情報処理学会論文誌　データベース　Vol.12　No.1　17–23 (Jan. 2019)

テクニカルノート

SQLiteのファイル固定長化による更新処理時間の評価

藤井 雄規[1,a)]　水口 武尚[1]　樋口 毅[1]

受付日 2018年6月8日, 採録日 2018年8月3日

概要：組込みシステムにて，逐次発生するデータを SQLite に蓄積していくアプリケーションが多くあるが，データベースの更新処理時間の短縮が重要な課題として指摘されている．Linux 上で SQLite を利用する場合，データを格納するデータベースファイルやジャーナルファイルの領域を固定化する，ファイル固定長化が有効である．本研究では，SQLite の実装を基にファイル固定長化による更新処理時間の短縮効果とファイル固定長化で定義するデータサイズの 2 つについて机上検討により見積り式を作成した．この 2 つのうち，更新処理時間の短縮効果の見積り式については，実機評価により妥当性を確認した．

キーワード：組込み向け DBMS，SQLite，トランザクション，チェックポイント，ファイル固定長化，ext4

Evaluation of the Update Response Time by Preallocating Storage Area for SQLite's Files

Yuki Fujii[1,a)]　Takehisa Mizuguchi[1]　Tsuyoshi Higuchi[1]

Received: June 8, 2018, Accepted: August 3, 2018

Abstract: There are many embedded applications storing successive data in SQLite. It is an important challenge to shorten the database update time using such applications. When we use SQLite in Linux, it is effective to preallocate storage area for a database file or a journal file. In this work, we have studied an effect of the shortening update time by this preallocating and data size to preallocate, on the basis of SQLite implementation. We also have evaluated this effect to confirm the validity of our study.

Keywords: Embedded Database, SQLite, Transaction, Checkpoint, Preallocating Storage Area, ext4

1. はじめに

OSS の組込み向け RDBMS SQLite [1] はスマートフォン等にて標準的な DBMS として採用されている [2]．逐次発生するデータを SQLite に蓄積していく組込みシステムのアプリケーションが多くある [3], [4] が，SQLite ではトランザクションのサポートによりデータベースの更新処理にて多数のストレージ書込みが発生するため，更新処理時間の短縮が重要な課題として指摘されている [5]．

Linux 上で SQLite を利用する場合，更新処理時間を短縮する手段の 1 つとしてデータベースファイル（以下，DB ファイル）やジャーナルファイルの固定長化が知られている [6], [7], [8]．SQLite のデフォルトの設定では fdatasync システムコール（サポートされている場合）により更新を行うが，ファイルサイズの変更がある場合はメタデータの更新が発生する．ファイルを固定長化した場合はファイルサイズの変更がないためメタデータの更新が省略され，更新処理時間を短縮できる．

一方でこれまでの取組みでは，ファイル固定長化による更新処理時間の定量的な短縮効果およびファイル固定長化で定義するデータサイズの見積り方法は明らかにされていなかった．本研究では SQLite3.18 の実装を基に，ファイル固定長化による更新処理時間の短縮効果およびファイル固定長化で定義するデータサイズの 2 つについて机上検討により見積り式を作成した．この 2 つのうち，更新処理時間の短縮効果については実際の処理時間からストレージ種別ごとの特性に依存する部分を除外したものについて見積

[1] 三菱電機株式会社情報技術総合研究所
Information Technology R&D Center, Mitsubishi Electric Corporation, Kamakura, Kanagawa 247–8501, Japan
[a)] Fujii.Yuki@df.mitsubishielectric.co.jp

り式を構築することを目的とし，実機評価により妥当性を確認した．本研究では冒頭のアプリケーションの多くが満たすと考えられる，以下の条件を想定した．

想定（1）OS は fdatasync システムコールをサポートする Linux であり，ファイルシステムは ext4 である．
想定（2）1 トランザクションの更新範囲は数ページである．
想定（3）データの削除や更新よりもデータの投入が多い．
想定（4）データの投入のほとんどがテーブルの主キーに関して昇順である．
想定（5）同じページが何度も更新される．
想定（6）トランザクションの ACID 特性が求められる．
想定（7）レコードサイズは 1 ページに収まる程度である．
想定（8）平均レコードサイズを把握できる．
想定（9）最大レコード数が決まっており，必要に応じて古いデータの削除等を行う．
想定（10）更新処理時間は最悪値が重視される．

