High Performance SQLite の開発
—もっと速い SQLite—

1. 背景

SQLite は便利なソフトウェアであり，広くユーザも獲得しているが，動作速度についてはその他の RDBMS に劣っているとの見方が一般的である．しかし，手軽に利用できる RDBMS だからといって動作が遅くても良いということにはならず，SQLite が高速化することは次のような点で重要だと考える．

近年ビッグデータという言葉を聞く機会が増えてきている．情報技術の爆発的発展により，人間が計算機上で扱うデータも飛躍的に増大しているのだ．この潮流の中，サーバサイドの RDBMS では日々高速化の努力が続けられている．クライアントサイド，つまりスタンドアロン RDBMS の動作する場面においても，その扱うデータ量は増加していくとクリエータは考える．SQLite を使用するアプリケーション開発者は，二次記憶装置の容量が年々大きくなっていくに従い，多くのデータを SQLite に入れて永続化し，便利に操作しようと考える．例えばスマートフォン上で動作する某メッセンジャーアプリケーションは，メッセージ本文を含めあらゆる情報を SQLite で管理しているという．また，SQLite で巨大なデータを扱うことを望むのはアプリ開発者だけではない．PC 上で便利にデータ管理をしたいと望む人々も存在し，クリエータ自身も，研究中に得られた大量の実験データを全て SQLite のデータベースで管理を行っている．

このように SQLite で扱うデータは巨大化しているが，SQLite の処理能力がデータの肥大化に追いつけなくなる可能性がある．データインテンシブ処理の大家である Jim Gray は，記憶装置の容量の増加スピードは帯域の増加スピードより 1 桁速いと指摘した．だとすると，記憶容量の増加に合わせて肥大化するデータに対処するのは，ハードウェア側の努力だけでは難しく，ソフトウェア側の高速化が望まれていると考えられる．

以上のように，これから SQLite の扱うデータが益々増大することを考えると，高速化の必要性は大きいと考えられる．

2. 目的

本プロジェクトでは，SQLite をベースとした，次のような RDBMS の開発を目的とする．

・SQLite の「手軽さ」を引き継ぐ．
・元の SQLite より十分高速である．
・SQLite を使用しているアプリケーションも High Performance SQLite による高速化の恩恵を受けられる．

3. 開発の内容

本プロジェクトでは，SQLite を高速化するための手法の実装と，それを補助するツールの開発を行った．本章では，初めに高速化を目指した開発について，次に開発補助ツールの作成について記す．

3. 1. 高速化を目指した開発

3. 1. 1. 適応的プリアロケーション

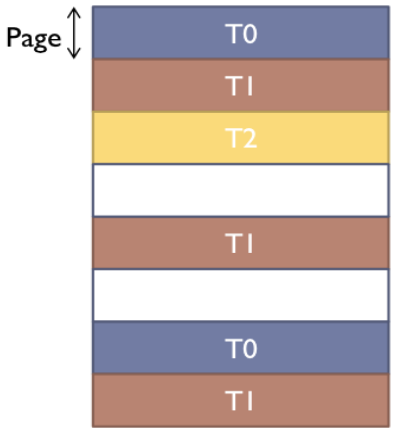

図 1 – フラグメンテーションのあるデータベース

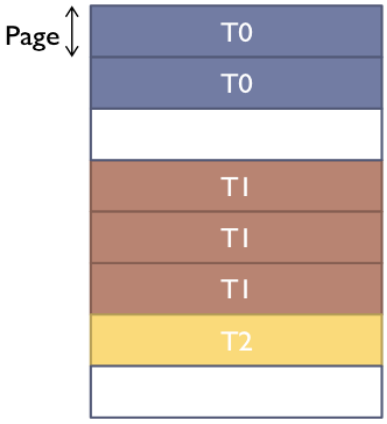

図 2 – フラグメンテーションのないデータベース

ディスク I/O を削減するための手法として，フラグメンテーションを抑えるプリアロケーションを実装した．SQLite におけるフラグメンテーションは，同種のテーブルまたはインデックスが不連続領域に配置された状態（図 1）であると定義し，同種のテーブルやインデックスが連続配置された状態（図 2）になるようにプリアロケーションを実装した．

プリアロケーションのサイズ（同種のテーブル・インデックスを何ページ連続させるか）を変化させ，フルスキャンクエリの実行速度を計測すると，図 3 の結果が得られた．これより，**プリアロケーションサイズが 32 のときは，元々の SQLite より 5.0 倍高速**であることが示された．ただし，プリアロケーションサイズが 1 のときは元々の SQLite を使用して計測をしている．

