2007－EVA－23 (2)
2007／11／19

ext3ファイルシステムにおける ジャーナリング処理トレースツールの開発

関山 友輝， 大島 敬志， 角谷 有司， 大島 訓，杉田 由美子

日立製作所システム開発研究所

大規模なエンタープライズ用途へのLinuxの適用が進むにつれ，高い信頼性や可用性が求められている．Linuxの標準的なファイルシステムであるext3ではジャーナリング機能によって信頼性，可用性を高めているが，取り扱うデータ量の増大に伴い，ジャーナリング機能の影響でユーザプロセスの処理が遅延するという問題が顕在化している．しかしながら，その根本的な原因は明らかになっていない．そこで，ジャーナリング処理の挙動に関する情報を収集するためのトレースツールを開発し，遅延原因を解明した．

The Development of Tracer for Journaling Operation in Ext3 Filesystem

TOMOKI SEKIYAMA KEISHI OSHIMA YUJI KAKUTANI
SATOSHI OSHIMA YUMIKO SUGITA
Systems Development Laboratory, Hitachi, Ltd

To apply Linux to large-scale enterprise systems, high reliability and availability are required. Ext3, the standard filesystem used in Linux, has a journaling system to improve reliability and availability. However, it is revealed that user processes can be delayed by journaling as the system handles large data, and the root cause of this probrem is not clarified. To solve this problem, we have developed the tracer to collect information about the behavior of journaling operations, and have clarified the cause of the delay.

1 はじめに

近年Linuxのエンタープライズ用途への適用が進んでおり，高い信頼性や可用性が求められるようになっている．現在Linuxで標準的に採用されているファイルシステムとして，ext3ファイルシステム[1]がある．ext3ファイルシステムは，初期のLinuxシステムで広く使われていたext2ファイルシステム[2]にジャーナリング機能[3]を追加したものである．これによってシステムが予期せずクラッシュした場合にファイルシステムのリカバリを高速化することができ，信頼性や可用性の向上を図ることができる．

しかしながら，ジャーナリング機能に必要な処理の一部がユーザプロセスのコンテクストで実行されるため，ディスクアクセスを行ったユーザプロセスの処理がジャーナリング機能の影響によって遅延することがあり、それによって応答性能が悪化するという問題がある．特にシステムが大規模になるにつれて取り扱うデータ量が増大したことで，ジャーナリング処理によって引き起こされる遅延も長時間に及ぶ事例が見られる．

システムの安定化を図るためには，遅延が発生した際のジャーナリング処理の挙動を分析する手法が必要となる．そこで本稿では，ジャーナリング処理によって処理遅延が発生する例を示すとともに，ジャーナル処理の挙動を記録して遅延の原因解析を支援するトレースツールの開発について述べる．また，開発したトーレスツールを用いて遅延の原因を解析し，対策方法の検討を行う．

2 ext3における処理遅延の例

本章では，ext3ファイルシステムを使用するLinuxシステムで，ユーザプロセスの処理が長時間遅延した例を示す．

遅延の発生を確認したLinuxシステムの構成は

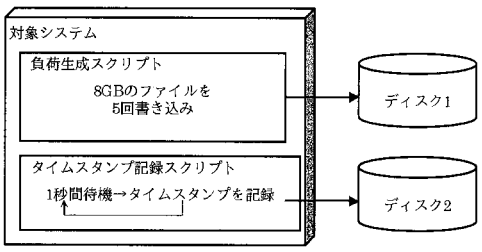

図 1　ディスク負荷生成スクリプト

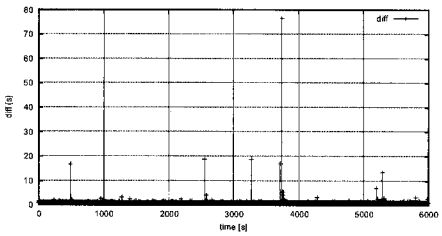

図 2　タイムスタンプ間の差分

以下の通りである．

- OS: Linux (kernel-2.6.9)
- アーキテクチャ: Intel 64 (CPU: Itanium 2)
- メモリ容量: 16GB
- ディスク: RAID1 構成 133GB × 2 組
  FibreChannel 接続

