Vol.2011-ARC-195 No.4
Vol.2011-OS-117 No.4
2011/4/13

フォールトインジェクションを用いたアプリケーションのディスクエラーに対する耐性調査

西山貴彦[†1] 山田浩史[†1,†2] 河野健二[†1,†2]

ハードディスクの高密度化やファームウェアの複雑化に伴って，ディスクのエラーが起こりやすくなっている．ディスクエラーを適切に処理しないと，エラーを起こしたセクタ上のデータが他のデータを破壊したり，ディスク上に保持していたデータが消失してしまったりしてしまう．しかし，ディスクを管理するファイルシステムはディスクエラー対策は十分であるとは言い難く，ユーザレベルで稼働するアプリケーション自身でもディスクエラー対策を取らざるを得ない．本論文ではアプリケーションに対するディスクエラーへの耐性を調査する．MySQL を題材とし，5 種類のディスクエラーを挿入し，ディスクエラーを想定した実装となっているかどうかを検証する．ディスクエラーを想定した実装となっていない場合，どのような対策をとるべきかを検討する．データベースサーバのひとつである MySQL 5.1.53 を用いて実験を行ったところ，データベース内のデータが消失したり，付属の修復ツールを用いても修復できなくなることがわかった．また，多くの場合，MySQL がメモリ上で管理している情報を活用することで，ディスクエラー対策を施せることがわかった．

An Analysis of Application-Level Robustness against Disk Errors with Fault Injection

Takahiko Nishiyama,[†1] Hiroshi Yamada[†1,†2] and Kenji Kono[†1,†2]

Disk faults are critical to system reliability. If not handled appropriately, they cause data corruption and/or loss of data, leading to unacceptable service downtime or incorrect service operation. To mitigate their impacts, it is necessary and important to design software-level functionality for gracefully handling disk faults. However, according to some research reports, existing file systems, which manage disks, do not successfully handle disk faults. This fact motivates us to explore the possibility of application-level countermeasures against disk faults. In this paper, we investigate application behavior under disk faults. Specifically, we observe MySQL behavior when we inject five disk faults. We conduct a fault injection experiment with MySQL 5.1.3. The result shows that under our fault injection, data loss sometimes occurs and the corrupted data cannot be corrected even with the MySQL recovery tool. We also find that we can recover from the many situations by exploiting data structures MySQL keeps in memory.

1. はじめに

ハードディスクの高密度化やファームウェアの複雑化に伴って，ディスクのエラーが起こりやすくなっている．たとえば，あるセクタへのアクセスが失敗する latent sector エラー[1)]や，指定したセクタとは異なるセクタにデータを書き込む Misdirected write[1)] がある．これらが生じる原因のひとつに，ディスクが高密度化，高速化していることがある．高密度化のために磁気ヘッドと円盤の距離をより近づけ，高速化のためにより素早い動きを磁気ヘッドに要求するので，ディスクの信頼性を保つのは困難である．[2)] また，ファームウェアの巨大化，複雑化も挙げられる．巨大なファームウェアをバグを混入せず作成することは難しい．たとえば，Seagate 製のディスクのファームウェアは約 40 万行からなっている．[3)]

ハードディスクを改良してディスクエラーの発生を完全に防ぐことは非現実的なため，ディスクエラーを予め想定したソフトウェアの設計が求められる．もしディスクエラーを適切に処理しないと，エラーが発生したセクタ上のデータが他のデータを破壊したり，ディスク上に存在していたデータが消失したりしてしまうためである．特に，ディスク自身を管理したり大規模なデータを取り扱うソフトウェアではその被害は深刻となる．そのようなソフトウェアとして，ファイルシステムやディスクのセクタを直接変更するデフラグメンタ, ファイルシステムチェッカーなどの管理ツール，更にはデータベースに代表されるデータの信頼性が重要となるアプリケーションが挙げられる．もしデータベースソフトが書き込み時のエラーを検知しなければ，テーブルの内容やテーブルそのものの消失を引き起こす場合がある．

既存のファイルシステムはディスクエラー対策を十分に施されているとは言い難く，ユーザレベルで稼働するアプリケーション自身でもディスクエラー対策を取る必要がある．ファ