SQLite のジャーナリングの動作は undo ベースのロールバックジャーナルモード（デフォルト設定）と redo ベースの WAL モードの 2 つがあるが，本研究では各モードについてファイル固定長化を行う場合を想定し，2 つのモードの比較を行う．SQLite や ext4 は特に断らない限りデフォルトの設定で利用することを想定する．

2. SQLite の実装

2.1 SQLite が利用するファイル

SQLite は，永続化が必要なデータを DB ファイルおよびジャーナルファイルにて管理する．DB ファイルは 4 KiB のページ単位で管理され，テーブルやインデックスのデータを管理するページ，データの削除により空となった空ページ，ファイル先頭のヘッダのページからなる．ヘッダのページには全ページ数等が格納される．テーブルやインデックスは B 木として管理され，B 木の各ノードが 1 つのページとなる．各ページに格納される最小レコード数は，テーブルの場合 1 でインデックスの場合 4 である．テーブルの B 木のキーはテーブルの主キーであり，インデックスの B 木のキーはインデックスのキーである．テーブルやインデックスのページ内は，レコードの領域（本論文ではレコードの位置を示す 2B のオフセットも含める），未使用の空領域，B 木の子ページへのポインタ等の付加情報の領域（1%未満であり無視できる程小さい）がある．

ロールバックジャーナルモードの場合，ジャーナルファイルは 512B のヘッダの領域の後に更新前の状態のページと 8B の付加情報とがセットになったログが 0 個以上続く構成となっている．WAL モードの場合，ジャーナルファイルは 32B のヘッダの領域の後に更新後の状態のページと 24B の付加情報がセットになったログが 0 個以上続く構成となっている．ヘッダ部分には次のチェックポイントで DB ファイルに反映させるページを識別するための識別子（以下，チェックポイントの識別子）等が格納されている．どちらのモードもジャーナルファイルに対してはログが追記されていく構造となっているため，ジャーナルファイルの更新はシーケンシャルな書込みとなる．

```
Input: journal_mode, journal_path, database_file,
       current_directory
 1: // step1: open journal file
 2: journal_file = open(journal_filepath, O_CREATE)
 3:
 4: // step2: append journal log
 5: if (journal_mode = wal) and is_first() then
 6:    journal_header = new checkpoint identifier
 7:    write(journal_file, journal_header)
 8:    fdatasync(journal_file)
 9: end if
10: for each journal_log do
11:    write(journal_file, journal_log)
12: end for
13: if journal_mode = persist then
14:    fdatasync(journal_file)
15: end if
16:
17: // step3: update journal header
18: if journal_mode ≠ wal then
19:    journal_header = number of journal logs
20:    write(journal_file, journal_header)
21: end if
22:
23: // step4: update journal file
24: fdatasync(journal_file)
25: if journal_mode = delete then
26:    fdatasync(current_directory)
27: end if
28
29: // step5: update database file
30: if journal_mode ≠ wal then
31:    for each dirty_page do
32:       write(database_file, dirty_page)
33:    end for
34:    fdatasync(database_file)
35: end if
36:
37: // step6: invalidate journal file
38: if journal_mode = delete then
39:    unlink(journal_filepath)
40: else if journal_mode = persist then
41:    journal_header = zero values
42:    write(journal_file, journal_header)
43:    fdatasync(journal_file)
44: else if journal_mode = truncate then
45:    length = 0
46:    ftruncate(journal_file, length)
47:    fdatasync(journal_file)
48: end if
```

図 1　SQLite のトランザクションにおけるジャーナリングの動作
Fig. 1　Algorithm for Journaling in SQLite's Transaction.

