C-020

フラッシュストレージ搭載 Android 端末における I/O スケジューリング
A Performance Study on I/O Scheduling in Android Terminals using Flash Memory Storage

中村 優太†　永田 恭輔†　野村 駿†　山口 実靖†
Yuta Nakamura　Kyosuke Nagata　Shun Nomura　Saneyasu Yamaguchi

１．はじめに

近年，スマートフォンやタブレット PC が普及し，それらの携帯端末で動作するソフトウェアプラットフォームである Android OS が注目されている．スマートフォンにおける Android OS の世界シェアは 2012 年第 2 四半期には 68%となり[1]，Android OS は非常に重要なプラットフォームの 1 つになっている．

Android OS は Linux OS をベースとしており，Android OS 搭載端末は従来の携帯電話とは異なり RDBMS などの高度なデータ処理も可能となっている．よって，今後はスマートフォンやタブレット PC において I/O 量の多いアプリケーションが実行される状況も生じると予想され，Android プラットフォームにおける I/O 性能の向上は重要な課題の一つと考えられる．

I/O の最適化や性能向上は主に I/O スケジューラにて行われるが，これまでの I/O スケジューラの研究は HDD を対象したもの[2]や，仮想化環境を対象としたもの[3][4]がほとんどである．フラッシュストレージにおいては，スケジューラによる並び替えを行わない NOOP が適しているとされ[3][5]，フラッシュストレージを対象とした I/O スケジューリングの研究はほとんどなされていない．

本稿では，最初にフラッシュメモリ搭載 Android 携帯端末における I/O 基本性能の評価結果と，I/O スケジューリングの効果の検証結果を示し，フラッシュメモリ搭載 Android 端末においても I/O スケジューリングに効果があることを示す．そして，フラッシュメモリ搭載 Android 端末の I/O スケジューラの性能評価と解析を行い，スケジューラの変更による I/O 性能が向上可能であることと，Android OS の動作解析に基づく性能向上の理由を示す．

本論文の構成は以下の通りである．2 章で Android OS と，そのカーネルおよび I/O スケジューラについて紹介する．3 章ではフラッシュメモリ搭載 Android 端末の I/O 基本性能評価と I/O スケジューリングの効果の検証を行う．4 章では，I/O スケジューラの評価を示す．5 章で関連研究を紹介する．6 章で考察を行い，第 7 章にてまとめを述べる．

２．Android OS

2.1　Android OS の構成

Android OS はアプリケーション，アプリケーションフレームワーク，ライブラリ，Android ランタイム，Linux カーネルで構成されている[6].

† 工学院大学大学院工学研究科電気電子専攻, Electrical Engineering and Electronics, Kogakuin University Graduate School

I/O スケジューラは Linux カーネル内に実装されており，後述の様に Android OS では Linux カーネルの I/O スケジューラの実装をほぼ改変せずに使用している．

ただし，HDD 搭載の Linux 計算機では HDD は SATA 規格のものも含めて SCSI サブシステムを用いて制御されているが，Android スマートフォンではフラッシュストレージは MMC ドライバにより制御されている．

2.2　I/O スケジューラ

I/O スケジューラはプロセス群から発行される I/O 要求群を適切な順に並び替え，高性能や高公平性などを実現する OS 内のソフトウェアである．新しい I/O 要求が追加されたときに I/O スケジューラが呼び出され，新しい I/O 要求をキューのどの位置に追加するかを決定する．HDD 搭載の計算機においては，I/O スケジューラが I/O 性能に大きな影響を与えることが確認されており[2][5]，Linux には NOOP, DEADLINE, CFQ の 3 種類の I/O スケジューラが搭載されている．各 I/O スケジューラの特徴は以下の通りである．

NOOP はスケジューリングにおける並び替えは行わずに単純に I/O 要求を到着順に処理する I/O スケジューラである．スケジューリングによる負荷が小さく，SAN ストレージなど下位層において高度なスケジューリングが行われる環境[3]や，フラッシュメモリなどのディスク型でないストレージに適している[5][7]とされている．DEADLINE は現在処理している I/O 要求の近隣の I/O 要求を優先して処理することで HDD ヘッドの移動の削減を行うスケジューラである．これにより近隣ではない I/O 要求は保留されてしまうが，保留時間には限界値(deadline)が用意されており限界値より長く保留された I/O 要求は優先的に処理される．レイテンシの上限が保障されているため，リアルタイムアプリケーションやデータベース管理システムなどに適しているとされている．CFQ は，プロセスから発行される I/O 要求をプロセス毎のキューに割り振っていくスケジューラであり，処理対象のキューを切り替えながら I/O 要求を処理していく．処理対象キューを一定時間で変更するため，I/O 要求はプロセス単位で公平に処理される．CFQ にも DEADLINE と同様に待ち時間の長い I/O 要求を優先的に処理する機能が備わっている．Linux カーネルにおいては，初期状態では CFQ が I/O スケジューラとして選択されている．

2.3　Android OS における I/O スケジューラ

Android OS は Linux カーネルを用いているため，I/O スケジューラは Linux カーネル内に実装されているものが使用可能となっている．具体的には，前述の NOOP，Deadline，CFQ が使用可能である．

Copyright © 2013 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.