†1 慶應義塾大学
Keio University
†2 JST CREST

Vol.2011-ARC-195 No.4
Vol.2011-OS-117 No.4
2011/4/13

イルシステムを全面的に信用してアプリケーションを設計した場合，ディスクエラーはアプリケーションの管理するデータに容易に伝播してしまうためである．Prabhakaran ら[4),5)]の調査では，既存のファイルシステムは，ディスクの書き込みエラーを見逃してしまうことや破損したデータを修復できないことがあることを示している．調査結果を基にファイルシステムレベルでの対策方法を述べているが，既存のファイルシステムに新たな機構を正しく修正することは難しい．なぜなら，数千行にも及ぶコードの中で読み込みや書き込みといった処理のコードがソースコード中に点在しており，ディスクエラーの検知と修復のコードを適切な場所に挿入することが難しい．実際，ext3 においてスーパーブロックのコピーを作成しているが，修復には使用されていないといったことが知られている．

本論文では，ユーザレベルで稼働するアプリケーションのディスクエラーへの耐性を調査する．ディスクエラー耐性の必要なアプリケーションとして，代表的なデータベースである MySQL を題材とし，ディスクエラーに対する挙動を観察し，ディスクエラーを想定した設計になっているか，なっていない場合どのような対策をとるべきかについて検討する．また，ディスクエラーに対する挙動を観察するために，ディスクエラーを再現するフォールトインジェクタを作成し，文献[1)]に記述されているディスクエラーを挿入していく．

実験では，MySQL 5.1.53 を稼働させ，テーブルを構成するデータファイル，インデックスファイルに対してフォールトを挿入した．結果として，データベース内のデータが消失する事例や付属の修復ツールを用いても修復できない事例があり，ディスクエラーに対しての耐性が完全であるとは言えないことが判明した．挙動を調査した結果，多くの場合，MySQL がメモリ上で管理している情報を活用することで，テーブル全体に対してはディスクエラーに対する対策が施せる分かった．一方で，テーブルのデータ内容については対策が施されていないことが分かった．

本論文の構成を以下に示す．2 章ではどのようなディスクエラーが想定されるかについて説明する．3 章ではディスクエラーを想定するべきシステムの信頼性を調査している関連研究を紹介する．4 章ではフォールトインジェクタについて説明する．5 章ではフォールトインジェクタの実装について説明する．6 章では MySQL に対して行ったディスク故障に対する耐性調査とその結果について説明する．7 章にて今後の課題とまとめを述べる．

2. ディスクエラー

まず，ディスクエラーがおこる原因について述べる．最初に，ディスクの高密度化，高速化があげられる．磁気ヘッドが円盤を傷つけ，ディスクセクタにアクセスできなくなることがある．[2)] もう 1 つの原因としてはファームウェアの巨大化である．例えば現在の Seagate 社のディスクではファームウェアが 40 万行にも及ぶ[3)]．巨大なソフトウェアにおいてはバグを完全になくすことは非常に難しい．次に，ディスクにおいて市場が品質より価格を重視している点があげられる．現在，システムに高い信頼性が要求されるクラスタやサーバ環境においても，信頼性の低い安価なディスクが使われている．[6),7)]．これにより企業は品質よりも価格や容量を重視した開発を行ってしまう．

ディスクエラーの調査結果[1)]によると，ディスクに対してアクセスや書き込みを行う時に発生するエラーは主として 5 種類のエラーがあると述べている．この調査におけるディスクエラーでは，従来のフェイルストップなエラーと異なり，エラーが発生してもディスクがそのまま動作してしまうため，故障が他のセクタに伝搬したり，データが消失してしまう危険がある．それぞれのエラーの内容について説明する．

latent sector エラーは，データの読み込み時に特定のセクタのアクセスに失敗するというエラーである．このエラーが発生した場合にはファームウェアからエラーコードが返ってくるので，処理を要求したシステムにおいてエラーコードのチェックをすれば検出は可能である．このエラーは，32ヶ月の間に 153 万個のディスクのうち 3.45%のディスク上で発生したと報告されている．[8)]

lost write は，データの書き込み時に，要求された処理を実行せずに，通常の処理成功のステータスコードを返すというエラーである．このエラーでは，表向きには書き込みが行われているように見えるため，データを読み込み直してチェックしなければ発見ができない．

torn write は，データの書き込み時に，要求された処理のうち一部のみしか処理されないというエラーである．このエラーが発生した場合には，書き込み要求で指示された容量より少ない容量が返戻値になるのでエラーコードをチェックすれば発生自体は発見が可能であると言える．

misdirected write は，データの書き込み時に本来データが書かれるべきセクタと違うセクタにデータを書き込んでしまうというエラーである．このエラーは発生した場合でも，書き込み自体は成功しているため，特にエラーコードを返さないため，正しく書き込みができているかどうかを調べるためには，一旦読み込み直してデータの内容をチェックする必要があると言える．

データ破損は，名前の通りデータの一部の要素が破損するエラーである．データ破損としてよくある事例は，セクタのうち一部ビットが反転してしまうことである．データ破損の場合も特にエラーコードなどはでないので，データの内容をパリティビットなどでチェックし