2.2 SQLite のジャーナリングの動作

ジャーナリングのモードは DELETE（デフォルト設定），PERSIST, TRUNCATE, WAL の 4 つの設定パラメータがあり，DELETE, PERSIST, TRUNCATE の場合はロールバックジャーナルモード，WAL の場合は WAL モードで動作する．SQLite のトランザクションにおけるジャーナリングの動作を擬似コードで表現したものを**図 1** に示す．図 1

では [9] で指摘されている，ext4 上で SQLite を利用する場合に不要なシステムコールの呼び出しは無効化されていると仮定している．以下，図 1 を基に SQLite のジャーナリングの動作を説明する．*open*，*write*，*ftruncate*，*fdatasync* は各々 Linux のシステムコールを表している．ステップ 1 では，open システムコールによりジャーナルファイルを開く．DELETE ではジャーナルファイルを新規作成する．

ステップ 2 では，write システムコールによりジャーナルファイルにログを追記する．WAL では，前回のチェックポイントから 1 回目のトランザクションの場合，チェックポイントの識別子を write システムコールと fdatasync システムコールにより更新する処理が追加で発生する．*is_first* は，前回のチェックポイントから 1 回目のトランザクションかどうかを判定する関数である．PERSIST では，ログを追記した後に fdatasync システムコールの呼び出しが追加で発生する．後述するように PERSIST ではファイルサイズの変更がないため，この fdatasync システムコールの実行でメタデータの更新は発生しない．

ステップ 3 はロールバックジャーナルモードの場合のみ実行する．ステップ 3 では，write システムコールによりジャーナルファイルのヘッダにログ数を書き込む．

ステップ 4 では，fdatasync システムコールにより更新内容をストレージに反映する．ストレージに反映されるのは，PERSIST ではジャーナルファイルのヘッダであり，WAL ではログであり，DELETE および TRUNCATE ではジャーナルファイルのヘッダとログの両方である．WAL ではジャーナルファイルはデータベース接続終了時に削除されるため，データベース接続直後から 1 回目のチェックポイントまでは本ステップでファイルサイズが増加し，メタデータの更新が発生する．DELETE および TRUNCATE では，後述するように前回のトランザクション終了時にジャーナルファイルが削除されているか空ファイルであるため，本ステップでジャーナルファイルのサイズが増加し，メタデータの更新が発生する．

ステップ 5 はロールバックジャーナルモードの場合のみ実行し，write システムコールにより更新後のページ（以下，ダーティページ）すべてを DB ファイルに書込み，fdatasync システムコールにより更新内容をストレージに反映する．DELETE では，ディレクトリに対しても fdatasync システムコールを実行する．DB ファイルのサイズが増加する場合はメタデータの更新が発生する．

ステップ 6 はロールバックジャーナルモードの場合のみ実行し，ジャーナルファイルの無効化を行う．DELETE では unlink システムコールによりジャーナルファイルを削除し，PERSIST では write システムコールおよび fdatasync システムコールによりジャーナルファイルのヘッダを 0 クリアし，TRUNCATE では ftruncate システムコールおよび fdatasync システムコールにより空ファイルとする．

WAL の動作に関して補足する．WAL では，ステップ 5 の処理はチェックポイントで実行する．チェックポイントのタイミングは，トランザクションによりジャーナルファイルログ件数が 1,000 以上（ログ件数は設定により変更可）となったときか，アプリケーションから専用の API 等が呼び出されたときか，データベースの接続終了時である．**想定（2）**では 1 回のトランザクションで生成されるログ件数は 10 件以下のため，チェックポイントの間隔はトランザクション 100 回分以上の期間となる．

本研究ではジャーナルファイル固定長化をテーマとしているため，以降，TRUNCATE は対象外とする．

2.3　SQLite が利用するファイル固定長化

設定変更により DB ファイルおよびジャーナルファイルを固定長化することが可能である．ロールバックジャーナルモードの場合は，PERSIST がジャーナルファイルを固定長化するモードに相当する．WAL モードの場合は，設定変更およびデータベース接続直後にチェックポイントを行うことで，ジャーナルファイルの固定長化が可能となる．データベース接続直後のチェックポイントが必要な理由は，WAL のヘッダ部分のチェックポイントの識別子が更新されるのがチェックポイント完了後の 1 回目の更新のときであるためである．

2.4　SQLite の更新処理の対象となるページ

データの投入（Insert 文の処理），データの更新（Update 文の処理），データの削除（Delete 文の処理）によりテーブルやインデックスの B 木のページや，必要に応じて DB ファイルのヘッダのページや空ページのリストのページが更新される．Update 文については，テーブルやインデックスの B 木のキーを変更しない場合は B 木のページが上書きされ，B 木のキーを変更する場合はデータの削除と投入が個別に実行される．