表 1 測定端末の仕様

端末名	使用 OS	CPU	メモリ	FS
NexusS	4.0.3	CPU cortexA8 (Hummingbird) Processor 1GHz	512MB	VFAT
Nexus7	4.1.2	NVIDIA® Tegra® 3 mobile processor 1.3GHz	1GB	FUSE

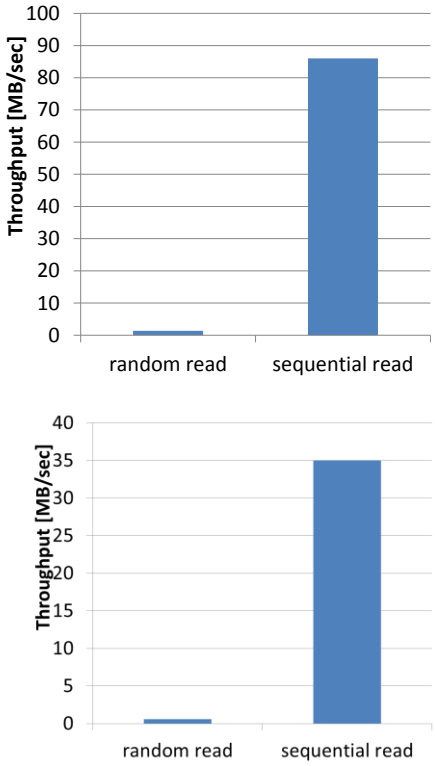

図 1　ランダムリードとシーケンシャルリード性能
(上段 NexusS，下段 Nexus7)

I/O スケジューラの実装に関しては，Android OS は Linux OS の実装をほぼ無変更のまま使用しており，初期状態にて選択されている I/O スケジューラも Linux 同様に CFQ となっている．

Linux OS と Android OS の I/O スケジューラの実装の差異は以下の通りである．I/O スケジューラ群は，カーネルソースコードの/block/ディレクトリ内に実装されており，Linux 3.1.10 およびこれを用いた Android においては，同ディレクトリ内に 32 個のファイルが存在している．このうち 31 個のファイルにおいては完全に内容が一致し，1 個ファイル(/block/genhd.c)においてのみ差異が存在している．同ファイルは 1,817 行あり，差異は 5 行となっている．このことから，両 OS の I/O スケジューラの実装に大きな差がないと考えることができる．

3. I/O 基本性能評価

本章において，フラッシュメモリ搭載 Android 端末における I/O 基本性能の評価結果を示す．具体的には，シーケンシャルリードとランダムリードの性能測定結果と，I/O スケジューリングの効果検証を示す．

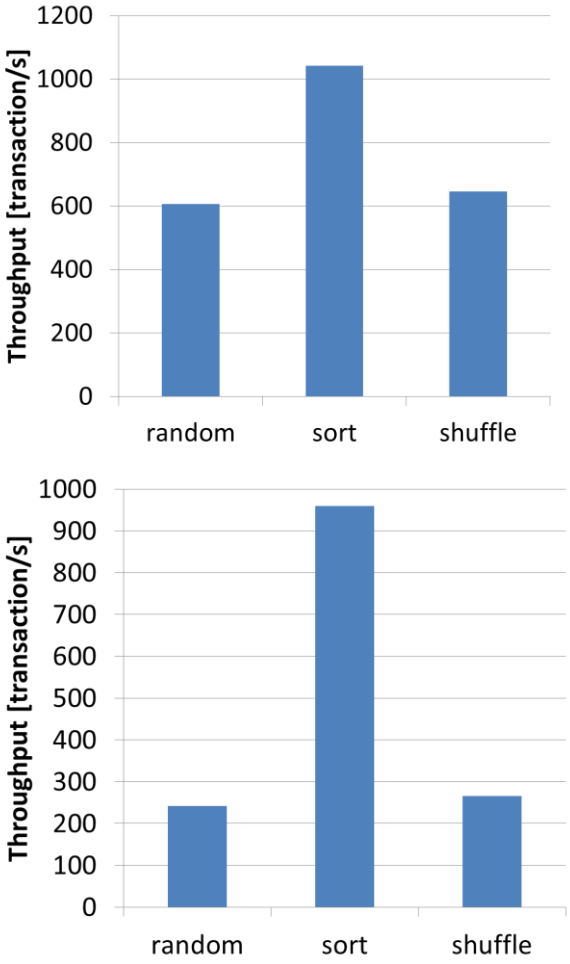

図 2　I/O スケジューリングの効果検証
(上段 NexusS，下段 Nexus7)

3.1　基本性能評価

フラッシュメモリ搭載 Android 端末のシーケンシャルリード，ランダムリードの性能を図 1 に示す．測定に使用した端末の仕様は表 1 の通りである．測定はフラッシュメモリ内に 1GB のファイルを作成し，それに 4KB の I/O 要求を 1,000 回発生させ所要時間を計測することにより行った．I/O スケジューラは CFQ を用い，シングルスレッドで行った．一般にフラッシュメモリはランダムアクセスを高速に処理できるとされているが，図 1 より Android OS を搭載した端末ではシーケンシャルリード性能がランダムリード性能を大幅に上回っていることが分かる．この差には OS の先読み処理による影響も含まれていると考えられる．

3.2　I/O スケジューリングの効果検証

フラッシュメモリ搭載の Android 端末におけるスケジューリングの効果の有無を調査するために，ソート済みアドレスへの I/O 要求と，非ソートアドレスへの I/O 要求の性能を比較する実験を行った．測定に使用した端末は表 1 の

Copyright © 2013 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.

