Vol.2018-OS-142 No.3
2018/2/27

ファイルシステムにおけるtail latencyの定量的分析

青田直大[1]　河野健二[1]

概要：ファイルシステムにおけるテール・レイテンシ (tail latency) は, サービス品質を保証する上での障壁となることが知られている. 現在のファイルシステムにおける最適化は, アクセス性能を平均的に向上させることを主目的としているため, アクセス状況によってはすべての最適化が無効となり, テール・レイテンシが増大する傾向にある. テール・レイテンシが増大する状況は, メタデータの管理手法, 最適化の手法, アクセスパターンによって異なってくる. 本研究では, btrfs, ext4, XFS, F2FS という 4 つのファイルシステムを対象に, テール・レイテンシが増大する要因を定量的に分析した結果を示す.

キーワード：テール・レイテンシ, ファイルシステム, 性能評価, オペレーティングシステム

1. はじめに

現代の多くのシステムが, 低く, 安定したレイテンシを求めている. 現代の Web サービスにおいて, ユーザからのリクエストに対するレスポンス時間はユーザエクスペリエンスにとって重要な指標となっている [1]. そのため, クラウドプラットフォームでは, 数十ミリ秒から数百ミリ秒以内でリクエストを返す要求が高まっている [2,3].

近年では特にテール・レイテンシ (tail latency) に注目が集まっている. たとえば, データの分散処理システムにおいては, 1 つのマシンの遅れにより, 全体の処理が大きく遅れてしまう. システムが大規模になればなるほど, マシン単位でのテール・レイテンシによるシステム全体への影響は大きくなる [3].

多くの先行研究が, ストレージシステムのパフォーマンスが安定していないと指摘し, その原因を分析している [4–7]. たとえば, Hao らによれば, RAID システムの稼働時間の 1.5%から 2.2%の時間において, 少なくとも 1 つのディスクが他のディスクより 2 倍以上遅くなっていると言われている [6]. また, Cao らはファイルシステムの様々な設定上でベンチマークを実行し, その設定の 25%以上において, ベンチマークのスループットが 10%以上ばらつくことを示した [5]. また, He らは, ファイルデータの配置位置に注目してファイルシステムにおけるテール・レイテンシの悪化を分析した [4]. ファイルデータの配置が離れることで, テール・レイテンシが増大することを示し, その配置の乖離を指標としてどのようなファイルシステムの設定がテール・レイテンシの悪化につながるかを分析した.

レイテンシの増大は様々な要因で発生し, 複雑かつ巨大な現在のストレージシステムにおいて, なにが原因となってあるレイテンシの増大が発生しているのかを特定することは難しい. 先行研究は, ファイルのデータ配置や, ベンチマークのスループットといったマックな指標を用いて, レイテンシの増大がファイルシステムの処理中のランダム性・キャッシュの書き出しなどのバックグラウンドの処理などによりひきおこされることを示し, それらがどのようなファイルシステムの設定というマクロな条件のもとにひきおこされているのかを分析している. 本研究では, よりミクロな視点から, ひとつひとつのシステムコールのレイテンシがどのようにばらつき, テール・レイテンシの原因となるのかを分析する.

システムコールのレイテンシを計測することで, 全く同じシステムコールであっても, ある時には他の時よりも 2 倍以上もレイテンシが増大していることが分かった. また, 同様の実験を繰返し, その結果を分析すると, ある特定の条件においてレイテンシが増大していることがわかった.

先行研究では, 主にランダム性やバックグラウンド処理といった実行時の事象を, テール・レイテンシの原因と指摘してきた. 本研究では, そういった偶発的原因に因らないファイルシステムの構造上の問題に起因するテール・レイテンシについて評価し, 分析を行い, どのような構造上の要因でファイルシステムのシステムコールのレイテンシが増大するのかを示す.