大容量のデータを取り扱う際のシステムの応答性能を評価するため，図 1 に示す負荷をかける．まず，このシステムに接続された 2 組のディスクのそれぞれに ext3 ファイルシステムを作成する．そのうち一方のディスク上に 8GB のファイルを複数作成する「負荷生成スクリプト」を実行し，I/O 負荷をかける．この状況下で，別のディスク上のファイルにタイムスタンプを 1 秒ごとに追記する「タイムスタンプ記録スクリプト」を実行する．

このとき記録されたタイムスタンプ間の差分をプロットしたものを図 2 に示す．この図から，最大で 76 秒間，タイムスタンプ間に空白が発生した期間が見られる．また，5 秒を超える空白が同スクリプトの実行に要した約 100 分間に 9 回見られた．

空白の発生していた期間中は，タイムスタンプ記録スクリプトの動作が停止していたと考えられる．同様の評価を ext2 ファイルシステムで行った場合には正常に 1 秒ごとにタイムスタンプが記録されることから，ext3 のジャーナリング処理の影響でプロセスの処理が遅延している可能性が疑われる．

このような遅延が発生すると，ログ等をディスクに書き込む際にユーザプロセスが長時間停止し，システムの応答性能が大幅に悪化する恐れがある．従って，この原因を分析し対策を講じる必要がある．

Linux ではジャーナリング機能のデバッグのために，ジャーナリング処理の動作をコンソールに表示するメッセージ出力機能が備わっているが，大容量のデータを取り扱う場合に同機能を使用すると非常に多量のメッセージが出力されるため，ジャーナリング処理の挙動を把握することは困難である．また，メッセージ出力自体が新たな負荷となり，システムの挙動が変わってしまう恐れがある．

そこで以下では，ext3 におけるジャーナリング機能の動作を説明するとともに，ジャーナリング処理の挙動を把握するための情報を低負荷で収集する JBD トレースツールを開発すること検討する．

3　ext のジャーナリング機能の概要

ext3 では，JBD(Journaling Block Device) と呼ばれるシステムを利用してジャーナリング機能を実現している．本章では，JBD を用いたジャーナリング処理の概要を説明する．

ジャーナリング機能は，複数の互いに関連したブロックに変更を加える際，一旦ジャーナル領域と呼ばれるディスク領域に更新内容を書き出しておき，それが完了してから実際のブロックを変更する．予期しないシステムクラッシュが発生した際には，再起動時にジャーナル領域の記録に基づいて更新途中だったブロックの更新を完了させることで，ファイルシステム内の一貫性を常に維持することができる．

3.1　ジャーナリング処理のモード

JBD がジャーナリングの対象とするブロックには，ファイルシステム内の管理情報を格納するメタデータブロックと，ファイル内に記録された情報を格納するデータブロックの 2 種類がある．

このうち，メタデータブロックは常にジャーナル領域に変更内容を記録した後に更新される．一方，データブロックの取り扱い方法は，ext3 ファイルシステムのマウントに以下の 3 つのモードのいずれかを指定することで変更することができる．

- data=journal
  データブロックもメタデータと同様，ジャーナリング処理の対象とする．

- data=ordered
  データブロックはジャーナリング処理の対象としないが，メタデータの変更内容をジャーナル領域に書き出す前にデータブロックを書き出すことにより，未書き込みのデータがファイル内に現れることを防ぐ．

- data=writeback
  データブロックの更新はジャーナリングの対象とせず，メタデータとの書き出し順序も保障しない．

以降の議論では，デフォルトのモードである data=ordered を前提とする．

3.2 JBD によるジャーナリング処理の流れ

JBD によるジャーナリング処理は，次の流れで行われる．

1. ジャーナリング対象ブロックの更新開始
   ユーザプロセスが ext3 ファイルシステム上のブロックを変更しようとすると，ext3 ファイルシステムから JBD にジャーナリング処理の開始が通知される．このとき JBD は，必要と思われるジャーナル領域を予約する．十分な領域が予約できない場合は，後述するチェックポイント処理を開始し，必要な領域が確保できるまで待機する．

2. ブロックの更新
   変更するブロックを JBD に通知した後，そのブロックに対応するメモリ上のキャッシュの内容を変更する．これを必要なだけ繰り返す．JBD は一連の変更されたブロックをトランザクションと呼ばれる単位に纏めて管理する．

3. ジャーナリング対象ブロックの更新終了
   一連の変更が完了したら，それを JBD に通知する．このとき，予約したジャーナル領域のうち使用しなかったものは解放される．非同期書き込みの場合，ユーザプロセスはここでもとの処理に復帰する．