想定（7）では，データ投入対象のページがあふれたり，データ削除対象ページ内の空領域の割合が 2/3 以上となった場合には，一部の例外を除き対象ページと B 木において対象ページの左右のページの計 3 ページにてレコードの再分配処理が実行される．各ページは右隣のページとレコードを 1/2 ずつに分配する．前述の例外は新規のレコードのキーが B 木のキーの最大値である場合であり，その場合は新規のページを確保し，確保したページに新規のレコードを投入して終了する．新規のページは通常空ページから確保され，空ページがない場合は DB ファイル末尾の新規の領域から確保される．

B 木のレコード再分配と DB ファイル末尾の新規領域確保が同時に発生するトランザクションの頻度は比較的低いが，更新するページ数やメタデータの量が多いためトランザクションの応答時間が通常ケースよりも長くなると考え

られる．したがって，想定（10）のアプリケーションではB木のレコード再分配とDBファイル末尾の新規領域確保が同時に発生するトランザクションの応答時間を短縮することが重要である．

3. 机上検討を基にした見積り式

3.1 トランザクション応答時間の見積り式

2.2節の内容を基に，ファイル固定長化有無によるトランザクションの応答時間を比較した結果を**表 1** に示す．表1のR/Vがデフォルトの設定に相当する．表1の$2j_h$はPERSISTにて2回発生するジャーナルファイルのヘッダの更新処理時間に相当する．R/Vの見積り式にj_hを含めなかった理由は，DELETEでは図1のstep4のfdatasyncシステムコールによりログとヘッダの両方をシーケンシャルにストレージに反映するためである．2.1節で言及したようにジャーナルファイルの更新はシーケンシャルな書込みになるため表1のj，m_jは更新するページ数（1〜10）に依存して変化すると考えられる．表1のd，m_dは更新するページ数（1〜10）やそのアクセスパタンに依存して変化すると考えられる．

ロールバックジャーナルモードの場合，ジャーナルファイル固定長化による短縮効果は$m_j + j_c - 2j_h$である．$2j_h$に対応する処理では，ジャーナルファイルのヘッダに対応するデータブロック1件を2回更新する．一方，$m_j + j_c$に対応する処理で更新されるext4のブロックは少なくとも6件（空ブロック数を管理するスーパーブロックとブロックグループディスクリプタのブロック，ディレクトリに対応するデータブロック，空ブロックを管理するブロックビットマップのブロック，inodeを管理するinodeビットマップブロックとinodeブロック）あるため[10]，$m_j + j_c - 2j_h$は正値であると考えられる．また，DBファイル固定長化によりm_dだけ更新処理時間の短縮が見込める．

表1から分かるように，すべての設定の中で，WALモードでファイルを固定長化する設定の応答時間が最も短いと考えられる．WALモードの応答時間はデフォルトの設定より$m_d + d + j_c$（ファイルを固定長化した場合は$m_d + d + m_j + j_c$）だけ短いと考えられる．

表 1　トランザクション応答時間の見積り式
Table 1　Transaction response time in desk study.

設定	応答時間の見積り式 [6]
R[1]/V[3]	$j_c + j_l +$ $m_j + d + m_d$
R[1]/F[4]	$j_l + 2j_h +$ d
W[2),5]/V[3]	$j_l +$ m_j
W[2),5]/F[4]	j_l

[1] ロールバック（Rollback）ジャーナルモードの意味．
[2] WALモードの意味．
[3] ファイル固定長化なし（Variable）の意味．
[4] ファイル固定長化あり（Fixed）の意味．
[5] WALモードについては，データベース接続後から2回目〜最初のチェックポイント直前のトランザクションを想定．
[6] j_c：ジャーナルファイルの作成（create）および削除およびそれらにともなうディレクトリの更新の時間．j_l：ジャーナルファイルのログ部分（log）を更新する時間．j_h：ジャーナルファイルのヘッダ部分（header）を更新する時間．ファイルシステムの1ブロック分をファイル内の既存の領域に書く時間に相当する．m_j：ジャーナルファイルのメタデータを更新する時間．d：DBファイルのデータ部分を更新する時間．m_d：DBファイルのメタデータを更新する時間．