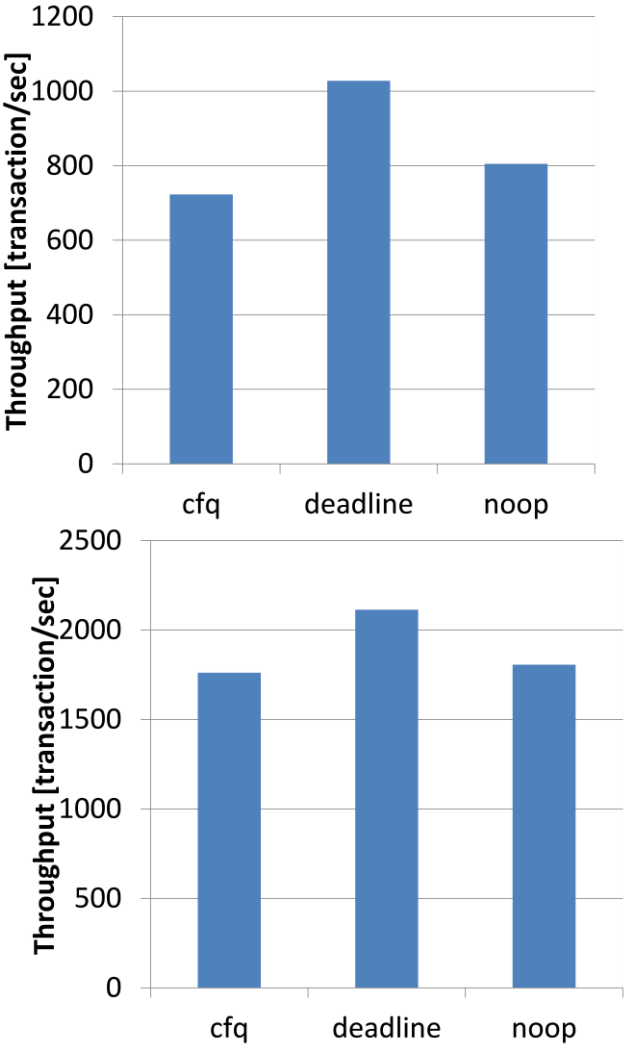

図 3　マイクロベンチによる I/O スケジューラの性能評価
(上段 NexusS，下段 Nexus7)

通りである．測定は，1GB のファイルに対して，1 バイトの read 要求を 1,024 回発行することにより行った．

“sort”と“shuffle”では，1GB のファイルを 1MB のブロック 1024 個に分割し，各ブロックの先頭の 1 バイトに対して read 要求を行った．この処理では，各読み込みアドレスは最低でも 1MB は離れることになり，OS による先読み処理の効果はあらわれない．“sort”では先頭ブロックから後方ブロックに対してアドレスが昇順になるように読み込みを行い，“shuffle”ではランダム順に並び替えて読み込みを行った．“rand”ではブロック分割をせず 1GB ファイル内のすべてのバイトから対象を決定し読み込みを行った．測定結果を図 2 に示す．図より，“sort”が”shuffle”よりも性能が高いことがわかる．同じアドレスへの I/O 要求群を異なる順で処理した“sort”と“shuffle”では性能が異なっていることより，フラッシュメモリ搭載 Android 端末においても I/O スケジューリングにより I/O 性能が変化することが分かった．