本論文の構成を以下に示す. 2 章では, 本研究の裏付けとなる XFS における構造上のテール・レイテンシの例を

[1] 慶應義塾大学
Keio Uniersity

© 2018 Information Processing Society of Japan

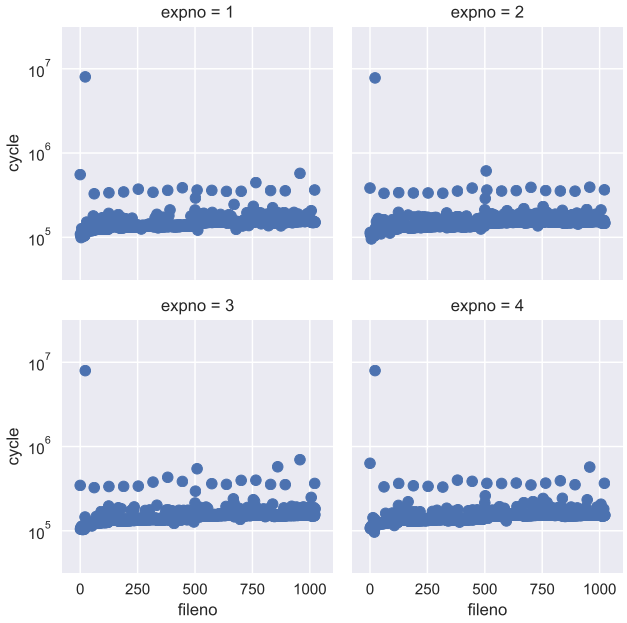

図 1 XFS 上での creat() のレイテンシ

示す. 3 章では調査の方法および調査対象について述べる. 4 章では調査結果および, その結果に対する考察を述べる. 5 章では, 関連する研究を紹介する. 6 章ではまとめを述べる.

2. Motivative Example

この章では, XFS [8] 上での creat() システムコールを例としてファイルシステムに構造上の問題による発生するテール・レイテンシがあることを指摘する.

新しくフォーマットした XFS のルートディレクトリ上に, creat() システムコールを用いて計 1,024 個のファイルを連続して作成する. また, 各 creat() の直前には sync() を行い, キャッシュなどバックグラウンドの影響によるレイテンシの増大を極力排除する. この実験を 10 回繰り返す.

図 1 は, 各ファイルの creat() のレイテンシを CPU サイクル数で計測した結果である. それぞれのグラフが各実験回の結果を示している (実験 1 から実験 4 までを表示). 多くの場合, レイテンシは 10^5 サイクル程度であることがわかる. 一方で, どの実験結果においても, 最悪ケースでは 10^7 サイクルのレイテンシが出ていることがわかる. また, 定期的に他のファイルよりもレイテンシが突出し, 300,000 から 400,000 サイクル要しているファイルが存在していることが見てとれる.

仮に一度目の実験の各ファイルを 300,000 サイクル未満の集合 (低レイテンシ群) とそれ以上の集合 (高レイテンシ群) とで 2 群に分ける. この時, 表 1 に示す通り, 低レイテンシ群の個数は 1006, 平均は 145,669.86 であり, 標準偏差は 17,071.93 となっていた. 一方, 高レイテンシ群の個数は 18, 平均は 809,339.3 であり, 標準偏差は 1,799,221 であった. 全体の 99 パーセンタイル (P_{99}) が, 358,531.6 であることから, 高レイテンシ群が全体のテール・レイテンシに寄与していると言える. また, 相対標準偏差は, 全体では 1.579 と平均の倍以上の大きなばらつきがあることを示している.

これら高レイテンシ群に属するファイルはどの実験でも共通して, 高レイテンシ群に属する傾向がある. 表 2 は, 実験 1 から実験 10 を通して, 一度でも高レイテンシ群に入ったファイルについて, そのファイルが何度の実験で高レイテンシ群に入ったかと, そのファイルのうち高レイテンシ群に入ったものでの平均・標準偏差を示したものである.

表 1 実験 1 における, レイテンシの統計. P_n は n パーセンタイルを表す. 全体および, 300000 サイクル未満の集合である低レイテンシ群と, それ以上の集合である高レイテンシ群とに分割した結果を示す.

	全体	低レイテンシ群	高レイテンシ群
個数	1,024	1,006	18
平均	157,335.9	145,669.86	809,339.3
標準偏差	248,385.1	17,071.93	1799,221
最小値	99,220	99,220	328,654
P_{95}	186,811.2	174,557	1689,668
P_{99}	358,531.6	196,898	6,748,542
最大値	8,013,260	292,552	8,013,260
相対標準偏差	1.579	0.117	0.987

表 2 少なくとも一度, 高レイテンシ群に属するファイルの統計. 回数は, そのファイルがいくつの実験において, 高レイテンシ群に入ったかを示す.