4. データブロックのフラッシュ処理
   data=ordered モードでファイルシステムがマウントされている場合，kjournald と呼ばれるデーモンが，下記のコミット処理に先立ってトランザクションに含まれるデータブロックの変更内容をメモリ上のキャッシュからディスクに書き出す．

5. メタデータブロックのコミット処理
   kjournald によって，トランザクション内のメタデータブロックの変更内容がジャーナル領域に書き出される．

6. トランザクションのチェックポイント処理
   トランザクション内のメタデータブロックの変更内容を，実際にディスクに反映させる．この処理は，1 で十分なジャーナル領域が予約できなかった場合や，同期書き込みを行った場合に，ユーザプロセスのコンテクストで行われる．

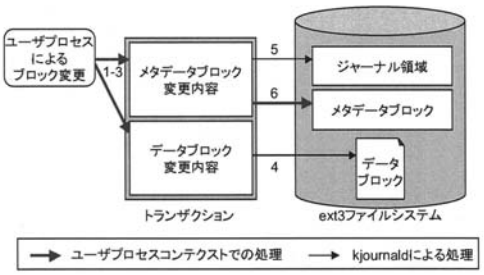

図 3　ジャーナリング処理におけるデータの流れ

上記の処理によるデータの流れを，図 3 に示す．このうち，1 - 3, 4 - 5, および 6 の処理はそれぞれ異なるトランザクションに対して同時に実行することができる．JBD ではトランザクションごとの進行度合いを管理するために，ジャーナリング処理の段階を表す値を，各トランザクションに付与された t_state という変数に記録する．図 4 にその変化を示す．

4 JBD トレースツールの設計と実装

本章では，JBD の動作をトレースツールに必要な機能を検討し，その実装方法について述べる．

4.1 JBD トレースツールの要件

ジャーナリング処理によるユーザプロセスの遅延を解析するためには，まずユーザプロセスが待

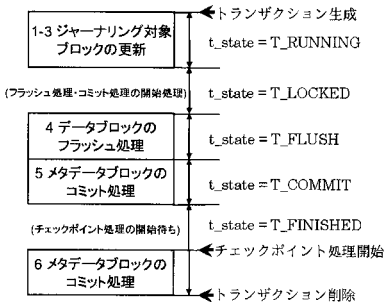

図 4　トランザクションの状態遷移

機させられる可能性のある処理の所要時間を分析する機能が必要である．そのためには，次の情報をトレースする必要がある．

要件 1 ユーザプロセスの待機を伴う可能性のある処理の実行開始，および終了のトレース

この機能によってユーザプロセスの遅延が確認された場合，次にその発生過程を分析する必要がある．そのためには，トランザクションやジャーナル領域といった，JBD が内部で管理する情報をトレースする必要がある．具体的なトレース項目としては，以下のものが挙げられる．

要件 2 トランザクションの状態遷移のトレース
図 4 に示すように，ジャーナリング処理の進行に伴ってトランザクションが生成，削除されたり，t_state の値が変更される．これらの状態遷移をトレースすることで，ジャーナリング処理の状況を分析することが可能になる．

要件 3 トランザクションの各処理対象ブロック数のトレース
各トランザクションには，ジャーナリング処理に含まれる各処理の対象となるブロックが含まれている．このブロック数の変化をトレースすることで，各処理ごとの負荷や実行時間を分析することが可能になる．

要件 4 ジャーナル領域の使用状況のトレース
ジャーナル領域内の予約されているブロック数，および空いているブロック数の変化をトレースする．これにより，チェックポイント処理の契機など，ジャーナル領域の使用量に応じた処理過程の分析が可能になる．

また，トレースによって CPU やディスクに対して新たな負荷が発生すると，評価対象システムの挙動が変わってしまう恐れがあるため，以下の機能が必要である．

要件 5 低負荷でトレース情報を蓄積する機能
トレースした情報をカーネルメモリ内に確保したバッファに記録することで，プロセス切り替えによる CPU 負荷やディスク負荷を抑制する必要がある．

4.2　JBD トレースツールの実装

前節で述べた要件を満たすために，本研究では LKST(Linux Kernel State Tracer)[4) 5)] に対して JBD の動作をトレースする機能を追加することで，JBD トレースツールを実装した．