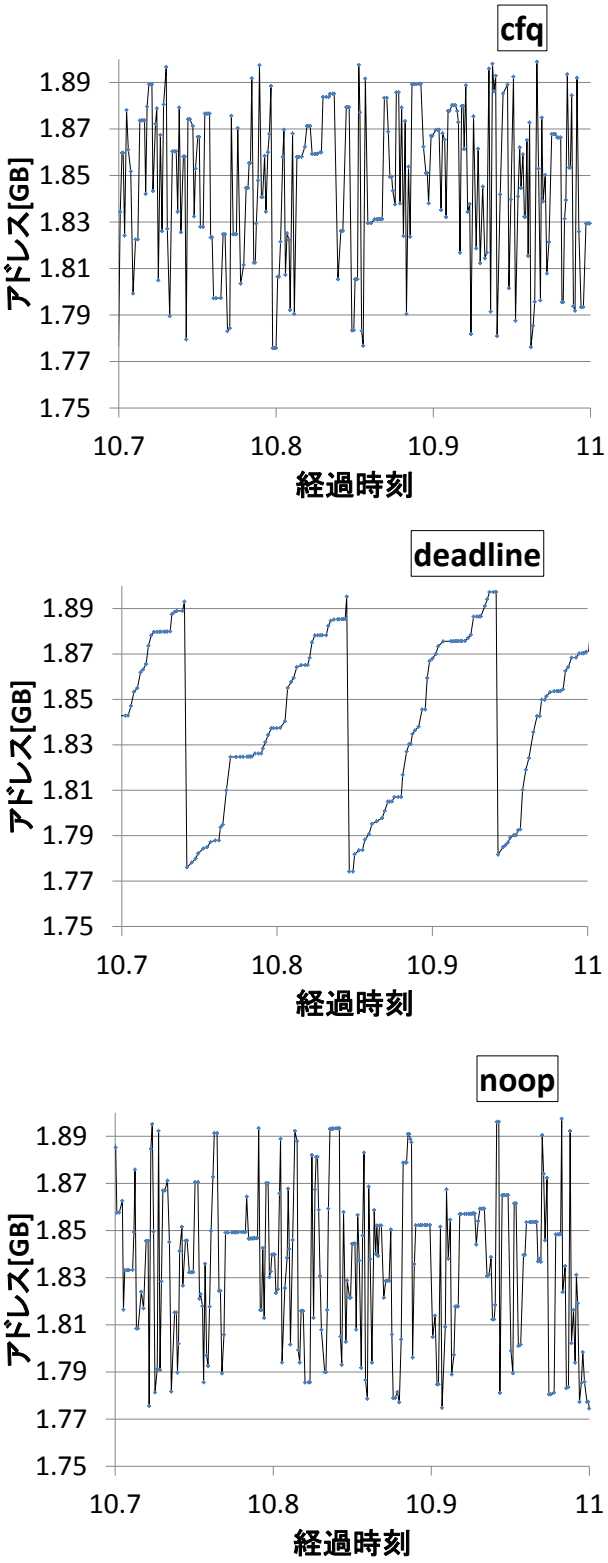

図 4　I/O スケジューリングの様子

4. I/O スケジューラの性能評価

本章にて，Android OS の I/O スケジューラの性能評価結果を示し，I/O スケジューラの変更により I/O 性能の向上が可能であることを示す．そして，Android のカーネルの動作

Copyright © 2013 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.

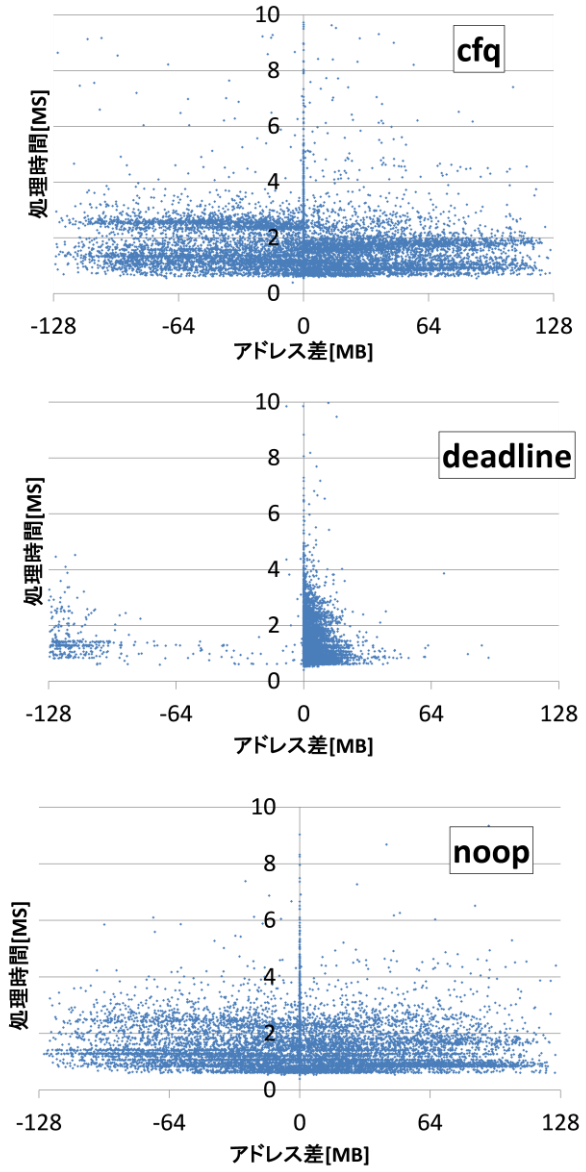

図 5　MMC ドライバ観測結果(NexusS)

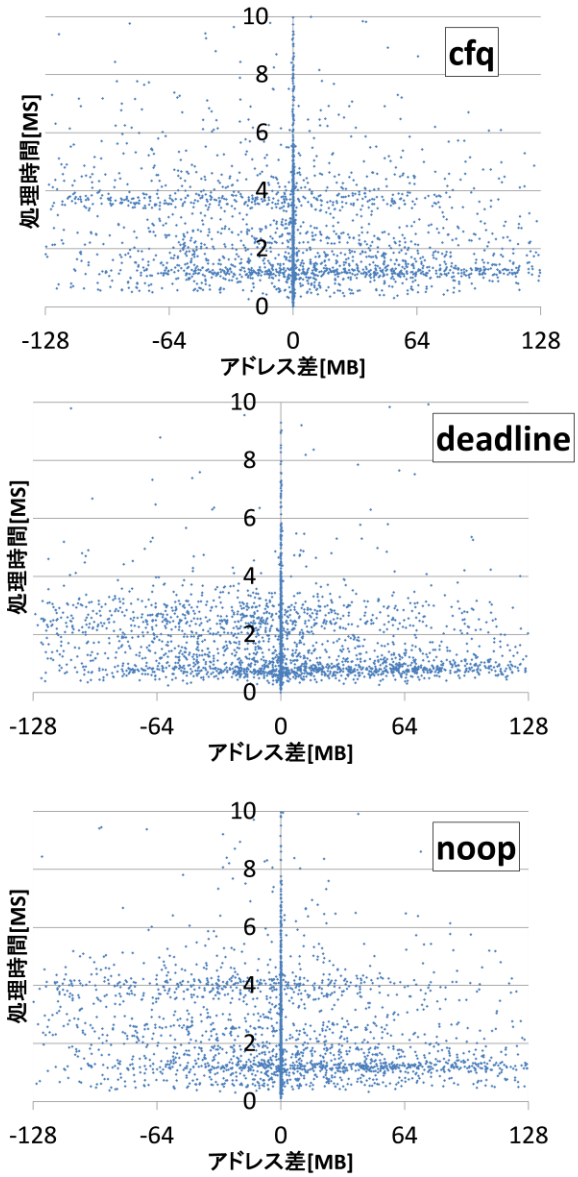

図 6　MMC ドライバの観測結果(Nexus7)