Vol.2011-ARC-195 No.4
Vol.2011-OS-117 No.4
2011/4/13

なければ発見ができない．

3. 関連研究

ファイルシステムの耐性調査[4] では，実際にファイルのメタデータにフォールトを挿入し，既存のファイルシステムの信頼性を検証している．この調査では，ext3[9], Reiser FS, IBM JFS を対象としている．これらのファイルシステムにおいて，各ファイルのメタデータに対してディスクエラーを再現したフォールトを挿入している．フォールトを挿入後，ファイルシステムの API を実行し，挙動を観察している．結果を，挿入されたフォールトを検知する過程と，検知した後に修復する過程に分け，メタデータの種類別に分類している．結果として，ファイルシステムにおいて，ディスクエラーに対する耐性が万全でないことが報告されている．

この研究と本研究を比較をすると，ファイルに対して実際にフォールトを挿入してディスクエラーに対する耐性の調査を行っている点は同様である．一方でファイルシステムは巨大なため修正自体も困難である．そのため，アプリケーションでも対策が必要と言える．よって本研究では調査対象をファイルシステムではなく，ユーザ空間で動作するアプリケーションとしており，異なる点と言える．

ファイルシステムチェッカーの信頼性についての研究[10] では，ext2 のファイルシステムチェッカーである fsck の信頼性について検証を行っている．結果として fsck には動作に様々な問題点があり，fsck を動作させることによって，却ってデータを破壊してしまうことがあることが報告されている．

この研究を本研究と比較をすると，ファイルにフォールトを挿入してディスクエラーに対する耐性の調査を行っている点は同様であるが，調査対象をファイルシステムチェッカーではなく，ユーザ空間で動作するアプリケーションとしている点が異なる点と言える．

4. フォールトインジェクタの設計

アプリケーションの利用するファイルへのフォールトの挿入を容易にするため，新たにフォールトインジェクタを設計した．従来のフォールトインジェクタの多くは，ファイルシステムやディスクを直接書き換えるツールであるために，フォールトを挿入するセクタ番号を直接指定するものになっている．そのため，アプリケーションの利用するファイルやそのメタデータにフォールトを挿入する場合，対応するセクタ番号を調べる必要があり，利便性が高いとはいえないものとなっている．

そこで本研究で作成するフォールトインジェクタは，パス名やデータもしくはメタデータの種類によってフォールトの挿入位置を指定可能にする．例えば，ファイル A の inode を読み込めなくする，ファイル A のデータのうち 10byte 目を破損させるといったものである．このような指定を可能にすることによって，ユーザが意図したファイルの位置にフォールトを挿入することが可能となる．

フォールトインジェクタで再現するエラーは，2 章にて紹介を行った，ディスクエラーの調査結果[1] である latent sector エラー, lost write, torn write, misdirected write, データ破損の 5 つとした．

それぞれの再現方法を説明する．latent sector エラーはアプリケーションからみると，データが読み込めずエラーコードが返されるので，システムコールにおいてデータブロックや inode にアクセスするシステムコールをフックし，指定条件に合った呼び出しはエラーコードを返し，その他の場合にはそのまま処理を続けるようにしている．

lost write はアプリケーションからみれば，エラーが発生した場合にもエラーコードを返さず正常に処理されたように見えるが書き込みに失敗していることになる．よって指定範囲を含む書き込み処理を行わずに，本来書き込まれるはずの容量を返すという再現を行っている．

torn write はアプリケーションからみると，書き込み容量が減少しているのでエラーコードが返される．よって指定した範囲への書き込みがきた場合には，その部分への書き込みは処理をしないようになっている．またこれによって実際に書かれる容量が減るので，その容量を返すようになっている．

misdirected write はアプリケーションからみれば，発生した場合でも正常に処理されたようにみえたまま書き込み先を誤っていることになる．よって書き込みを失敗させるページを指定して，指定先もしくはランダムな場所へ書き込み先を変更させるようにしている．また，その際ページ内でのオフセットはそのままとなる．例えばページサイズを 4096 bytes とすると，0 ページ目の書き込みを失敗させて，2 ページ目に書き込む指定をした場合には，ファイルの先頭から 4096 bytes 以内の領域への書き込みが 4096 * 2 + 元のオフセットへ書き込まれることとなる．最後に正常に処理された場合のステータスコードを返す．