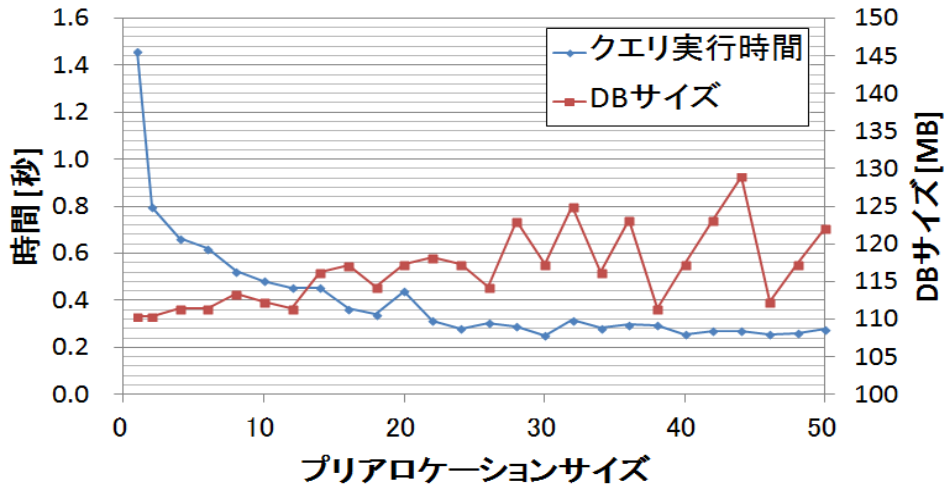

図 3 – プリアロケーションによる高速化効果と DB サイズの増加

しかし，プリアロケーションのサイズを大きくしていくにつれ，DB サイズも肥大化してしまう．これは，プリアロケーションが予約領域を予め余分に取る手法であるためである．DB サイズを抑えつつプリアロケーションの高速化効果を得ることが重要と考え，プリアロケーションサイズを

自動的に調整する**適応的プリアロケーション**を開発した．適応的プリアロケーションにおけるページ確保戦略は次のようなものである．
・大きなテーブル・インデックスにはプリアロケーションサイズを大きく調整し，速度向上を図る
・小さなテーブル・インデックスにはプリアロケーションサイズを小さく調整し，DB サイズを抑える
これをシステムが自動で行うことにより，ユーザは手間なく高速化の恩恵を DB サイズを小さく保ったまま得ることができる．

適応的プリアロケーションを図 3 と同様の評価に適用すると，単純なプリアロケーションとほぼ同様の高速化効果を，より小さな DB サイズで得られていることが確認できた．

更に適応的プリアロケーションの効果を，実アプリケーションに近いベンチマークで計測した．Evolution という Ubuntu Linux 標準のメールクライアントが実際に使用する DB，SQL を使用したベンチマーク（図 4）で，**適応的プリアロケーションは全ての SQL において元々の SQLite を上回る性能を示した**．最も効果が大きい SQL#1 では，**2.82 倍の高速化**となった．

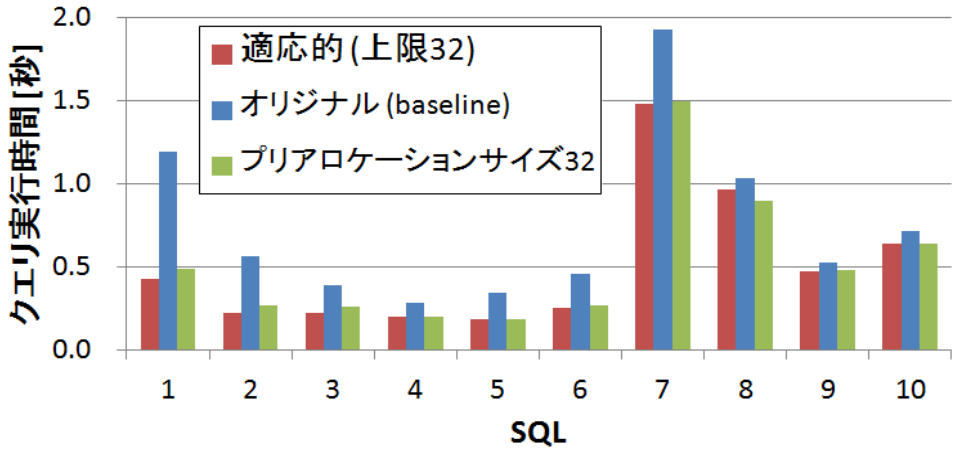

図 4 – Evolution の実 DB, SQL を用いたベンチマーク

3. 1. 2. Hash-Based GROUP BY

SQLite の GROUP BY クエリは，内部的にソートアルゴリズムを使用したもので低速であった．これをハッシュテーブルを使用するものに置き換え，簡単な評価を行った結果を図 5 に示す．

この結果，**Hash-Based GROUP By は，元々の SQLite が備えているソート，インデックスに基づく集約よりも十分に高速**であることが示された．ただし，この最適化はメモリをはみ出すサイズのデータに対応しておらず，ハッシュテーブルをヒープからディスクに退避させる工夫を加える必要がある．