解析結果を示し，I/O 性能が向上する理由を示す．

4.1　マイクロベンチマークによる性能評価

まず，マイクロベンチマークを用いた評価と考察を示す．I/O スケジューラごとの性能を図 3 に示す．測定に使用した端末は表 1 の通りである．性能は，前章の読み込み対象ファイルの 128MB のアドレス範囲からランダム選んだ 1 バイトに対して読み込みを行う実験を行い測定した．スレッド数は 16 である．本測定では Android アプリケーションランタイムの影響を排除するために，ベンチマークプログラムは C 言語で実装し，実行はコマンドラインシェル上にて行った．

図 3 より，フラッシュメモリ搭載 Android 端末においても I/O スケジューラによる性能の違いがあり，DEADLINE が NOOP や初期設定の CFQ より高性能であることがわかる．

この結果を解析するために，MMC ドライバにて命令が発行された時刻，読み込み先アドレス[8][9]を調査した．

図 4 に MMC ドライバにおける I/O 要求発行時刻と，I/O アドレスの関係を I/O スケジューラごとに示す．また，図 5 と図 6 に観測された I/O 要求のアドレス差(今回のアクセスアドレスと前回のアクセスアドレスの差)と，処理時間の関係を示す．図 4 より，DEADLINE においては下位アドレスから上位アドレスに移動しながらアクセスを行うことが多く，CFQ, NOOP においてはそのような特徴がないことが分かる．また，図 5,図 6 よりアドレスの差が正である(アドレス増加方向に移動する)場合はアドレスの差が負であった場合よりも処理時間が短いことがわかる．また，DEADLINE はアドレスの差が正方向となる場合が多く，結果として性能が高くなったと考えられる．

Copyright © 2013 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.

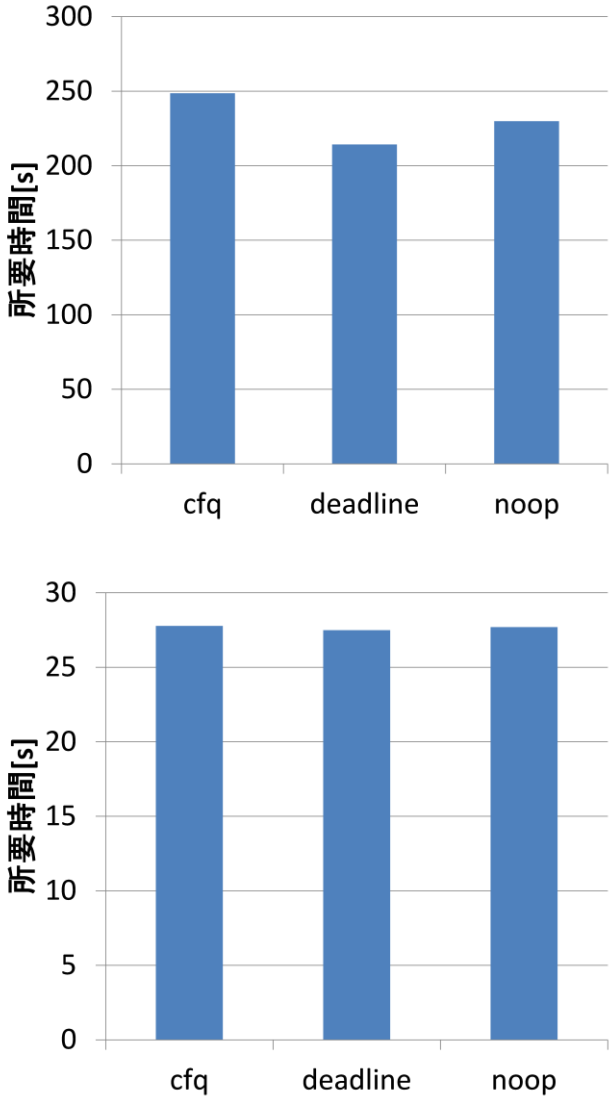

図 7　アプリベンチによる I/O スケジューラの性能比較
(上段 NexusS，下段 Nexus7)