データ破損は，ユーザの指定したファイル内のセクタの値を書き換えることによって再現を行う．

4.1 フォールトインジェクタの動作

次にフォールトインジェクタの動作について述べる．動作イメージを**図 1** に示した．今

Vol.2011-ARC-195 No.4
Vol.2011-OS-117 No.4
2011/4/13

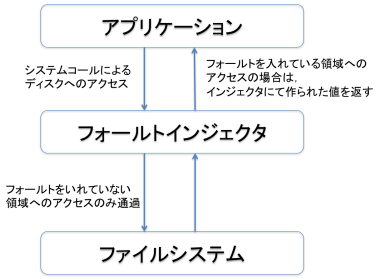

図 1 フォールトインジェクタの動作イメージ
Fig. 1 functioning of Fault Injecter

回作成したフォールトインジェクタでは，システムコールによる仮想ファイルシステムへのアクセス動作をフックして動作する．この設計にするメリットは，仮想ファイルシステム上の処理をフックしているため，実際のファイルシステムに依存せずに使えることとカーネルの書き換え箇所がほぼなく，コンパイルし直さなくても後からモジュールをロードするだけで使用可能になることである．例えば，読み込み時にフォールトを挿入する場合には，read システムコールが呼び出されたら，フォールトインジェクタにてパス名やデータ構造の指定に当てはまるかの確認を行い，read システムコールをそのまま実行するか，エラーコードを返すかを決定する．このようにファイルの読み込み，書き込みを行うファイルシステム API へのアクセスをフックする. アプリケーションにおいてはディスクへのアクセスに対して必ずシステムコールを介してファイルシステムを使用するので，システムコールからアプリケーションへ渡す値にフォールトを挿入することによって，アプリケーションに対してディスクエラーの再現したフォールトの挿入を実現することになる．

5. フォールトインジェクタの実装

5.1 概　　要

今回のフォールトインジェクタは Linux 2.6.36.1 上に実装した．モジュールは 2 つに分かれており，データ破損用モジュールと，その他のフォールトを再現するシステムコールフックモジュールとなっている． データ破損モジュールでは，ユーザが指定したファイルに対してデータ破損のフォールトを挿入し，システムコールフックモジュールでは，ユーザが指定したファイルに対して，latent sector エラー, lost write, torn write, misdirected write のいずれかを挿入する．

5.2 データ破損用モジュール

最初に，フォールトを挿入したいファイルのパスを指定する．フォールトの挿入先はデータブロック, inode, 実際のデータの位置を格納する i_block を選択できるようになっている．フォールトを挿入する場所の指定は，データブロックにおいては，データの先頭からのバイト数もしくはランダム，inode においては，メンバ名と対応した数字，先頭からのバイト数もしくはランダムを指定できるようになっている．このようにすることで例えば inode のサイズが入ったメンバを書き換えるといった指定を可能としている．また，ファイルシステム全体のブロックのいずれかにランダムに書き込む機能も実装している．書き込む値については，バイト単位での入力で指定値，全ビット反転，乱数の 3 パターンが使えるようになっている．これにより，どのようなデータ破損のパターンでも再現することが可能になる．

実際の動作を説明する．最初に，ユーザからパラメータの指定を受け，パス名からファイルを開く．ファイルを開くことにより，inode が取得できるので，inode 内のデータのメモリマッピングを参照して指定された書き換え先を開き，値を書き換えるといった物である．取得したバッファキャッシュから，実データの入っているセクタを取得し値の書き換えを行っている．

5.3 システムコールフックモジュール

データ破損以外のフォールトを扱うシステムコールフックモジュールについて説明する．このモジュールでは，latent sector エラー，misdirected write, torn write, lost write を挿入可能としている．今回の実装はシステムコールの呼び出しをフックすることによって行っている．

パラメータの指定はパス名，挿入するフォールトの種類，挿入するデータ構造 (データブロックもしくは inode)，データブロックの場合は挿入範囲の 4 つを指定する．データ破損用モジュールと違い，読み込みもしくは書き込み処理がよばれる度にフォールトを挿入するため，挿入するフォールト挿入位置を保持する必要があるので，ユーザからのフォールトの指定を保持しつつ，新しい指定されたパラメータを追加もしくは上書きするようにする．

次にそれぞれのフォールトの実装方法について説明する．latent sector エラーは，データブロックの一部もしくは全体，inode の全体を読み込み不可とする指定をすることができる．このエラーはセクタに対するエラーであるため，範囲指定は自動的にセクタサイズ単位に修正される．またデータの場合は指定の範囲を含む読み込み命令がきた場合に，全体の処理を無効にするのではなく，指定範囲部分のみを読み込まないようにして，エラーコードを返すようにしている．そうでない場合は正常な返戻値を返す．