図 5 に LKST の概要を示す．LKST は，Linux カーネルのソースコード内の様々な処理部分にフックポイントを埋め込むことにより，カーネル内部の情報を収集するトレースツールである．フックポイントが実行されると，LKST に対してイベントが発生する．イベントの種類に応じて，マスクセットで設定されたイベントハンドラが呼び出され，カーネル内の情報が引き渡される．イベントハンドラは，受け取った情報を加工するなどの処理をした上で，イベントバッファと呼ばれるカーネル内のメモリにトレース情報を記録する．これによりトレースによる負荷の発生を最低限に抑えられ，前節の要件 5 を満たすことができる．

こうして収集されたイベントバッファ内のトレース情報は，任意のタイミングで読み出してファイルに記録することができる．収集した情報を各種のアナライザと呼ばれる解析ツールに入力することで，I/O やプロセス状態など様々な観点での解析を行うことが可能である．

本研究では，前節で述べた要件 1 - 4 に対応する JBD トレース用のフックポイント，イベントハンドラ，およびアナライザを新規に開発し，JBD の挙動のトレースを実現した．以下では，各要件を満たすために追加した機能を説明する．

ユーザプロセスの待機を伴う可能性のある処理の実行開始，および終了のトレース

ユーザプロセスの待機を伴う関数として，3.2 で述べたジャーナリング対象ブロックの更新開始時に，必要なジャーナル領域が予約できない場合，ユー

ザプロセスが待機させられる．従って，この処理の開始，および終了時にフックポイントを挿入し，イベントを発生させるようにした．

また，この開始，終了イベントから，この処理の実行時間を計算する jbdhdltime アナライザを解析ツールに追加した．

トランザクションの状態遷移のトレース

トランザクションの生成部分，削除部分，チェックポイント開始部分，および t_state 変数を変更する処理の直後にフックポイントを挿入し，イベントを発生させるようにした．

また，そのイベントを抽出し，トランザクションの状態遷移を時系列に表示する jbdstate アナライザを追加した．

トランザクションの各処理対象ブロック数のトレース

トランザクション内に含まれるブロックの数は，コミット対象の必要なメタデータブロック数を除いては JBD 内では計数されていない．そこで，ジャーナリング処理に含まれる各処理の対象ブロックをトランザクションに関連付ける処理，およびその関連付けを解除する処理に対してフックポイントを追加した．

そして，このフックポイントによって生成されたイベントを計数し，求めたブロック数を記録するイベントハンドラを LKST に追加した．また，記録されたブロック数を抽出し，各トランザクションにつき，各処理ごとに対象ブロック数の変化を時系列に表示する jbdjhlen アナライザを追加した．

ジャーナル領域の使用状況のトレース

ジャーナル領域の予約，解放や，空きブロック数の変更を行っている箇所にフックポイントを追加し，イベントを発生させるようにした．

また，予約されたブロック数，および空きブロック数を抽出して時系列に表示するアナライザを解析ツールに追加した．

5 JBD トレースツールによる解析例

以下では，2 章で述べた処理遅延例について，開発した JBD トレースツールを用いて原因解析と対策を行った例を説明する．

5.1 処理遅延の原因解析

まず，2 章で述べたものと同一の環境において，Linux カーネルに対して JBD トレースツールを含む LKST のパッチを適用した．その上で，トレース情報の格納用に 256MB のメモリを割り当て，図 1 と同様のスクリプトを実行した．このとき，タイムスタンプの記録時に前回との差分を計算し，差分が 5 秒を超えた場合には LKST のイベントバッファからトレース情報を読み出してファイルに保存するよう，スクリプトの変更を行った．