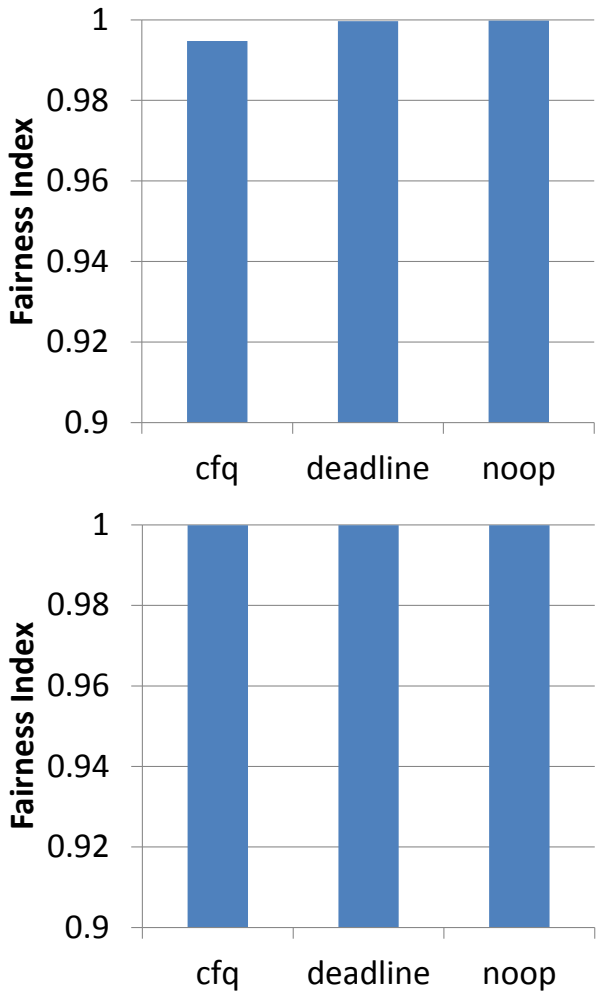

図 8　マイクロベンチマーク結果の公平性評価
(上段 NexusS，下段 Nexus7)

4.2　アプリベンチマークによる性能評価

次に，アプリケーションベンチマークを用いた評価と考察を示す．図 7 に，アプリケーションベンチマークによるマルチスレッド環境下における I/O スケジューラの性能評価を示す．測定に使用した端末は表 1 の通りである．ベンチマークプログラムは Android アプリケーションとして Java 言語で実装され，16 スレッドで 1GB の読み込み対象ファイルのランダムに選んだアドレスに対する 1 バイトの読み込みを 10,000 回行い，処理に要した時間を測定した．図 7 より，Nexus S においては DEADLINE が CFQ，NOOP よりも性能が優れることが分かり，Nexus 7 においては全 I/O スケジューラともにほぼ同程度の性能であることが分かる．以上より，フラッシュメモリ搭載の Android 端末においても多くの場合で I/O スケジューラが I/O 性能に影響を与え，DEADLINE が初期設定である CFQ の同等以上の性能を出せることが分かった．

4.3　公平性評価

本節にて，各 I/O スケジューラの公平性の評価を行う．公平性の評価は下記の Fairness Index を用いて行う．

$$\text{Fairness Index} = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n \cdot \sum_{i=1}^{n} {x_i}^2}$$

Fairness Index は 1 に近ければ公平性が高く，0 に近ければ公平性は低い．n にはスレッド数 16 を用い，x_0 から x_{15} には 4.1 節および 4.2 節の実験にて各スレッドが得た性能を用いた．

求めた Fairness Index を図 8,図 9 に示す．これらの結果から，DEADLINE スケジューラの公平性は，初期設定で使用されている CFQ の公平性と同等あるいはそれ以上であることが分かる．このことから，DEADLINE スケジューラを選択することにより I/O 性能を向上させた場合も，公平性の低下がないことが分かる．

Copyright © 2013 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.

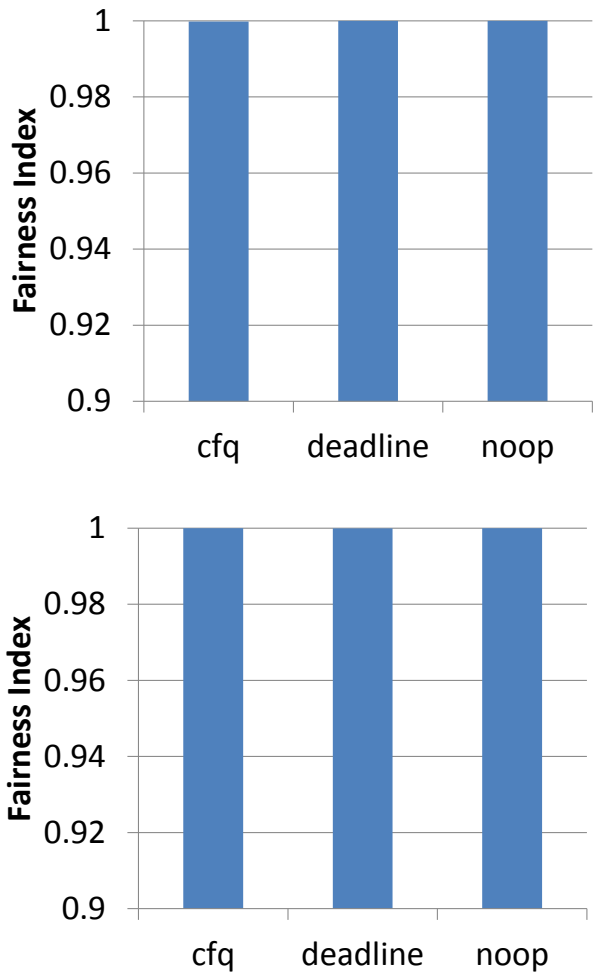

図 9　アプリベンチマーク結果の公平性評価
(上段 NexusS，下段 Nexus7)