3.2 チェックポイント応答時間の見積り式

2.2節で示したようにWALモードの場合はチェックポイントが実行されるが，チェックポイントの応答時間は，DBファイル固定長化なしの場合$m_d + d$であり，固定長化ありの場合dとなると予想される．m_dは新規の領域に書きこむページ数により変化し，dはチェックポイントで書き込むページ数やそのアクセスパタンに依存して変化すると考えられる．

3.3 ファイル固定長化で定義するデータサイズ

ファイル固定長化に際してはあらかじめデータサイズを決定しておく必要があり，本節ではそれを見積った．必要とするページ数はファイルシステムによらないため，本節の結果はext4以外のファイルシステムにおいても有効である．

3.3.1 ジャーナルファイル

ロールバックジャーナルモードの場合，ジャーナルファイルの固定長化で定義するデータサイズは，1回のトランザクションで更新するページ数分のデータサイズとなる．想定（2）では数十kB程度となり少量である．WALモードの場合，ジャーナルファイル固定長化で定義するデータサイズは，WALモードのログ1,000件分であり，2.1節で示したことからログ1件のサイズはおおむね4kBであるため，4MB程度となる．実際のアプリケーションではDBファイルのデータサイズは多くの場合数百MBになるため[5]，4MBは無視できる程小さいといえる．

3.3.2 DBファイル

空ページは再利用されるため，データベース上に作成するテーブルやインデックスのページ領域の最大データサイズの合計値を求めればよい．テーブルやインデックスの最大データサイズは**表 2** のようになる．要件（1）の場合に最大データサイズがLの3倍になる理由は，この場合にはデータの削除や更新によりテーブル/インデックスの各ページの中に空領域が発生する可能性があり，その割合は最大で2/3となるためである．要件（2）および（3）の場合はデータの削除や更新で発生する空領域は無視できる程小さい．要件（2）の場合に最大データサイズがLの2倍になる理由は，データの投入によりテーブル/ インデックスの各ページが右隣のページとレコードを1/2ずつ分け合い，最大1/2の割合の空領域が生成される可能性があるためであ

表 2　テーブル/インデックスの最大データサイズの見積り式
Table 2　Maximum data size of a table or an index.

No	要件		最大データサイズ（B）
(1)	RD[2] または CU[3]		$3L$[1]
(2)	RD[2] でも CU[3] でもない	RI[4]	$2L$[1]
(3)		RI[4] でない	$(1+\frac{1}{m})L$[1]

[1] L は，テーブル/インデックスの全レコードサイズの最大値（B）．**想定**（**8**），（**9**）から平均レコードサイズと最大レコード数が把握可能であり，それらを掛けることで求めることが可能．m は各ページに格納される最小レコード数．
[2] RD（Randam Delete）は，テーブル/インデックスの B 木のキーに関して昇順ではないデータの削除を行う場合．
[3] CU（Changing b-tree layout Update）は，データの更新によりテーブル/インデックスの B 木のキーまたはレコードのサイズを変更する場合．
[4] RI（Randam Insert）は，テーブル/インデックスの B 木のキーに関して昇順ではないデータの投入を行う場合．

る．要件 (3) の場合には，2.4 節で説明したようにデータの投入で新たな空領域が発生することはない．新しく確保されたページの内割合が $(1-\frac{1}{m+1})$ の領域はレコードで埋まるため，最大データサイズは $(1/(1-\frac{1}{m+1}))L=(1+\frac{1}{m})L$ となる．m はテーブルの場合 1 でインデックスの場合 4 であるため，この値はテーブルの場合最大 $2L$ でインデックスの場合最大 $(5/4)L$ となる．この値は，最大レコードサイズが小さい場合は m が大きくなるためおおむね L となる．

4. 更新処理時間の評価

4.1 評価環境と評価条件

評価環境を**表 3** に示す．SQLite のソースコードには 2 点の変更を加えた．1 点目は，文献 [9] に基づいた，ext4 上で利用する場合に不要な処理の無効化（SQLITE_IOCAP_SAFE_APPEND フラグ，SQLITE_OS_UNIX マクロ，SQLITE_DISABLE_DIRSYNC マクロの有効化）である．2 点目は，図 1 の個別処理時間測定用処理（clock_gettime システムコール）の追加である．

