0040	2863	0.0133
Q2M	0.0004	645	0.0003
I2D	25.3428	2864	84.4377
M2D	2.1123	644	1.5825
D2C	3.3980	3527	13.9424
Q2C	24.3717	3527	

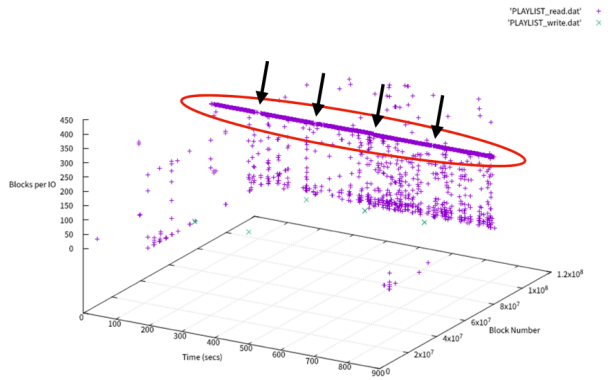

図 6　プレイリスト再生時のの I/O 分布

した．

ナビゲーションにおける大きな I/O 負荷が発生するのは地図の読み込みであるため，これを定量的に測定するために，機械的なマウス操作を実行できる xdotool を使用した．xdotool によって，0.5 秒間隔で（10，-1）座標だけスワイプ動作をさせることで Navit の地図を動かし，地図の読み込みが常に発生し続けるようにすることで，車両の移動による地図の読み込みを擬似的に発生させた．Navit が起動している状態から blktrace によってトレースを開始し，xdotool を起動させ負荷をかけ，この状態の I/O を 3 分間トレースした．

5.4.2　分析結果

分析結果を，表 8 と図 7 に示す．動画再生と同様に，I2D と M2D の平均時間が D2C の平均時間よりも長く，デバ

Vol.2021-ARC-244 No.10
Vol.2021-SLDM-194 No.10
Vol.2021-EMB-56 No.10
2021/3/25

表 8　Navit 実行時の I/O 時間

	平均時間 [μs]	I/O 数 [N]	割合 [%]
Q2Q	483.3316	359	
Q2G	0.0038	147	0.013
G2I	0.0073	140	0.023
Q2M	0.0003	213	0.001
I2D	3.1749	140	10.336
M2D	12.8400	213	63.602
D2C	2.8574	360	23.922
Q2C	11.9445	360	

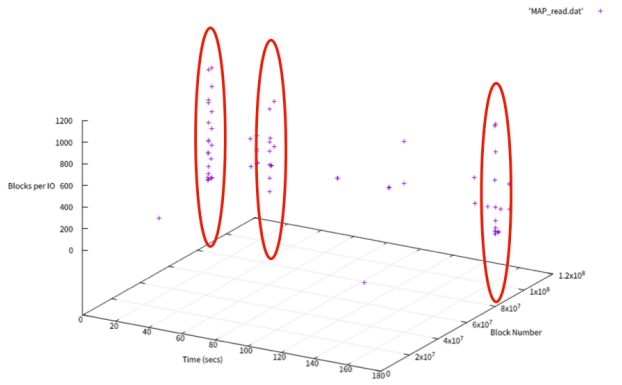

図 7　Navit 実行時の I/O 分布

イス操作がボトルネックになっていることがわかる．I/O 分布においては，地図のスワイプが 0.5 秒周期で発生するため，動画再生と同様に時間に連続した I/O 分布が見られると予想していたが，実際にはあるタイミングで大きな I/O がまとめて発生し，しばらくは I/O が発生しないという結果を得た．分析開始前にキャッシュを削除しているため，計測前に地図データを読み込んでいることはない．したがって，Navit または I/O システムによって，必要な分よりも大きく地図データを読み込んでいると考えられる．xdotool による 0.5 秒間隔の移動量を（300,-200）座標にして計測した結果において，より短い時間の間隔で I/O の大きな読み込みが発生することも確認した．

6. おわりに

複数アプリケーションが共有するストレージデバイスに対するパーティショニング技術の確立に向けて，アプリケーションごとのストレージデバイスへのアクセスを記録し，分析する手法を提案した．ftrace と blktrace を使用して，実際にアプリケーションのストレージのアクセス特性について分析した．結果として，動画再生による連続読み込みが実行中常に一定量で発生すること，ナビゲーションでの地図読み込みは大きな読み込みによって必要範囲以上に読み込んでおり，I/O は時間に連続ではないことがわかった．このように，単一のアプリケーションによる I/O のみを抽出して，ストレージアクセス特性を正確に分析できた．

今後，複数のアプリケーションを同時に動作させた場合のアクセス特性の変化についても分析し，どのようにパーティショニングをするかを検討していく．

参考文献

[1] Yuan, W. and Nahrstedt, K.: Energy-Efficient CPU Scheduling for Multimedia Applications, *ACM Trans. Comput. Syst.*, Vol. 24, No. 3, p. 292–331 (online), DOI: 10.1145/1151690.1151693 (2006).
[2] Shan, Y., Tsai, S.-Y. and Zhang, Y.: Distributed Shared Persistent Memory, *Proceedings of the 2017 Symposium on Cloud Computing*, SoCC '17, New York, NY, USA, Association for Computing Machinery, p. 323–337 (online), DOI: 10.1145/3127479.3128610 (2017).
[3] : eMMC Electrical Standard v5.1 JESD84-B51(online), https://www.jedec.org/sites/default/files/docs/JESD84-B51.pdf. (Accessed on 12/12/2020).
[4] : Linux Kernel MMC Storage driver Overview(online), https://www.slideshare.net/rampalliraj/linux-kernel-mmc-storage-driver-overview. (Accessed on 12/12/2020).
[5] : Linux Storage System Analysis for e.MMC With Command Queuing(online), https://www.micron.com/-/media/client/global/documents/products/white-paper/linux_storage_system_analysis_emmc_command_queuing.pdf. (Accessed on 12/12/2020).
[6] Zhou, D., Pan, W., Wang, W. and Xie, T.: I/O Characteristics of Smartphone Applications and Their Implications for EMMC Design, *Proceedings of the 2015 IEEE International Symposium on Workload Characterization*, IISWC '15, USA, IEEE Computer Society, p. 12–21 (online), DOI: 10.1109/IISWC.2015.8 (2015).