Vol.2011-ARC-195 No.4
Vol.2011-OS-117 No.4
2011/4/13

lost write では，データブロックにおいては書き込みを失敗させる範囲を指定し，範囲内へ書き込み処理を行わないようにしている．実際の動作は， まず latent sector エラーと同様に，書き込み先ファイルが，フォールトの挿入先と一致しているかどうかを調べる．次にフォールトが入っている領域への書き込みの場合は，実際の処理は行わないが正常に書き込まれたようにアプリケーションに対して見せるので，書き込む領域とフォールトの挿入されている領域が重なっていた場合には処理を行わずに書き込み要求のサイズである count を返すようになっている．

torn write の動作は，latent sector エラーと非常に似ている．違いとしては，読み込みの代わりに書き込みを行う点と，返戻値に実際に書き込まれた容量を返すことである．

misdireted write の動作は，初めに write 関数が呼び出されたときに，書き込み先ファイルがフォールトを挿入先のファイルか調べる動作は，lost write と同様である．次に，ファイルポインタの書き込み開始地点が，フォールトの入っているページかどうかを調べている．今回の実装では，フォールトの入っているページを含んでいる処理全体を違うページに書き込ませている．そのため，書き込みの先頭位置を修正すれば，あとは他のフォールトと同様にメモリ内に書き込む処理を行えば良いことになる．

このように，計 5 つのエラーを再現し，パス名やバイトによる挿入位置の指定によってフォールトを適用することが可能となっている．

6. アプリケーションの耐性調査

6.1 調 査 方 法

今回は，データの信頼性が重要となるアプリケーションのうちデータベースとして広く使われている MySQL 5.1.53 を対象とし耐性を調査した．**表 1** に実験環境を示す．MySQL でテーブル作成時に生成されるデータファイル，インデックスファイルに対してフォールトを挿入し，MySQL を起動して，エラーを検出できるかを調べる．但し misdirected write では，フォールトによって書き込まれる書き込み先が，既存のデータを上書きしない位置にしている．MySQL 起動前にフォールトを挿入する理由は，MySQL においては，インデックスファイルをメモリに保持し続けていて，これによって，起動時に正常にデータにアクセスできた場合には，その後の 2 ファイルに対するアクセスの結果に関わらず，MySQL を終了するまでは正常に動作するためである．

破損を検出できた場合には，その時点で破損したファイルに対して MySQL に付属の修復ツールである mysqlcheck の修復オプションを用いて対象ファイルを修復し，どのような修復が行われたかを調べた．今回はそれぞれのファイルに対してフォールトを挿入して MySQL を起動した後，latent sector エラー以外では insert 文によるデータの追加を行い，全てのフォールトに対して select 文によるテーブル全体の読み出しを行った．今回の調査は仮想マシン上において行った．

表 1 マシンの構成
Table 1 computer configuration

CPU	Intel Core2 Duo T9900
メモリ	DDR3 8GB
ディスクデバイス	APPLE SSD TS256A
オペレーティングシステム	Mac OS X 10.6.6
仮想マシンモニタ	VMware Fusion 3.1.2
オペレーティングシステム	linux 2.6.36.1
ファイルシステム	ext3
MySQL	5.1.53

MySQL のデータファイル，インデックスファイルは次のような構造になっている．データファイルの内部データは，各データに対して囲みの前後 1 バイトとデータ長分の領域が確保される．例えば int 型と char 型が 1 列ずつあるテーブルの場合は，データ 1 行につき，データの始めを示す 1 バイト，int 型の 4 バイト，char 型の 1 バイト，データの終わりを示す 1 バイトの計 7 バイトが確保される．

インデックスファイルにおいては MySQL ドキュメントとソースコードにてインデックスファイルの内部構造の調査をおこなった．インデックスファイル内のバイナリ格納位置を把握できたメンバのうち，データの追加によって値が書き換えられ，値が数値をそのまま表したものになっているメンバについてデータ破損を挿入し検証を行った．このようなメンバは 7,8 バイト目のインデックスファイル内で実際にデータの入っている領域を表した header_length とデータファイルのサイズを表した 73 から 80 バイト目までに格納されている key_file_length であった．

6.2 調 査 結 果

6.2.1 データファイルにフォールトを挿入したときの調査結果

最初にファイルに対して，実装した 5 つのうちデータ破損以外の 4 フォールトを挿入した場合の調査結果について述べる．結果を**表 2** に示す．

データファイルにおいては，メモリ上にデータを保持していないため，フォールトの挿入種類によってデータが破損してしまうことがある. ディスクエラーが発生した場合の対応は完全ではなく，ディスクエラーが発生した場合には，データが消失する危険性が高いとい