ファイル番号	回数	平均	標準偏差
0	10	537350.8	151341.91
22	10	7942296.0	284938.73
61	10	331319.0	2616.39
125	10	353485.0	19386.90
189	10	340421.8	9897.34
253	10	343218.8	11736.62
317	10	342970.8	13076.46
381	10	386030.8	32390.70
445	10	384441.0	10634.23
502	1	304514.0	-
506	1	613526.0	-
509	10	385235.4	56315.71
573	10	356795.4	4351.04
637	10	360271.6	9490.51
701	10	360545.8	17632.88
765	10	380826.0	30916.16
829	10	375386.8	42201.74
860	1	575048.0	-
893	10	361555.8	14104.86
957	10	472855.0	116315.59
1021	10	373816.4	13387.21

© 2018 Information Processing Society of Japan

表 3　調査を行う環境

CPU	Intel Xeon CPU X5650 2.67GHz
メモリ	16GB
ストレージ	INTEL SSDSC2BW12 120GB
カーネル	Linux 4.13.8
ファイルシステム	ext4, XFS, btrfs, F2FS

表 2 中に示されている通り, 高レイテンシ群に入ったファイルのほとんどが, どの実験においても高レイテンシ群に入っている.

以上の実験結果から, 同一のシステムコールを sync の直後という, 最もキャッシュやバックグラウンド処理の影響を排除した場合においても, 試行回数の 1%以上で, 全体の平均値の 2 倍近くのレイテンシとなるファイルがあることがわかった. また, それらのファイルは実験を繰り返しても, 同様にレイテンシが高くなる傾向があり, ファイルシステムの構造上の問題に起因するテール・レイテンシがあることを示唆している.

3. 調査方法

この章では, 調査を行う環境, 調査対象のファイルシステムおよび調査の方法について述べる.

表 3 は調査を行う環境を示したものである. CPU の周波数は 2.67GHZ であり, SSD を使用してハードウェアの IO にかかる部分のレイテンシを小さくして調査した. 調査対象のファイルシステムは, Linux 4.13.8 に実装されている ext4 [9], XFS [8], btrfs [10], F2FS [11] とした. 調査指標として, システムコール 1 回ごとの CPU のサイクル数を用いた. それぞれのシステムコールは, sync() の直後に実行し, キャッシュやバックグラウンド処理の影響を極力排除するようにした.

4. 調査結果と分析

ここでは, 各ファイルシステムにおける creat() システムコールのレイテンシの調査の結果を述べ, それぞれの高レイテンシがどのようなファイルシステム上の構造に起因するかを分析する.

creat() システムコールのレイテンシを分析の実験は以下のように行う. まず, ファイルシステムを新たに作成する. そのファイルシステムのルート直下に, ファイル 0 は"0000000", ファイル 1 は"0000001"というようにファイル名が 7 文字ののファイルを creat() システムコールを用いて順次作成する. 前述したように各 creat() の前には sync() を行う. creat() システムコールを呼びだしてから, 返ってくるまでの CPU サイクル数を計測する. 全ファイルを作成し終われば, 一回の実験は終了する. この実験を合計 10 回繰り返す.

4.1 XFS

XFS における, creat() のレイテンシは, すでに図 1 に示した通りである. 表 2 にあるように, どの実験においてもレイテンシが高くなるファイルが存在する.

表 2 中のうち, どの実験でも高レイテンシのもの, すなわち「回数」が 10 であるファイルを, さらに 3 群に分けることができる. 1 つはファイル 0 で, おおよそ 500,000 サイクルのレイテンシとなっている. もう 1 つはファイル 22 で, おおよそ 7,900,000 サイクルのレイテンシとなっている. 残りのファイル (61, 125, 189, ...) は, おおよそ 300,000 から 400,000 サイクルのレイテンシとなっている.

ファイル 0 は, ディレクトリ中の最初のファイルとなる. したがって, 様々な初期化処理がこのタイミングで行われることが推察される. アプリケーションにとっても, 最初のファイルであれば高いレイテンシになるということは十分に予測できる.