5. 関連研究

I/O スケジューラの提案に関する既存の研究としては，Iyer らや Andrew による Anticipatory スケジューラの提案[10][11]がある．I/O スケジューラの性能に関する研究としては，Pratt らによる性能評価[12] や，Axboe による性能評価[13] がある．また，2 階層の I/O スケジューリングを考慮した研究として Cello[14] が，自己学習型（Self-Learning） ディスクスケジューラの研究として Zhang らによる研究[15] がある．

また，仮想化環境における I/O スケジューラの研究として，Boutcher の性能評価[16]，Kesavan らの仮想ディスクの振る舞いと通常の物理ディスクの振る舞いの違いの調査[17]，Xu らのプロセスの I/O 要求の局所性を利用した新しいスケジューラの提案[18]，IBM による巨大ストレージ仮想計算機環境における I/O スケジューラのチューニングに関する報告[19]などがある．

しかし，これらはフラッシュメモリや SSD を考慮したものではなく，近年普及しているスマートフォンにおける I/O スケジューラの考察に必ずしも有用ではない．また，カーネル空間における I/O スケジューラの動作解析結果が示されておらず，本稿ほど詳細な考察が示されていない．

SSD の振る舞いを考慮した研究として，Wang らの評価[20][21]がある．これらでは，SSD の振る舞いやファイルシステム，アプリケーションの振る舞いを考慮して性能に関する深い考察を行っているが，トランザクション処理を主たる目標としており，スマートフォンの I/O 性能について考察する本研究とは目的が異なる．

Android 端末の I/O 性能に関する研究としては，以下のものがある．文献[22]において，Android OS における演算性能と I/O 性能の評価と解析がされており，現状では多くの例において Linux が Android よりも性能において優れていることが確認されている．また，文献[8]において，Android 端末上における RDBMS の問い合わせ処理性能が示され，そのボトルネックの発見方法の提案と性能向上手法に関する考察が行われている．しかし，これはら汎用的 I/O 性能の向上の考察として有益であるが，I/O スケジューリングに特化しておらずスケジューリングに関する考察はなされていない．

Linux カーネルをモニタリングするための既存研究として，カーネルモニタリングツールの FTrace[23][24]，SystemTap[25][26]，LTTng[27][28]，OProfile[29]がある．これらのツールを用いることにより，Linux カーネルの内部処理の観察が可能になる．しかし，それらは Linux 用に構築されているため MMC ドライバの解析などの Android の解析には適していない．また，我々の提案手法[8][9]は，次章で述べるようにオーバーヘッドが非常に小さく，多くの既存の手法よりこの点において優れている．

6. 考察

6.1 フラッシュメモリコントローラ

本稿では，フラッシュストレージにおける I/O 要求のアドレス差と応答時間の関係に着目し，I/O 性能の向上手法についての考察を行った．HDD の場合は，この両者の関係は物理的構造により決定され，ほぼすべての製品において同様になると期待される．これに対してフラッシュメモリの場合は，両者の関係はフラッシュメモリのコントローラの実装にも大きく依存していると予想される．しかし，4 章で述べたように，2 台の端末においてほぼ同等の結果が得られているため，現状や近い将来の多くのスマートフォンやタブレット端末においては本稿における考察は有効であると考えられる．今後，フラッシュメモリコントローラに大きな変更が加えられた場合は，それらの端末に対して再度本稿と同様の考察を行う必要があると考えられるが，その様な場合でも本稿で用いた手法(アドレスソートの有無による性能の比較，MMC ドライバ層における観察)は有効であると考えている．

今後はフラッシュメモリコントローラにより多くの最適化が施され，さらに複雑な振る舞いをするようになることも考えられ，I/O スケジューラの最適化の重要性はますます高まっていくと期待される．

Copyright © 2013 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.

6.2 カーネルモニタリングのオーバーヘッド

本稿の MMC ドライバ層における I/O 要求の観測の方法と，そのオーバーヘッドについて述べる．MMC ドライバに履歴保持機能を追加して，観測を行った．履歴保持機能では，OS 起動時に履歴保存用のメモリをカーネル空間に確保する．そして，MMC 層の I/O コマンドが発行されたときとコマンドが終了したときに，時刻，I/O 種類(read/write)，アクセスアドレス，アクセスサイズをメモリにコピーする．そして，計測終了後にメモリに確保されたデータを/proc インターフェイスを通してユーザ空間に転送する[8].

履歴保存時に発生する処理は，データのメモリへのコピーのみであるため，実行時オーバーヘッドは非常に小さいと期待される．このオーバーヘッドを計測するため，履歴保存 1,000 回に要する時間を計測したところ 850 マイクロ秒であり，モニタリング 1 回あたりのオーバーヘッドは約 0.85 マイクロ秒であった．これはフラッシュメモリアクセスにより生じる遅延時間と比較して十分に小さく，十分に正確にモニタリングできていると考えることができる．

7. まとめ

本論文では，フラッシュメモリ搭載 Android 端末における I/O の基本性能の評価を行い，フラッシュメモリ搭載 Android 端末においてシーケンシャルリード性能がランダムリード性能よりも極めて高い性能であることを示した．次に，同じアドレス群をソートして読み込みを行った場合と，非ソートで読み込みを行った場合では，ソートをして読み込みを行った場合が高い性能を示し，フラッシュメモリを搭載した Android 端末においても I/O スケジューリングの効果があることを示した．そして，マイクロベンチマークとアプリケーションベンチマークを用いて I/O スケジューラの性能評価を行い，初期選択の CFQ よりも DEADLINE の方が性能が高いことを示した．また，カーネル内部にて発行された I/O 要求とその処理時間の調査を行い，I/O 要求のアドレス差が正方向である場合の方が負方向である場合よりも応答時間が短いことが性能差の原因であることを示した．