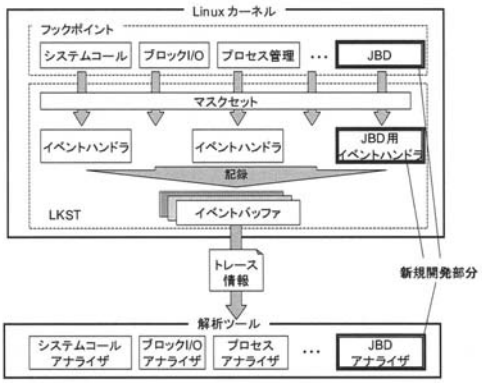

図 5 LKST の構成と新規開発部分

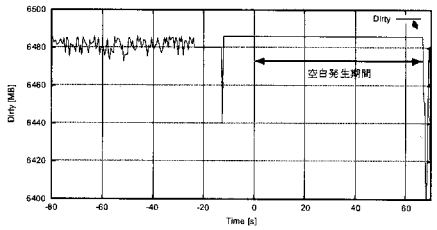

図 6 空白発生期間前後のダーティページ量

タイムスタンプ記録スクリプトの遅延要因の解析

収集した LKST のトレース情報に対して，従来の LKST で提供されている，システムコールの実行に要した時間を解析する syscall アナライザ，プロセスの状態遷移を解析する procstat アナライザ，およびプロセスの待機時間および待機要因を解析する waittime アナライザを適用し，タイムスタンプ記録スクリプトの長時間待機の発生時の挙動を調査した．

その結果，同スクリプトがタイムスタンプを記録するために write システムコールを呼び出した際，システムコールの内部から blk_congestion_wait と

いう関数が繰り返し呼ばれたことでプロセスが待機させられており，長時間に亘ってユーザ空間の処理が行われていなかったことが確認できた．この関数についてLinuxカーネルのソースコードを検証したところ，ディスクに未書き込みのキャッシュであるダーティページの量が一定以上になった際に，ディスク書き出しが完了してダーティページ量が減少するのを待機するという処理を行っていることが分かった．

タイムスタンプ記録スクリプトの停止していた期間の前後のダーティページ量の変化を図6に示す．本実験では負荷生成スクリプトによって，大量のダーティページが生成されている．図中の約-16秒の時点からダーティページ量が一定のままとなり，約71秒に急激に減少している．

同期間のうち，約0 - 71秒の期間中にタイムスタンプ記録スクリプトは停止していた．このことから，負荷生成スクリプトによって大量のダーティページが生成され，そのままダーティページ量が減少しなくなったことが，タイプスタンプ記録スクリプトがディスクに書き込みを行った際に遅延を起こした原因と考えられる．

負荷生成スクリプトの遅延要因の解析

通常Linuxでは，ダーティページ量が閾値を上回った場合，そのダーティページが属するディスクに書き込みを行ったプロセスによって，ダーティページをディスクに書き出す処理が行われ，それによってダーティページ量が減少する．従って，本実験においては負荷生成スクリプトが書き出し処理を行い，ダーティページ量は減少するはずである．しかし，図6では長時間に亘ってダーティページ量が減少していないことから，負荷生成スクリプトも動作していなかったことが考えられる．

そこで，従来のLKSTで提供されているwait-timeアナライザを使用し，負荷生成スクリプトの待機の状況を確認した．その出力の一部を図7に示す．図中のaddressはプロセスが待機中のアドレス，calling_pointはaddressに対応する関数，startは待機の開始時刻，wait-timeは待機時間を表す．この結果から，(*)で示す箇所において約93.98秒間の待機が発生しており，これによってダーティページの書き出しが停止していたと考えられる．

待機箇所である__log_wait_for_space関数はJBDに含まれる関数であり，3.2の1で述べた，必要なジャーナル領域が予約できなかった場合の待機処理を行う．しかし，従来のLKSTの機能では，この関数で長時間に亘って待機した原因を解析するのに十分な情報が得られず，原因究明は困難である．そこで，開発したJBDトレースツールを用いて，処理遅延が発生した際のJBDの動作の解析を行った．

処理遅延発生時のJBDの挙動の解析

最初に，この処理遅延がJBDの影響であることを確認するため，開発したJBDトレースツールのうちjbdhdltimeアナライザを使用して，負荷生成スクリプトのブロック更新開始時に要した時間を確認した．その出力の一部を図8に示す．deviceはジャーナリング対象のデバイス，startは処理開始時刻，starting-timeは処理に要した時間を示す．前項で述べた長時間待機は，図の(§)で示した期間内であることから，ジャーナリング対象ブロックの更新開始処理中にユーザプロセスの遅延が発生したことが分かる．

次に，jbdstateアナライザを使用して，負荷生成スクリプトが書き込んでいるディスクについて，待機の発生した期間のJBDの挙動を確認した．その出力の一部を図9に示す．transactionはトランザクションのアドレス，device-tidはtransactionに該当するデバイスとトランザクションに付与されたID，startは状態遷移イベントの発生した時刻，stateは状態遷移の内容を表す数値である．stateの各値の意味は，同図中に付記した．この結果から，図7に示した待機期間中に，ID=8983のトランザクションのチェックポイント処理が行われており，その処理に約94.22秒を要していることが分かる．図8の(†)に示した期間内であることから，ジャーナリング対象ブロックの更新開始処理中に領域の予約ができなかったために，このチェックポイント処理が開始されたと判断できる．また(‡)部分から，同時にID=8987のトランザクションのフラッシュ処理が行われており，その処理に約94.50秒を要していることが分かる．