ファイル 22 は, XFS において常に最長のレイテンシを必要とするファイルである. ここでレイテンシが高くなるのには, XFS におけるディレクトリエントリの管理手法に原因がある. XFS では, あるディレクトリ中のファイル数が少なく, ディレクトリエントリに必要な容量が小さい場合には, ディレクトリエントリをディレクトリの inode 中の空きスペースに直接保存する [12]. デフォルトで XFS の inode のサイズは 512 バイトであり, その中の 176 バイトは inode 自体の内部情報に使われる. したがって, 残りの $512 - 176 = 336$ バイトにディレクトリエントリが記録される. このスペースにディレクトリエントリを記録する場合, 6 バイトのヘッダがつく. また, ファイル名が 7 文字の場合, 1 つのディレクトリエントリには 15 バイトが使われる. ファイル 22 を作る前までに, 22 個のファイルが作られている. したがって, inode 中でディレクトリエントリに使われているのは $\$15 \times 22 + 6 = 336\$$バイトである. これは, さきほどの inode の内部情報を除いた空き領域のサイズと一致する. すなわち, ファイル 22 を作る前の時点で, XFS の inode にはそれ以上のディレクトリエントリを記録できなくなっている. ここで新しくファイルを作るために, XFS は inode 以外の領域を確保し, そこにディレクトリエントリを記録していく. それらの新たな領域の確保, および inode 中のディレクトリエントリをその領域に移し変える作業が必要となるため, ファイル 22 は高レイテンシになっている.

残りの高レイテンシファイルであるファイル 61, 125 などは定期的に現われている. ここには法則があり, 64 個ごとに高レイテンシになっていることがわかる. これは inode のアロケートによる高レイテンシである. XFS では, inode の領域を 64 個分 $512 \times 64 = 32KB$ 分のブロックでまとめて確保する. したがって, 64 個のファイルを作るごとに, 新たな inode の領域を確保する必要がある. この領

© 2018 Information Processing Society of Japan

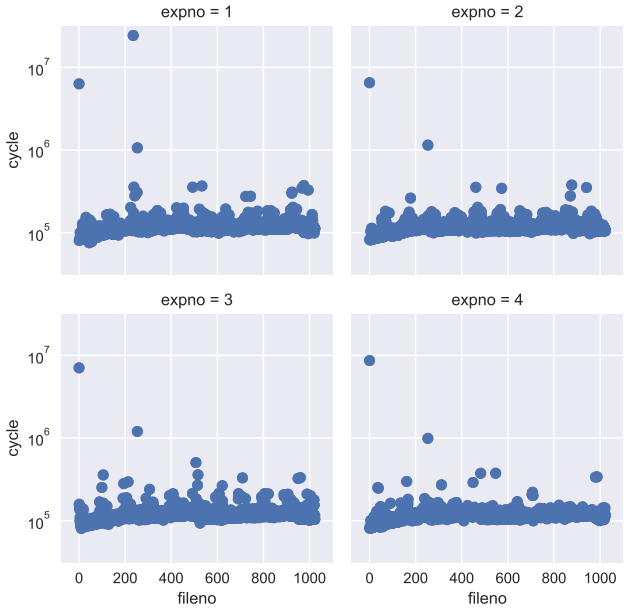

図 2 ext4 上での creat() のレイテンシ

表 4 ext4 において, 7 回以上の実験で平均以上のレイテンシになったファイル

ファイル番号	回数	平均	標準偏差
0	10	6835519	683750.12
253	10	1150177	68457.61
469	7	171953.7	64265.45
485	8	139949	8979.58
949	8	140869.2	8497.84

域確保の処理が高レイテンシにつながっている.

また, ファイル 502 やファイル 506 など, 一度の実験でのみレイテンシが高くなっているファイルも存在する. これらはランダム性や, バックグラウンド処理によるものだと考えられる. 繰り返しの実験により, こうした非構造的な要因によるテール・レイテンシは排除することができる.

4.2 Ext4

図 2 は, ext4 における creat() のレイテンシを示したものである. 全実験を通しての, レイテンシの平均は 134,746 であり, 標準偏差は 665862.6 であった.

ext4 においては, 一定して高レイテンシになるファイルは少ない. 表 4 は, 7 回以上の実験で平均以上のレイテンシになったファイルをまとめたものである. 全ての実験において, 平均以上のレイテンシになっているのはファイル 0 とファイル 253 のみである.