今後はフラッシュメモリ搭載 Android 端末に適したスケジューリングシステムについて考察していく予定である．

謝辞

本研究は JSPS 科研費 24300034 の助成を受けたものである．

参考文献

[1] Android and iOS Surge to New Smartphone OS Record in Second Quarter
http://www.idc.com/getdoc.jsp?containerId=prUS23638712
[2] S. Iyer and P. Druschel, “Anticipatory scheduling: A disk scheduling framework to overcome deceptive idleness in synchronous I/O.” Proc of ACM SOSP, 2001.
[3] Dave Boutcher and Abhishek Chandra,“Does Virtualization Make Disk Scheduling Passe?,” ACM SIGOPS Operating Systems Review archive Volume 44 Issue 1,2010 , pp.20-24
[4] 新居健一,山口実靖,“I/O スケジューラの改善による仮想計算機環境における I/O 性能の向上”, WebDB Forum 2011
[5] Steven L. Pratt, Dominique A. Heger, “Workload Dependent Performance Evaluation of the Linux 2.6 I/O Schedulers,” Ottawa Linux Symposium Proceedings ,2004
[6] What is Android?|Android Developers: http://developer.android.com/guide/basics/what-is-android.html
[7] Tuning I/O Performance, http://doc.opensuse.org/products/draft/SLES/SLES-tuning_sd_draft/cha.tuning.io.html
[8] 服部拓也 新居健一 山口実靖,“Android 端末 におけるデータベースのアクセス性能に関する考察”,分散,協調とモバイル(DICOMO2012)シンポジウム,pp.91-96(2012)
[9] 三木香央理,山口実靖,小口正人,“Android 端末におけるカーネルモニタの導入”,第 22 回コンピュータシステム・シンポジウム,2010
[10] S. Iyer and P. Druschel,“Anticipatory scheduling: A disk scheduling framework to overcome deceptive idleness in synchronous I/O,” SOSP’01:Proceedings of the eighteenth ACM symposium on operationg systems principles. ACM, 2001.
[11] Andrew Morton,“Linux: Anticipatory I/O Scheduler,” http://kerneltrap.org/node/567
[12] S. Pratt and D. Heger,“Workload Dependent Performance Evaluation of the Linux 2.6 I/O Schedulers,” In Proceedings of the Linux Symposium, 2004
[13] Jens Axboe, “Linux Block IO - present and future,” In Proceedings of the Ottawa Linux Symposium, pp.51-61
[14] P. Sheoy and H. Vin,“Cello: A disk scheduling framework for next generation operating systems,” In Proceedings of ACM SIGMETRICS Conference, pp.44-555, 1997
[15] Y. Zhang and B. Bhargava,“Self-learning disk scheduling,” IEEE Trans, On knowl. And Data Eng.,21(1), pp.40-65, 2009
[16] D. Boutcher and A. Chandra, “Does Virtualization Make Disk Scheduling Passe?,”SOSP Workshop on Hot Topics in Storage and File System,2009
[17] M. Kesavan, A. Gavilovska and K. Schwa, “On Disk I/O Scheduling in Virtual Machines,” WIOV’10, May 2010
[18] Y. Xu and S. Jiang, “A Scheduling Framework that Makes any Disk Schedulers Non.work.conserving solely based on Request Characteristics,” FAST2011
[19] Kernel Virtual Machine (KVM) Best practices for KVM http://publib.boulder.ibm.com/infocenter/lnxinfo/v3r0m0/topic/liaat/liaatbestpractices pdf.pdf
[20] Yongkun Wang, Kazuo Goda, Miyuki Nakano and Masaru Kitsuregawa, “Performance Evaluation of Flash SSDs in a Transaction Processing System,” IEICE TRANS. INF. & SYST., E94-D, 3, 2011
[21] Yongkun Wang, Kazuo Goda, Miyuki Nakano and Masaru Kitsuregawa, “Early Experience and Evaluation of File Systems on SSD with Database Applications, “Proceedings of the 5th IEEE International Conference on Networking, Architecture, and Storage (IEEE NAS2010), 467-476 ,2010
[22] 服部拓也 新居健一 山口実靖, “Android OS における演算性能と I/O 性能の評価と解析 マルチメディア”,分散,協調とモバイル(DICOMO2011)シンポジウム, pp.1399-1406 ,2011

Copyright © 2013 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.

[23] Steve Rostedt. ftrace tracing inftrastructure. http://lwn.net/Articles/270971/.
[24] T. Bird, “Measuring Function Duration with Ftrace,” in Proc. of the Japan Linux Symposium, 2009
[25]SystemTap http://sourceware.org/systemtap
[26] Frank Ch. Eigler. “Problem solving with systemtap,” In Proceedings of the Ottawa Linux Symposium 2006, 2006
[27] LTTng Project http://lttng.org/
[28] M. Desnoyers, M. R. Dagenais, “The LTTng Tracer: A low impact performance and behavior monitor of GNU/Linux” Proceedings of Ottawa Linux Symposium 2006, 2006, pp. 209-223.
[29] J. Levon and P. Elie. Oprofile: A system profiler for linux. http://oprofile.sf.net, September 2004.

Copyright © 2013 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.