MySQL 5.6における大量データロード時の考慮点

第18回 AWS User Group - Japan 東京勉強会

2013/10/04 平塚 貞夫

2013/10/07 Revision 2

自己紹介

- DBエンジニアやってます。専門はOracleとMySQL。
  - システムインテグレータで主にRDBMSのトラブル対応をしています。
  - 仕事の割合はOracle:MySQL:PostgreSQL＝5:4:1ぐらいです。

- Twitter:@sh2nd
- はてな:id:sh2
- 

- 写真は実家で飼っているミニチュアダックスのオス、アトムです。

本日のお題

InnoDBデータベース 大量データ投入

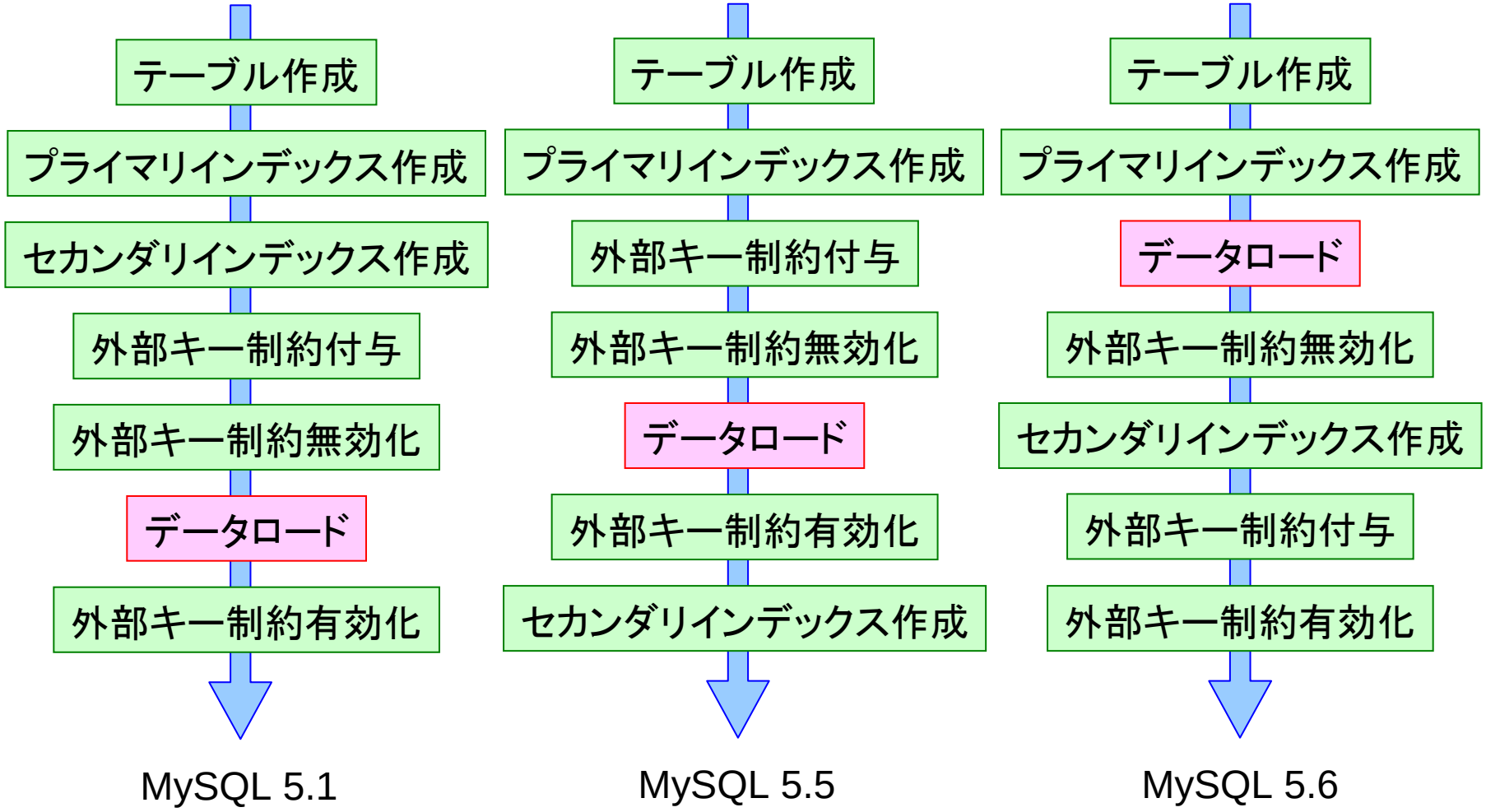
- 日本HPさんが以下の検証結果を公開されています。すばらしい資料です。
  http://h50146.www5.hp.com/products/software/oe/linux/mainstream/support/doc/read/

- しかし、2008年の資料なので内容が古くなってきました。
- 本日は2013年の状況をご紹介します。
- 青字はRevision 2で追記、修正したところです。