ファイル 0 は, 平均で 6,835,519 サイクルと全体の平均と比べて 50 倍近くの大きなレイテンシを持つ. これは XFS の場合と同様に, ディレクトリ内の最初のファイルであるため, 初期化に関する処理ために高いレイテンシになっていると考えられる.

ファイル 253 は, 平均で 1,150,177 サイクルと, 全体の

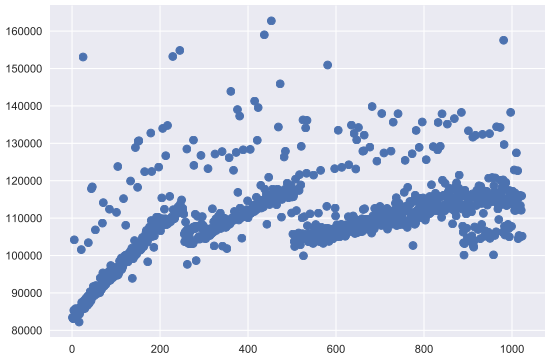

図 3 実験 1 での ext4 上での creat() のレイテンシのうち, 200,000 サイクル未満のもの

平均と比べて 8.5 倍以上の大きなレイテンシを持つ. これは ext4 のディレクトリ構造に起因するレイテンシである. Ext4 はディレクトリのデータブロックの中に, ファイルが作られた順番でリニアにディレクトリエントリを記録する [9]. ファイル 253 を作る前までの段階で, ext4 のルートディレクトリのデータブロックの中には, ファイル 0 からファイル 252 までのディレクトリエントリとともに, ".", "..", そして"lost+found"のそれぞれのディレクトリエントリが保存されている. ディレクトリエントリのサイズは, "."と".."が 12 バイト, "lost+found"が 20 バイト, そしてファイル 0 からファイル 253 までが 16 バイトとなる. したがって, ファイル 253 を作る前の段階で, ルートディレクトリののデータブロックは, $12+12+20+16\times253=4092$ バイトが使用されている. デフォルトで ext4 のデータブロックのサイズは 4096 バイトであるため, ファイル 253 のディレクトリエントリが入る空きスペースは残っていない. ここで ext4 は新しくディレクトリ用のデータブロックを確保する. データブロック内のエントリはリニアにしか探索できず, ファイル数が増えてくると効率が悪くなる. そこで, ext4 はディレクトリエントリが 1 つのデータブロックに収まらなくなったこの段階で, ファイル名のハッシュによるツリーを構築する. このツリーをたどることで, あるファイルのディレクトリエントリがどのデータブロックに配置されているかを知ることができる. このようなツリーを構築する処理が入るという ext4 の構造のために, ファイル 253 は高レイテンシとなる.

このように ext4 においても, 一定して高レイテンシを出す構造を特定することができた. さらに細かくレイテンシの挙動を分析するため, 実験 1 において, 200,000 サイクル未満のレイテンシのファイルにしぼり, 図 3 にプロットした.

図 3 から, ファイル 0 から 250 付近, 250 付近から 500 付近, そして 500 付近から 1000 付近までと 3 つの部分に分けて, それぞれの中でレイテンシが上昇傾向にあること