共通的な評価条件を**表 4** に示す．8 分位数以上の値を除いたのは，microSD カードの書込み特性により発生する外れ値を取り除くためである．データ投入とは別のタイミングで保管期限を過ぎたデータが主キーに関して昇順に削除されることを想定した．後述するように評価では使用したデータのサイズは実際のアプリケーションで想定されるデータサイズ（3.3.1 項参照のこと）よりも小さいが，更新処理の内容は変わらないため，本評価の条件で 3.1 節および 3.2 節で作成した見積り式の妥当性を検証可能と判断した．

使用したテーブルの構成を**表 5** に，データのパタン一覧を**表 6** 示す．データは**想定**（**2**），（**3**），（**4**）に従い作成した．テーブル t は表 2 の要件 (3) のケース，インデックス i は表 2 の要件 (1) のケースに該当する．

表 3　評価環境
Table 3　Evaluation platform.

項目	内容
組込み機器	Raspberry Pi 3
CPU	ARM Coretex-A53（1,200 MHz，4 コア）
メモリ	1 GB
ストレージ	16 GB，microSD（Kingston，SDHC class10）
OS	Raspbian 9.1（stretch，カーネルは 4.9.59-v7）

表 4　評価条件（共通事項）
Table 4　Evaluation conditions (Common).

項目	内容
SQLite のページサイズ	4 KiB
ext4 のブロックサイズ	4 KiB
処理時間算出方法	100 回測定した値から 8 分位数以上を除いた平均値
DB ファイルの更新処理時間測定方法	図 1 のステップ 5 の処理時間
ジャーナルファイルの更新処理時間測定方法	全体の処理時間と DB ファイルの更新処理時間の差分
DB ファイルの固定長化サイズ	1,956 KiB（489 ページ分）
想定最大レコード件数	15,000

表 5　評価に使用したテーブル t の構成
Table 5　Table characteristics.

列名	データ型	データ概要	備考
c_1	INTEGER	0 から始まる連番	主キー
c_2	TEXT	固定値が連続するパタンの繰返し	二次インデックス i を設定
c_3	TEXT	固定値	

表 6　評価に使用したデータのパタン一覧
Table 6　Data characteristics.

パタン名	c_2 列の内容	c_3 列の内容
データ 1	10 種類，長さ 3，連続回数 50	長さ 66
データ 2	100 種類，長さ 3，連続回数 1	長さ 66

トランザクションの応答時間の評価条件を**表 7** に示す．初期処理は，データベースのスキーマを作成してレコード数 0 の状態から評価対象の処理直前までに実行した処理を意味する．3.1 節の見積り式の妥当性を検証するため，B 木のレコード再分配と DB ファイル末尾の新規ページ確保が同時に発生するケースを選定した．更新されるページはレコード再分配の対象となるインデックスの B 木の子ページおよび親ページ 4 件，新規で確保されるインデックスのページ 1 件，テーブルのページ 1 件，DB ファイルのヘッダである．テーブルの B 木とインデックスの B 木の高さは 2 以下であるため，B 木の内部ノードについてレコードが再分配されることはない．データの削除については，データの投入との相違点は DB ファイルの更新時にメタデータの

表 7　評価条件（トランザクションの応答時間）
Table 7　Evaluation conditions (transaction response time).

項目	内容
使用データ	データ 1
初期処理	1,035 件のデータを投入
初期処理後のページ数	231（ヘッダ 1 件，テーブルのページ 198 件，インデックスのページ 32 件）
トランザクションの内容	1 件データを投入（DB ファイル末尾に新規ページが 1 件追加される）
トランザクションの更新ページ数	7（想定（2）に従い設定）

表 8　評価条件（チェックポインの応答時間）
Table 8　Evaluation conditions (checkpoint response time).

項目	内容
使用データ	データ 2
初期処理	10,000 件のデータを投入しデータベース接続を終了．その後 10 件のデータを投入するトランザクションを 177 回実行
初期処理後のページ数	231（ヘッダ 1 件，テーブルのページ 198 件，インデックスのページ 32 件）
初期処理後のログ数	1,005
チェックポイントのダーティページ数	71
チェックポイントで増加するページ数	39

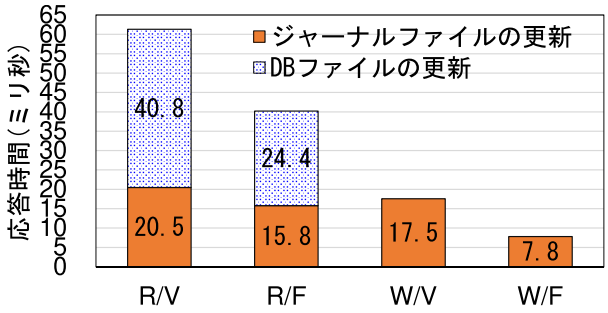

図 2　トランザクションの応答時間（略記は表 1 のものと対応）
Fig. 2　Transaction response time.