Vol.2011-ARC-195 No.4
Vol.2011-OS-117 No.4
2011/4/13

表 2 データファイルに対する調査結果
Table 2 result of the search for data file

挿入したフォールト	書き込み結果	読み込み結果	修復結果
latent sector エラー		+	—
lost write	*	+	—
torn write	+		—
misdirected write	*	+	+

書き込み		読み込み		修復	
記号	説明	記号	説明	記号	説明
*	エラー検知せず	*	エラー検知せず	*	破損検知せず
—	検知して処理続行	—	検知して処理続行	—	修復しデータ消失
+	検知して処理停止	+	検知して処理停止	+	修復しデータ復活
○	エラー発生せず	○	エラー発生せず	/	破損を検知し修復不可

える．

それぞれの結果をみていく．latent sector エラーを挿入した場合には，select 文を実行した際に，エラーを検出し，テーブルの内容を表示することが不可能になった．一方で，データの修復を行った場合には，修復はできたが，テーブルを表示してみるとデータが消失し，空のテーブルとなった．つまり，データファイルが読み込めなくなった場合には，そのままデータが消失してしまう．そのため，このような事態を避けるために予め 2 ファイルに書き込むなど対策が必要なのではないかと考えられる．

lost write を挿入しデータを追加を行うと，データが正しくファイルに書き込まれていないのにも関わらず，エラーを検知しなかった．追加後のテーブルの内容を表示しようとすると，エラーが検出され，テーブルの内容が表示できなかった．つまり，データの書き込みに失敗した時点で，追加したデータは消失しているということであり，ディスクエラーを考えていないといえる．書き込んだ内容を照合して，正しいディスクに書き込まれたことを確認するまではメモリに内容を保持するなど対策が必要といえる．

torn write を挿入し，データの追加を行うと，書き込み時にエラーが検出され処理が停止した．エラーが検知されたので修復を行ったところ，修復は完了したが，追加したデータは消失した．この結果からエラーコードはチェックしていることが分かるが，データが消失するという点では lost write と同様であり，書き込み完了まではメモリに内容を保持し，再度書き込みを行うことや，書き込んだ内容を照合するなどの対策が必要と考えられる．

misdirected write では，書き込み時にはエラーが検出されず，テーブルの表示を行ったときにエラーが検出され，内容を表示できなかった．修復を行ったところ，修復は完了し，追加されたデータも正常に読み込むことができた．このことから，データがディスク上に書き込まれている場合には書き込み位置がずれていても修復は可能になっている．

表 3 インデックスファイルに対する調査結果
Table 3 result of the search for index file

挿入したフォールト	書き込み結果	読み込み結果	修復結果
latent sector エラー		+	/
lost write	*	*	—
torn write	+		/
misdirected write	*	*	—

6.2.2 インデックスファイルにフォールトを挿入したときの調査結果

次にインデックスファイルにおける調査結果について**表 3** に示す．インデックスファイルにおいては，読み込みに失敗した場合には，テーブル自体の復旧も不可能になってしまう．また，書き込み時のエラーはファイルへの変更が反映されないことによって，テーブルの内容そのものにも影響を及ぼし，データファイル上に内容があったとしても，表示ができないということとなる．また，修復においては，インデックスファイルを用いて修復するため，このファイルが破損した場合は修復自体ができなくなる可能性が高くなると言える．ただし，MySQL を終了するまでは書き込みを行った内容がメモリ上にあるので，そこまでの段階でチェックできるようにすれば発見が可能になると言える．

それぞれの結果を見ていく．latent sector エラーを挿入した場合には，テーブル読み込み時にエラーを検出し，内容が表示できなかった．また修復操作もフォールトは検出するが，復旧することは不可能であった．つまり，インデックスファイルが読み込めなかった場合には，テーブルそのものへのアクセスができなくなってしまうため，データを取り出すにはデータファイルのバイナリから手動でテーブルを作り直すしか方法がなくなってしまう．インデックスファイルが破損した場合の対策として，一時的にデータファイルのみを用いてテーブルを見れるようにするか，データファイルやテーブル定義ファイルからテーブルのデータを取り出せるツールが必要なのではないかと言える．

lost write を挿入した場合には，MySQL を再起動するまでは，書き込み時にエラーは発生せず，テーブルの表示も追加したデータを含めて正常に行われる．しかし，MySQL を再起動すると追加したデータはテーブルに表示されなかった．修復操作を行うと，修復は完了するが追加したデータは表示されないままとなった．これは，書き込み時にエラーコードをチェックするのみで，書き込んだ内容をチェックしないために起こることである．インデックスファイルの場合には MySQL がメモリ内に内容を保持しているので，書き込んだ内容の照合を行う必要があるのではないかと考えられる．