© 2018 Information Processing Society of Japan

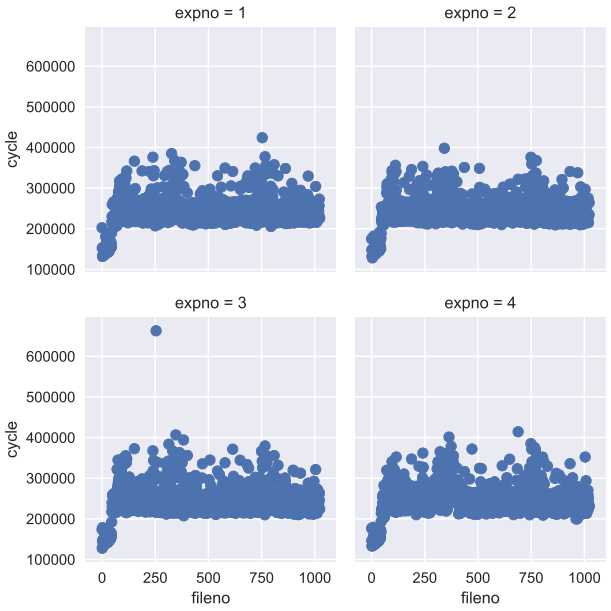

図 4 btrfs 上での creat() のレイテンシ

がわかる．これは ext4 が，ext3, ext2 といった過去のファイルシステムとの互換性を保つために生じている．もともと ext2 においては，ファイル名のハッシュツリーを使わず，ただリニアにディレクトリエントリを配置していた．ext4 は，以前のバージョンでもディレクトリを読むことができるように，1 つのデータブロックの中ではリニアにディレクトリエントリを配置し，ハッシュツリーはどのブロックにファイルが存在するかを示すためだけに使われている．すなわち，データブロック中のエントリが増えるほどに，リニアにレイテンシが増大するのは ext4 でも変わらないということになる．結果として，0 から 253 までは 1 つ目のデータブロックにリニアにエントリが配置され，それがそのままリニアなレイテンシの増大につながる．ファイル 253 からファイル 500 付近までは，ファイル名のハッシュにより，2 つのデータブロックに個々のファイルが分かれて配置される．ハッシュ関数の性質上，2 つのデータブロックには平均して同じ数のファイルが作られる．したがって，2 つのブロックに分散されている分だけ，ファイル 0 から 253 までの時よりはゆるやかに，リニアにレイテンシが増大していくことになる．さらに，ファイル 500 付近において，ふたたびデータブロックがいっぱいになりハッシュツリーによる分配が行われていることが推察される．

このように ext4 においても，構造上の問題によるテール・レイテンシ，およびレイテンシのリニアな上昇傾向を見ることができた．

4.3 Btrfs

図 4 は，ext4 における creat() のレイテンシを示したものである．全実験を通しての，レイテンシの平均は 246,546.70 であり，標準偏差は 37767.06 であった．全体的な傾向として，btrfs ではほとんどのファイルにおいて，200,000 から 300,000 サイクルにちらばったレイテンシをとっていることがわかる．

btrfs の結果において特徴的なのは，ファイル 0 からファイル 45 あたりまでに，他のファイルよりも低いレイテンシを持つ部分があることである．これは，btrfs のファイル管理に起因している．Btrfs では，ファイルの inode やディレクトリエントリなど全てを B-tree 上のキー/バリューのペアのアイテムで表現する．1 つのファイルを作る時，4 つのアイテムが btrfs の B-tree 上に挿入される．ひとつは inode のアイテムであり，ふたつが親ディレクトリからファイルを参照するためのアイテムである．そして，もうひとつがファイルがどの親ディレクトリにどんな名前で所属しているかを示す逆参照のアイテムである．これら 4 つのアイテムおよびそのキーを保存するためには，全部で 351 バイトの領域が必要となる．Btrfs において，B-tree のノードのサイズはデフォルトで 16KB であり，ファイルシステム作成時には，さきほどのアイテムが作られるノードには，ヘッダ部分などを除いて 16,061 バイトの空き領域がある．したがって，$16061/351 = 45.75\ldots$ と，45 個のファイルを作った時点でノードがひとつ埋まってしまう．この時点までは，さきほどの 4 つのアイテムは 1 つのノードに記録できる．一方，これ以降のファイルでは少なくとも 2 つ以上のノードに分かれて，さきほどの 4 つのアイテムが追加される．すなわち，ファイル 45 以前では，書きかえるノードの数が 1 つであったのに対して，それ以後では書きかえるノードが 2 つ以上となっている．これにより，ファイル 45 までとそれ以後とでレイテンシに差異が出ている．

Btrfs においても，XFS や ext4 の場合と同様に，最初に書きはじめた領域が不足することで，新たな領域の確保が必要となりレイテンシが増大している．しかしながら，他の 2 つのファイルシステムのような突出したレイテンシの増大は見られない．これは btrfs が Copy-on-Write のファイルシステムであるからだ．Copy-on-Write のファイルシステムは，書きかえに際して常にコピーを行う．したがって，2 つのノードを書きかえるための処理と，1 つのノードを 2 つに分割する処理との間では，どちらも新たな領域を 2 つ確保してコピーを行うと，ほぼ同じように動作してしまう．こうした性質により，突出したレイテンシが見られないものと考えられる．