このチェックポイント処理およびフラッシュ処理の対象ブロック数をjbdjhlenアナライザによって分析した．その出力の一部を図10に示す．図中のtid-listはトランザクションIDおよび対象処理内容(16進数形式および可読形式)，startは記録時刻，list-lenは対象ブロック数を表す．可読形式の対象

```
waiting time on waitqueue analyzer
         address             calling_point           start[sec]       wait-time
a0000001001334e0      __wait_on_buffer+140 1187594516.241611102      0.562848369
a000000100376320      get_request_wait+140 1187594516.910661594      0.000002631
a0000001001334e0      __wait_on_buffer+140 1187594517.504729471      2.514059677
a000000200044f80 __log_wait_for_space+2e0 1187594520.761237702     93.981075869 (*)
a000000100376320      get_request_wait+140 1187594615.815810365      0.000004403
a0000001001334e0      __wait_on_buffer+140 1187594615.815823074      0.064846280
```

図 7　負荷生成スクリプトの待機状況

```
JBD  handle starting time analyzer
  device   device           start[sec]   starting-time
00800011 00800011 1187594495.040165894      0.000000266
00800011 00800011 1187594495.040183208      0.000000249
00800011 00800011 1187594495.040213060    120.794389311 (§)
00800011 00800011 1187594615.834674188      0.000000580
00800011 00800011 1187594615.834677691      0.000000446
```

図 8　負荷生成スクリプトのジャーナリング開始の所要時間

```
JBD transaction state analyzer
     transaction         device-tid           start[sec]          state
0080001100002317      00800011-8983 1187594520.471499008  6.000000000 チェックポイント処理開始 ) 94.22 秒 (†)
0080001100002317      00800011-8983 1187594614.691078122  7.000000000 チェックポイント処理完了

008000110000231b      00800011-8987 1187594456.138898231  0.000000000 トランザクション生成
008000110000231b      00800011-8987 1187594519.296336397  1.000000000 コミット処理開始待ち
008000110000231b      00800011-8987 1187594520.026027934  3.000000000 フラッシュ処理開始   ) 94.50 秒 (‡)
008000110000231b      00800011-8987 1187594614.522368694  4.000000000 コミット処理開始     ) 0.14 秒
008000110000231b      00800011-8987 1187594614.661523100  5.000000000 コミット処理完了
```

図 9　処理遅延発生中のジャーナリング処理の挙動

```
JBD journal_head list length analyzer
         tid-list   tid-list           start[sec]          list-len
0000231b00000000   8987_CHKP 1187594614.644261960    1647.000000000
0000231b00000001   8987_SYNC 1187594520.026026248 2304387.000000000
```

図 10　チェックポイント処理およびフラッシュ処理の対象ブロック数

処理内容で，CHKP はチェックポイント処理対象ブロック数，SYNC はフラッシュ処理対象のデータブロック数を意味する．この結果から，チェックポイント処理対象ブロックは 1647 ブロック，フラッシュ処理の必要なデータブロックが約 230 万ブロックあったことが分かる．1 ブロックのデータ量は 4KB であることから，フラッシュ処理が必要なデータは約 9.2GB に相当し，処理完了までに長時間を要したと考えられる．また，大量のデータのフラッシュ処理によってディスク I/O が輻輳し，チェックポイント処理に必要なディスク I/O が遅延したことが長時間待機の要因と考えられる．

処理遅延要因のまとめ

以上の分析から，タイムスタンプ記録スクリプトの処理遅延は図 11 に示す原因によって発生したと考えられる．まず，負荷生成スクリプトによって，大量のダーティページが生成され，ダーティページ量の閾値を超えた状態となる．このため，タイムスタンプ記録スクリプトはダーティページ量が減少するまで待機状態となる．

このとき，負荷生成スクリプトがジャーナリング開始処理を行った際に必要なジャーナル領域を予約できず，空き領域を作成するためにチェックポイント処理を開始する．これと同時期に kjournald によって大量のデータのフラッシュ処理が行われることで，ディスク I/O が輻輳し，チェックポイント処理も長時間に及ぶ．これによって，ダーティページの回収が長時間停止し，タイムスタンプ記録スクリプトも待機したままとなる．