Vol.2011-ARC-195 No.4
Vol.2011-OS-117 No.4
2011/4/13

表 4 データファイルに対するデータ破損挿入による調査結果
Table 4 result of the search for data file by corruption

データ破損挿入先	読み込み結果	修復結果
実データ部	*	*
行区切り部	*	*

torn write を挿入すると，書き込み時にエラーが発生し，処理が停止した．修復操作を行うと，破損していることは検知されるが，修復がエラーとなった．こちらの場合には，エラーコードによって発生を検知することはできている．また，メモリ上にデータを保持しているので再度処理を行うことも可能である．そのため，書き込みに失敗した領域とは別の領域にインデックスファイルを作り直して，書き込みを再び行えば torn write への耐性を持つことができる．

misdirected write を挿入すると，書き込み時，読み込み時ともエラーは検出されないが，読み込み時にはテーブルに追加した内容が反映されなかった．修復は，修復自体は完了するが，追加した内容は消失したままであった．インデックスファイルの場合には，予め値が入っているため，データブロックと違い，別の場所に書かれた部分をそのまま結合することはできない．よって，書き込んだ内容をチェックして，正しくインデックスファイルを更新できたかを確かめる必要があると言える．

6.2.3 データ破損挿入による調査結果

次にファイルに対してデータ破損を挿入した場合の動作結果について示す．まずデータファイルにデータ破損を追加した結果を示す．まず int 型，char 型の実データ部にあたる 2 バイト目から 6 バイト目を乱数によって書き換え，テーブルで表示した結果，エラーを検出せず，書き換えられた値がそのまま表示された．つまり，データの内容は破損によって消失することになる．

次に 1 バイト目と 7 バイト目について同様に乱数にて書き換えを行い，テーブルを表示したところ，エラーはでなかったが，改変された行のデータが乱数の値によって，片方もしくは両方の列が NULL と表示され，内容を確認できなかった．またどちらの場合にも修復ツールにおいて破損していることを検出できず，修復はできなかった．

データファイルに対する結果を**表 4** にまとめている．データファイルにデータ破損を行った結果についてまとめる．データファイルでは，実データの内容はチェックを行っていないことが判明した．また，実データ以外のヘッダ部分に関しては，データ破損に対するチェックは行っているが，表示がおかしくなることやデータ消失につながることが判明した．また修復に関しては，データ内容を復旧することは不可能であることが判明した．全体としては，データファイルではデータ破損に対しての対策は行っていないと言える．データ破損への対策としては，データの内容そのものについてのチェックを行うべきである考えられる．

次に，インデックスファイルに対して行ったデータ破損の挿入結果をまとめる．インデックスファイルにおいては，データ破損が起きた場合，MySQL にてその他のファイルとのデータの相違をチェックし，データ破損への対策が行われているということが判明した．

header_length メンバに対して，データ破損を挿入したところ，フォールトを挿入する前の値より小さい値に書き換えられた場合に限って，テーブルの表示の際に，エラーが検出され，テーブルの内容が表示できなくなった．修復を行うと，通常の修復オプションのみでは，修復が不可能であった．テーブル定義ファイルを修復に利用する –use-frm オプションを用いると正常に修復が行われ，データ内容の消失などはなかった．

key_file_length に対してデータ破損の挿入を行うと，テーブルの内容表示の際にエラーが検出され，内容を表示できなかった．修復操作を行ったところ，正常に終了し，データ内容の消失はなかった．

6.3 調査結果のまとめ

調査結果をまとめる．MySQL では，latent sector エラーの検知をし，処理を停止することで，エラーの伝搬を防いでいる．一方で，修復ツールを用いても，データファイルの場合は，データが消失し，インデックスファイルの場合には，修復自体がエラーとなってしまいテーブルが表示できなくなるという問題がある．予め 2 ファイルに書き込むなどの対策が必要であると言える．

書き込みに関するフォールトについては，MySQL では対策が施されていないと言える．lost write に対しては，エラーの検知自体ができない．これは，現状ではエラーコードのみをチェック対象としているためである．torn write については，エラーコードを返すため，エラーの検知は行うことができるがそれによる動作は処理を停止するのみで，修復ツールを用いてもデータを復旧することはできない．misdirected write についてはエラーの検知自体ができない．

この 3 つのフォールトへの対策としては，書き込みが正常に行われたことを確認するまでは，メモリに内容を保持する，書き込みを行った後に書き込まれた内容を読み込み直すことによってチェックを行う，正常に書き込まれていない場合は再度書き込みを行うといったことが挙げられる．