4.4 F2FS

F2FS においては，1024 個のファイルの creat() ではファイル 0 でのレイテンシの突出はあったものの，大きな傾向を見つけるには至らなかった．そこで，実験を拡張し，32768 個のファイルを作成した．図 5 はその結果を示したものである．

© 2018 Information Processing Society of Japan

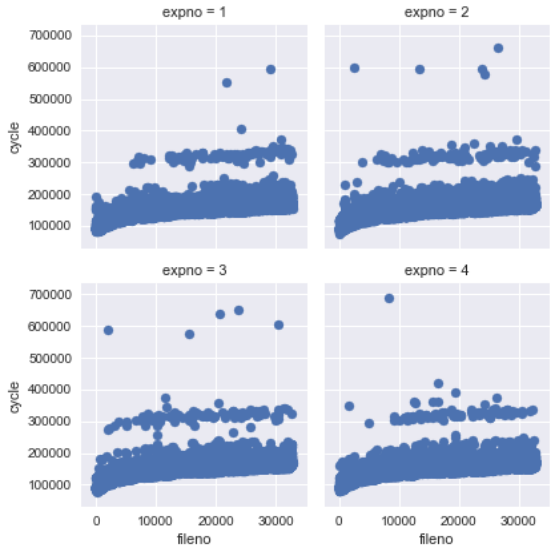

図 5 F2FS 上での creat() のレイテンシ

図の通り, 300,000 サイクル付近の群とそれ未満の群との大きな 2 つの集合を見ることができる. F2FS においても, なんらかの構造的テール・レイテンシがあることが示唆される. しかしながら, 高レイテンシ群に入るファイル番号はどの実験においても異なっており, さらに高レイテンシ群が初めにあらわれる位置も, ファイル 0 近くからファイル 10000 あたりまでと幅広くなっている. 現状においては, 一定の構造上の理由を見つけるに至ってはいない. F2FS が, ログ記録型のファイルシステムで追記のみが主に行われることから, GC が発生しにくい今回のワークロードでは, 構造上の問題が出にくいということも考えられる.

5. 関連研究

様々な先行研究によって, 大規模システムにおいてテール・レイテンシが大きな問題となることが指摘されている [3, 7]. MittOS では, read() システムコールに SLO の引数を追加し, ストレージスタックが SLO を満たせないと判断した場合に, エラーを返し, 早めに他のマシンへのリトライを可能にした [7]. 複数のマシンにデータが複製され, リクエストを分散して投げることができる環境においては, 早めにリトライすることでテール・レイテンシが改善される.

多くの先行研究が, ストレージシステムのパフォーマンスが安定していないと指摘し, その原因を分析している. Hao らは, ストレージシステムのうちハードウェアを対象にテール・レイテンシを分析した [6]. RAID システムの稼働時間の 1.5%から 2.2%の時間において, 少なくとも 1 つのディスクが他のディスクより 2 倍以上遅くなっていると言われている. また, Cao らはファイルシステムの様々な設定上でベンチマークを実行し, その設定の 25%以上において, ベンチマークのスループットが 10%以上ばらつくことを示した [5]. また, He らは, ファイルデータの配置位置に注目してファイルシステムにおけるテール・レイテンシの悪化を分析した [4]. ファイルデータの配置が離れることで, テール・レイテンシが増大することを示し, その配置の乖離を指標としてどのようなファイルシステムの設定がテール・レイテンシの悪化につながるかを分析した. Amvrosiadis らは, バックアップや dedupe などの大量の IO が必要なメンテナンス作業が, レイテンシを増大させる原因になることを示し, その作業に必要な IO を通常の IO が起きたタイミングで実行することで, 通常の IO によるキャッシュを活用しメンテナンスのパフォーマンスを改善するという手法を提案した [13]. これらの先行研究では, 主にファイルシステム全体の mount option や, mkfs の option がどのようにファイルシステムの全体的なパフォーマンスに影響するかを分析している. 本研究では, システムコール一度ごとのレイテンシを対象とし, ファイルシステムがどのような状態である時に, レイテンシが悪化するのかを明らかにしており, 相互に補完する関係にある.