5.2　処理遅延の対策

これまでの分析から，kjournald によるフラッシュ処理のデータ量を抑えて所要時間を短縮すれば，ユーザプロセスの待機時間を短縮することができると考えられる．kjournald がフラッシュ処理において書き出す必要があるのは，トランザクションに含まれる変更されたデータブロックのうち，未書き出しのもの，すなわちダーティページである．従って，ダーティページ量の上限値であるカーネ

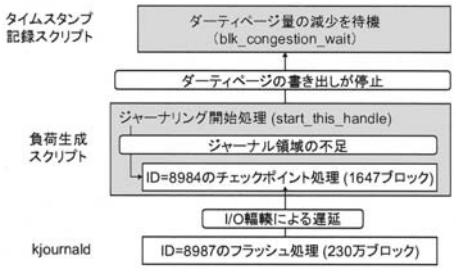

図 11　処理遅延の発生要因

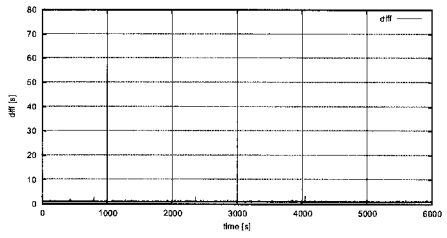

図 12　vm.dirty_ratio=5 での遅延時間

ルパラメータ vm.dirty_ratio の設定値を低減することで，フラッシュ処理に要する時間を短縮できると考えられる．

これを確認するため，vm.dirty_ratio をデフォルト値の 40 から最低値である 5 に変更した上で同様の実験を行った．その際記録されたタイムスタンプの差分を図 12 に示す．

図 2 と比較して，vm.dirty_ratio を低減したことで 5 秒以上の処理遅延が発生しなくなったことが分かる．このときのフラッシュ処理の対象ブロック数を jbdjhlen アナライザで確認したところ，最大で約 22 万ブロックとなっており，実際に kjournald の処理量が低減したことを確認できた．

5.3　トレースによるオーバーヘッドの計測

JBD トレースツールの利用時に発生するオーバーヘッドを計測するための実験を行った．前節の実験と同様の環境で，JBD トレースツールを全て有効にした場合と無効にした場合のそれぞれで，1MB のファイルを 1024 個作成するのに要する時間を計測した．

計測の結果を表 1 に示す．この結果から，JBD トレースツールを利用することによるオーバーヘッドは 1.3%程度であり，このオーバーヘッドに隠蔽されない数%以上の影響の評価に適用可能であるといえる．

表 1　1MB のファイル 1024 個の作成に要した時間

JBD トレースツール	有効	無効
所要時間	35.02 秒	34.56 秒
比率	101.3%	100.0%

6　おわりに

本稿では，大容量のデータを取り扱う Linux システムにおいて ext3 ファイルシステムを利用した際に，ジャーナリング処理の影響で長時間に亘ってユーザプロセスの処理遅延が発生する問題の例を示した．また，その原因を特定するために，LKST を拡張して低負荷でジャーナリング機能の挙動をトレースできる JBD トレースツールを開発し，同問題の原因を解明するとともに，カーネルパラメータの調整による対策でこの問題を回避できることを確認した．

今後，開発した JBD トレースツールをより多くの事例に適用していくとともに，ext3 ファイルシステムやジャーナリング機能によるユーザプロセスへの影響の要因分析に必要なトレース情報をより充実させていきたい．

参考文献

1) Theodore Y. Ts'o, Stephen Tweedie, *Planned Extensions to the Linux Ext2/Ext3 Filesystem*, The 2002 USENIX Annual Technical Conference, 2003.

2) Rémy Card, Theodore Ts'o, Stephen Tweedie, *Design and Implementation of the Second Extended Filesystem*, Procs. of the First Dutch International Symposium on Linux, 1994.

3) Stephen C. Tweedie, *Journaling the Linux ext2fs Filesystem*, LinuxExpo '98, 1998.

4) http://lkst.sourceforge.net/

5) 「LKST（Linux Kernel State Tracer）によるカーネル性能評価ツールの開発」報告書, http://www.ipa.go.jp/software/open/forum/Contents/DevInfraWG/lkstprospec-0316.pdf