更新が発生しない点のみであるため，評価対象外とした．

チェックポイントの応答時間の評価条件を**表 8** に示す．

4.2　評価結果

トランザクションの応答時間を**図 2** に，WAL のチェックポイントの応答時間を**図 3** に示す．以下，個別に説明する．ロールバックジャーナルモードにてファイル固定長化を行った場合のトランザクションの応答時間（図 2 中の R/F）は 40.2 ミリ秒であり，デフォルト設定（図 2 中の R/V）の応答時間 61.3 ミリ秒の 65.6%となった．

WAL モードにおいてファイル固定長化ありの場合，トランザクションの応答時間（図 2 中の W/V）は 7.8 ミリ秒であり，ファイル固定長化なしの場合の応答時間（図 2 中の W/F）17.5 ミリ秒と比較すると 44.6%となった．

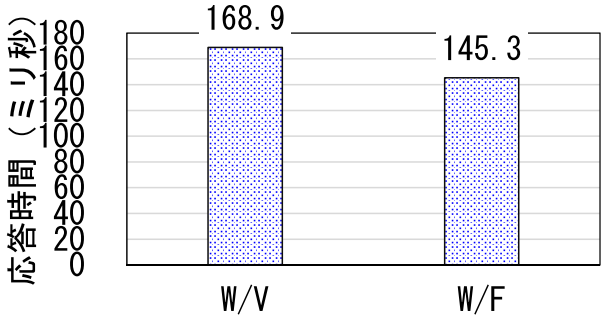

図 3　WAL のチェックポイントの応答時間
Fig. 3　Checkpoint response time.

表 9　見積り式の各要素の推定方法と推定値（トランザクション）
Table 9　Estimation of each component.

項目	推定値（msec）	推定方法
j_c	0.6	R/V[1] に関する次の A) と B) の差分 A) D_r[2] に対する図 1 step1，4，6 の時間 B) D_r[2]（事前作成）に対する図 1 step2，4（ディレクトリ更新除く）の時間
j_l	9.4	以下 2 つの平均時間 A) R/F[1] の D_r[2] に対する図 1 step2 B) W/F[1] の D_w[3] に対する図 1 step2
j_h	3.4	R/F[1] の D_r[2] に対する図 1 step3，4 の時間
m_j	8.3	以下 2 つの平均時間と j_l（推定値）の差分 A) R/V[1] の D_r[2]（事前作成）に対する図 1 step2，4（ディレクトリ更新除く） B) W/V[1] の D_w[3]（事前作成）に対する図 1 step2，4（ディレクトリ更新除く）
d	24.6	R/F[1] の D_d[4] に対する図 1 step5 の時間
m_d	18.5	R/V[1] の D_d[4] に対する図 1 step5 の時間と d（推定値）の差分

[1] 表 1 のものと対応．
[2] ロールバックジャーナルモードにおけるジャーナルファイルのダミーファイルの意味．
[3] WAL モードにおけるジャーナルファイルのダミーファイルの意味．
[4] DB ファイルのダミーファイルの意味．

評価対象のトランザクションについて，図 1 のジャーナリングの動作を模擬したダミーのファイルとプログラムを使用して表 1 の見積り式の各要素を推定した結果を**表 9** に示す．表 9 の推定値を基に表 1 の見積り式により見積り値を求めた結果（**表 10**）は同程度（誤差は最大で 1.6 ミリ秒）になることが確認でき，3.1 節で示した見積り式が妥当であると結論した．

チェックポイントに関しては，DB ファイル固定長化により応答時間が 168.9 ミリ秒から 145.3 ミリ秒となり固定長化を行わない場合の 86.0%となった．3.2 節で示した見積り式の各要素を推定した結果から作成した見積り値を**表 11** に示す．表 11 について，見積り式の各要素の見積り方法は，表 9 の d および m_d の推定方法と同様であり，

表 10　見積り値と測定結果（トランザクション）
Table 10　Estimated time and measurement time (transaction).