6. まとめ

本研究では, ファイルシステムの構造的な原因に起因するテール・レイテンシに着目し, そうしたテール・レイテンシの計測と分析を行った. creat() システムコールのレイテンシを計測する実験を繰り返すことで, 偶発的なテール・レイテンシを除き, 構造的なテール・レイテンシを見つけだすことができる. これらのテール・レイテンシについて詳細に分析を行い, なぜそういったテール・レイテンシが発生しているのかを明らかにした.

謝辞

本研究は JSPS 科研費 JP16J03272 および JP16K00104 の補助を受けています。

参考文献

[1] Brutlag, J.: Speed Matters for Google Web Search, `http://services.google.com/fh/files/blogs/google_delayexp.pdf` (2009).

[2] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P. and Vogels, W.: Dynamo: Amazon's Highly Available Key-value Store, *SIGOPS Oper. Syst. Rev.*, Vol. 41, No. 6, pp. 205–220 (online), DOI: 10.1145/1323293.1294281 (2007).

[3] Dean, J. and Barroso, L. A.: The Tail at Scale, *Commun. ACM*, Vol. 56, No. 2, pp. 74–80 (online), DOI: 10.1145/2408776.2408794 (2013).

[4] He, J., Nguyen, D., Arpaci-Dusseau, A. C. and Arpaci-Dusseau, R. H.: Reducing File System Tail Latencies with Chopper, *Proceedings of the 13th USENIX Conference on File and Storage Technologies*, FAST'15, Berkeley, CA, USA, USENIX Association, pp. 119–133 (2015).

© 2018 Information Processing Society of Japan

[5] Cao, Z., Tarasov, V., Raman, H. P., Hildebrand, D. and Zadok, E.: On the Performance Variation in Modern Storage Stacks, *15th USENIX Conference on File and Storage Technologies (FAST 17)*, Santa Clara, CA, USENIX Association, pp. 329–344 (online), available from ⟨`https://www.usenix.org/conference/fast17/technical-sessions/presentation/cao`⟩ (2017).

[6] Hao, M., Soundararajan, G., Kenchammana-Hosekote, D., Chien, A. A. and Gunawi, H. S.: The Tail at Store: A Revelation from Millions of Hours of Disk and SSD Deployments, *Proceedings of the 14th Usenix Conference on File and Storage Technologies*, FAST'16, Berkeley, CA, USA, USENIX Association, pp. 263–276 (2016).

[7] Hao, M., Li, H., Tong, M. H., Pakha, C., Suminto, R. O., Stuardo, C. A., Chien, A. A. and Gunawi, H. S.: MittOS: Supporting Millisecond Tail Tolerance with Fast Rejecting SLO-Aware OS Interface, *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, ACM, pp. 168–183 (online), DOI: 10.1145/3132747.3132774 (2017).

[8] Sweeney, A., Doucette, D., Hu, W., Anderson, C., Nishimoto, M. and Peck, G.: Scalability in the XFS File System, *USENIX ATC*, (online), available from ⟨`http://oss.sgi.com/projects/xfs/papers/xfs_usenix/index.html`⟩ (1996).

[9] Mathur, A., Cao, M., Bhattacharya, S., Dilger, A., Tomas, A. and Vivier, L.: The new ext4 filesystem: current status and future plans, *Proceedings of Linux Symposium*, pp. 21–33 (online), available from ⟨`http://ols.108.redhat.com/2007/Reprints/mathur-Reprint.pdf`⟩ (2007).

[10] Rodeh, O., Bacik, J. and Mason, C.: BTRFS: The Linux B-Tree Filesystem, *ACM Transactions on Storage*, Vol. 9, No. 3, pp. 1–32 (online), DOI: 10.1145/2501620.2501623 (2013).

[11] Lee, C., Sim, D., Hwang, J.-y., Cho, S. and Clara, S.: F2FS : A New File System for Flash Storage, *Proceedings of the 13th USENIX conference on File and Storage Technologies (FAST'15)* (2015).

[12] Inc., S. G.: XFS Filesystem Structure, `http://xfs.org/docs/xfsdocs-xml-dev/XFS_Filesystem_Structure/tmp/en-US/html/index.html` (2006).

[13] Amvrosiadis, G., Brown, A. D. and Goel, A.: Opportunistic Storage Maintenance, *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, New York, NY, USA, ACM, pp. 457–473 (online), DOI: 10.1145/2815400.2815424 (2015).

© 2018 Information Processing Society of Japan