本拡張の過程で SQLite にハッシュテーブルが実装されたことで，Hash-Join など別の高速なアルゴリズムを適用できる可能性が広がった．

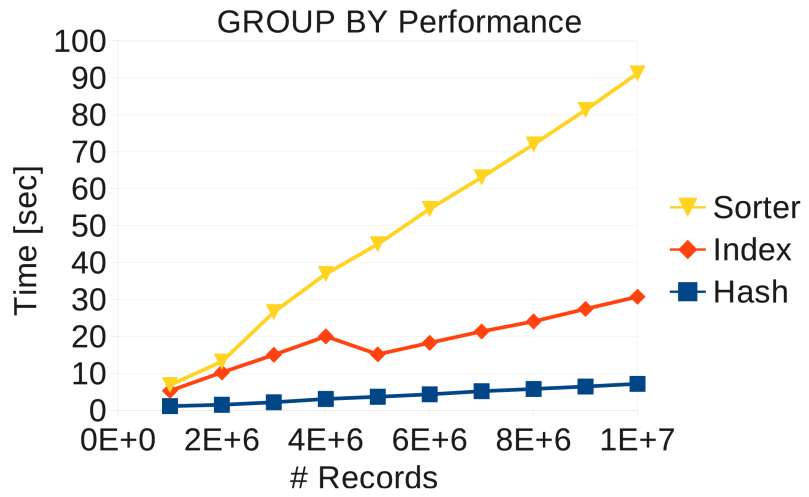

図 5 – Hash-Based GROUP BY の評価

3. 2. 開発補助ツール

3. 2. 1. SQLiteDbVisualizer

特に適応的プリアロケーションの開発過程において，**SQLite の DB を可視化**する必要に駆られた．しかしそのようなソフトは見受けられなかったため，独自に開発し Github に公開をした．詳細は関連 URL を参照されたい．

3. 2. 2. sqlite3_benchmark

性能最適化においては，小さい変更を加える度にベンチマークを繰り返しとることが重要である．その手間を軽減するため，**柔軟に SQLite のベンチマークを作成でき，コマンド 1 発で比較グラフまでを生成するツール**を開発した．こちらも関連 URL を参照されたい．

4. 従来の技術（または機能）との相違

High Performance SQLite は，元々の SQLite より十分に高速であることが実験的に示された．

その手法の一つである適応的プリアロケーションは，クリエータの知る限り新規のものである．単純なプリアロケーションのような手法は，Oracle Database のエクステントなどに類似するが，プリアロケーションのサイズを高速化・DB サイズの両面から自動的に調整する技術とその有効性の実証は新規のものと考える．

Hash-Based GROUP BY は SQLite 以外の RDBMS には従来も取り入れられていたが，SQLite にはクエリ処理でハッシュテーブルを用いる箇所は存在しなかった．これは，ワーキングメモリを小さく抑えようとする努力であると予想されるが，メインメモリのサイズが増加の一途を辿る今，重要になる機能だと考える．

5. 期待される効果

High Performance SQLite によって，SQLite バックエンドとするアプリの性能が向上すること

が考えられる．近年のスマートフォン普及により，SQLite を使用するアプリは益々増加している．アプリ開発者にとって基盤的なソフトである SQLite を高速化する意義は大きい．

更に，個人レベルでのデータ管理にも高速化効果が波及することが考えられる．クリエータは個人の研究や本プロジェクトの実験データを SQLite で管理しているが，データサイズが増加したときのデータ操作の遅さにストレスを感じることがあった．しかし，High Performance SQLite の導入により，巨大なデータを SQLite で管理する抵抗が解消された．これはあくまでクリエータの経験であるが，データ操作をより直感的にサポートする SQLiteManager のような GUI アプリも充実してきた今，このような要望は多くあると考えられる．

6．普及（または活用）の見通し

High Performance SQLite の目指すべきゴールは，SQLite のメインラインに取り入れられることだと考えている．しかし，SQLite の開発者メーリングリストから得られる情報によると，SQLite の開発コミュニティは閉鎖的であり，クリエータの改変が簡単に取り入れられるとも考えづらい．

閉鎖的であるのは，SQLite が広く使用されているので，毎回のリリースを安定したものにする必要があるからということである．

これを踏まえると，テストやレビューのしづらい巨大なパッチをいきなり提出するのは愚策であり，本プロジェクトで行った改変を小分けにしてパッチ化する必要があると考えられる．具体的には，例えば B-Tree/Pager レイヤの改変であるプリアロケーションについては，まず簡単に B-Tree/Pager レイヤを切り替えられる，MySQL で言うところのストレージエンジンのような機構の提案から始める，という手は考えている．

また，メインラインと並行し，SQLite の ″Contributed Files″ に掲載してもらうことも視野に入れたい．このページには，テストこそ十分にされていないが，SQLite の一部ユーザにとって有用である拡張機能が掲載されている．SQLite の Web サイトに掲載されるという意味で，ユーザを獲得する大きな一歩であると考える．

7．クリエータ名（所属）

中谷 翔（東京大学大学院情報理工学系研究科）

（参考）関連 URL

SQLiteDbVisualizer - https://github.com/laysakura/SQLiteDbVisualizer
sqlite3_benchmark - https://github.com/laysakura/sqlite3_benchmark