項目	見積り式	見積り値（ミリ秒）	測定値
R/V	$j_c + j_l + m_j + d + m_d$	61.5	61.3
R/F	$j_l + 2j_h + d$	40.8	40.2
W/V	$j_l + m_j$	17.7	17.5
W/F	j_l	9.4	7.8

表 11　見積り値と測定結果（チェックポイント）
Table 11　Estimated time and measurement time (checkpoint).

項目	見積り式	見積り値（ミリ秒）	測定値
W/V	$d + m_d$	163.7	168.9
W/F	d	141.3	145.3

図 1 step4 を模擬したダミーのファイルおよびプログラムを用いた．表 11 から分かるように見積り値は測定値と同程度（誤差は最大で 5.2 ミリ秒）であり，3.2 節で示した見積り式が妥当であると結論した．

5. おわりに

Linux の ext4 を利用する組込みシステムにて逐次発生するデータを SQLite に蓄積していくようなアプリケーションを対象として，ファイル固定長化による更新処理時間の短縮効果とファイル固定長化で定義するデータサイズの 2 つについて机上検討により見積り式を作成した．この 2 つのうち，更新処理時間の短縮効果については，実機評価により妥当性を確認した．今後の課題は，ストレージ種別ごとの特性も考慮した見積り式の作成，見積り式の各要素を測定する方法の確立である．

参考文献

[1] About SQLite, available from ⟨https://www.sqlite.org/about.html⟩ (accessed 2018-05-11).
[2] Well-Known Users of SQLite, available from ⟨https://www.sqlite.org/famous.html⟩ (accessed 2018-05-11).
[3] Park, J.-H., Oh, G., Lee, S.-W: SQL Statement Logging for Making SQLite Truly Lite, *PVLDB*, Vol.11, No.4, pp.513–525 (2017).
[4] Oh, G., Kim, S., Lee, S.-W, Moon, B.: SQLite Optimization with Phase Change Memory for Mobile Applications, *PVLDB*, Vol.8, No.12, pp.1454–1465 (2015).
[5] Tuan, D.Q., Cheon, S., Won, Y.: On the IO Characteristics of the SQLite Transactions, *Proc. International Conference on Mobile Software Engineering and Systems* (*MOBILESoft '16*), pp.214–224 (2016).
[6] Jeong, S., Lee, K., Lee, S., Son, S., Won, Y.: I/O Stack Optimization for Smartphones, *Proc. USENIX ATC 2013* (2013).
[7] Lee, W., Lee, K., Son, H., Kim, W.-H., Nam, B., Won, Y.: Waldio: Eliminating the filesystem journaling in resolving the journaling of journal anomaly, *Proc. USENIX ATC 2015 Annual Technical Conference* (2015).
[8] 金松基孝，山地　圭：組込み機器のデータ管理に適した軽量データベース TinyBrace，東芝レビュー，Vol.67, no.8 (2012).
[9] Pillai, T.S., Chidambaram, V., Hwang, J., A.-Dusseau, A.C., A.-Dusseau, R.H.: Towards Efficient, Portable Application-Level Consistency, *Proc. 9th Workshop on Hot Topics in Dependable Systems* (*HotDep'13*) (2013).
[10] Daniel, P.B., Marco, C.（著），高橋浩和（監訳），杉田由美子，清水正明，高杉昌督，平松雅巳，安井隆宏（訳）：詳解 Linux カーネル 第 3 版，O'Reilly Japan, Inc. (2007).

藤井 雄規　（正会員）

1987 年生．2010 年信州大学理学部数理・自然情報科学科卒業．2012 年東京大学大学院数理科学研究科修士課程修了．同年三菱電機入社．データベースの研究に従事．

水口 武尚

1969 年生．1991 年東京工業大学工学部情報工学科卒業．1993 年同大学大学院修士課程修了．同年三菱電機入社．組込み機器向けソフトウェアの研究に従事．電子情報通信学会会員．

樋口 毅　（正会員）

1990 年東北大学工学部卒業．同年三菱電機株式会社入社．情報処理技術の研究・開発に携わり現在に至る．機器の予防保全や最適化問題等の研究開発に従事．

（担当編集委員　山口 実靖）