データ破損については，ヘッダ部分のデータ破損は検出可能であるが，データファイル

Vol.2011-ARC-195 No.4
Vol.2011-OS-117 No.4
2011/4/13

の場合には，修復ツールを用いても復旧が不可能となっている．データ破損についても，2ファイルに書き込むことは良い対策と考えられる．

7. まとめと今後の課題

7.1 ま と め

本研究では，アプリケーションに対してフォールトを挿入しやすいように設計したフォールトインジェクタを作成し，アプリケーションのディスクエラーに対する耐性を調査した．今回作成したフォールトインジェクタでは，latent sector エラー, lost write, torn write, misdirected write, データ破損を再現し，ファイルシステムの情報を用いて，ファイル名や先頭からのバイト数によるフォールト挿入を可能にした．

フォールトインジェクタを Linux 2.6.36.1 上に実装し，MySQL 5.1.53 のディスクエラーに対する耐性を調査した．方法としては，実際にテーブルを作成し，テーブルを構成するデータファイル，インデックスファイルに対し，実際にフォールトを挿入し，データの追加，テーブルの読み込みを行って，実行結果の観察とファイルの修復を行い結果を記録した．

調査の結果，MySQL においては，データファイルやインデックスファイルへの書き込みの失敗がそのままデータの消失につながったり，読み込めなくなったデータは修復も不可能であったり，ディスクエラーに対する対策が完全ではないことが判明した．また，データ破損に対しても MySQL では対策がとれていないため，データが消失してしまう可能性が高くなっていることが判明した．

7.2 今後の課題

今後の課題としては以下のようなものが挙げられる．第一に MySQL 以外のアプリケーションに対しても調査を行うことである．MySQL においては，主にデータの追加の際のディスクエラーに対する脆弱性が明らかになったが，このことが他のデータベースでも共通であるのかどうかを調査によって確かめる必要がある．またデータベース以外にも信頼性が求められるアプリケーションは，バックアップツールなどがある．そちらの方についても調査を行っていくべきであると考えている．

次に，ユーザ空間で動作するアプリケーション以外への調査も行うことである．対象としては，既に検証が行われているファイルシステムやファイルシステムチェッカー以外にも，デフラグツールなどの直接ディスクにアクセスするツールについて調査を行っていきたいと考えている．またその際には，ディスクに直接フォールトを入れることに特化したフォールトインジェクタの作成が必要ではないかと考えている．

参 考 文 献

1) Krioukov, A., Bairavasundaram, L.N., Goodson, G.R., Srinivasa, K., Thelen, R., Arpaci-Dusseau, A.C. and Arpaci-Dusseau, R.H.: Parity Lost and Parity Regained, *Proceedings of 6th USENIX Conference on File and Storage Technologies (FAST '08)*, pp.127–141 (2008).
2) The Data Clinic: Hard Disk Failure. `http://www.dataclinic.co.uk/hard-disk-failures.htm`.
3) Dykes, J.: *A modern disk has roughly 400,000 lines of code*, Personal Communication from James Dykes of Seagate (2007).
4) Prabhakaran, V., Bairavasundaram, L.N., Agrawal, N., Haryadi S.Gunawi, A. C. A.-D. and Arpaci-Dusseau, R.H.: IRON File Systems, *Proceedings of the twentieth ACM symposium on Operating Systems Principles (SOSP'05)*, pp.206–220 (2005).
5) Prabhakaran, V., Arpaci-Dusseau, A.C. and Arpaci-Dusseau, R.H.: Model-Based Failure Analysis of Journaling File Systems, *Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN'05)*, pp.802–811 (2005).
6) Gunawi, H. S., Agrawal, N., Arpaci-Dusseau, A. C., Arpaci-Dusseau, R. H. and Schindler, J.: Deconstructing Commodity Storage Clusters, *Proceedings of the 32nd Annual International Symposium on Computer Architecture (ISCA '05)*, pp.60–73 (2005).
7) Ghemawat, S., Gobioff, H. and Leung, S.-T.: The google file system, *Proceedings of the nineteenth ACM symposium on Operating systems principles (SOSP '03)*, pp.29–43 (2003).
8) Lakshmi N. Bairavasundaram, Garth R. Goodson, Shankar Pasupathy, and Jiri Schindler: An analysis of latent sector errors in disk drives, *Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pp.289–300 (2007).
9) S. C. Tweedie: Journaling the Linux ext2fs File System (1998). In The Fourth Annual Linux Expo.
10) Gunawi, H.S., Rajimwale, A., Arpaci-Dusseau, A.C. and Arpaci-Dusseau, R.H.: SQCK: A Declarative File System Checker, *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI '08)*, pp.131–146 (2008).