最適なデータロード手順

最適なデータロード手順

- MySQLのバージョンによって、最適なデータロード手順が異なります。



Fast Index Creation

- MySQL 5.5の新機能です。

- 従来、MySQL(InnoDB)のセカンダリインデックス作成はテーブルコピーを伴っていました。これはセカンダリインデックスが定義された新しいテーブルを作成し、古いテーブルから新しいテーブルに全データをコピーして、古いテーブルと新しいテーブルの名前を入れ替えるという仕組みです。非常に遅いです。

- MySQL 5.5から、テーブルコピーを伴わずにセカンダリインデックスのみの作成、削除が行えるようになりました。

- Fast Index Creationと名付けられていますが、実際のところ今までSlowだったものが修正されてNormalになったという話です。他のRDBMSに比べて特別に速くなったということではありません。

Online DDL

- MySQL 5.6の新機能です。
- テーブルへのカラム追加、インデックス作成などが更新SQLと同時に行えるようになりました。従来は参照SQLのみ同時に実行可能でした。
- 制約条件が多いため、実施前にマニュアルで内容を確認してください。
  http://dev.mysql.com/doc/refman/5.6/en/innodb-create-index-overview.html
- 外部キー制約の付与についてもオンライン化されています。またパラメータforeign_key_checksをOFFに設定することでテーブルコピーが抑制され、高速化が図れるようになっています。

```
SET SESSION foreign_key_checks = OFF;

ALTER TABLE `order_line`
  ADD CONSTRAINT `order_line_fk1`
    FOREIGN KEY (`ol_w_id`, `ol_d_id`, `ol_o_id`)
    REFERENCES `orders` (`o_w_id`, `o_d_id`, `o_id`),
  ADD CONSTRAINT `order_line_fk2`
    FOREIGN KEY (`ol_supply_w_id`, `ol_i_id`)
    REFERENCES `stock` (`s_w_id`, `s_i_id`);

SET SESSION foreign_key_checks = ON;
```

データロードの性能測定

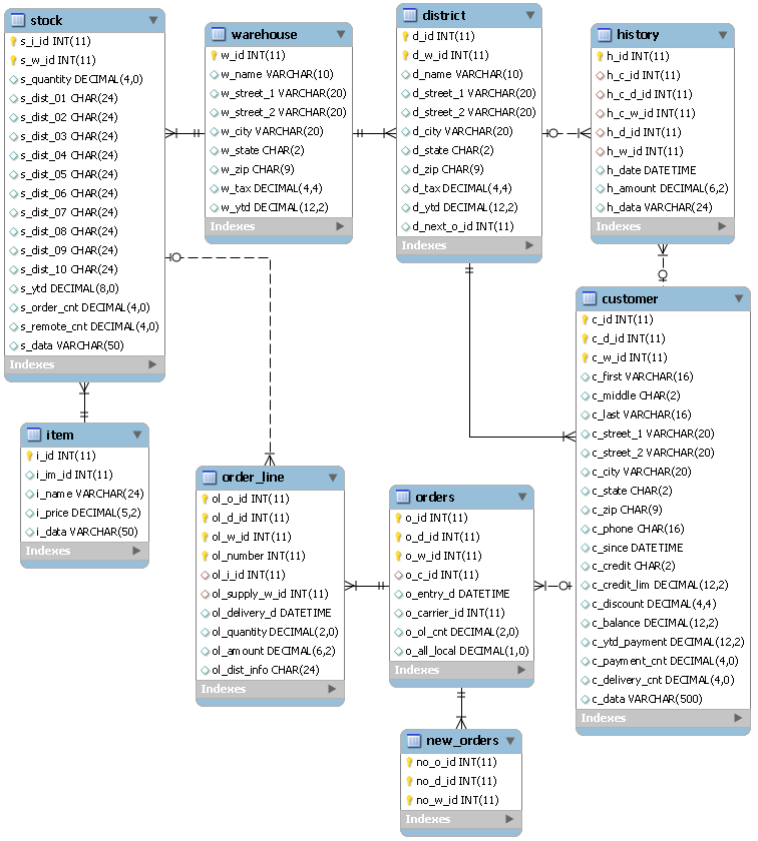
測定内容

- TPC-Cの注文明細テーブルに対してデータロードを行い、処理時間を測定しました。

- レコード数：30,007,060件
- ファイルサイズ：1,958MBytes
- プライマリインデックス順でソート済み
- 外部キー制約に違反するデータはない

- プライマリインデックス1個
- セカンダリインデックス2個
- 外部キー制約2個

```
PRIMARY KEY (`ol_w_id`,`ol_d_id`,`ol_o_id`,`ol_number`),
KEY `order_line_ix1` (`ol_i_id`),
KEY `order_line_ix2` (`ol_dist_info`),
CONSTRAINT `order_line_fk1`
  FOREIGN KEY (`ol_w_id`, `ol_d_id`, `ol_o_id`)
  REFERENCES `orders` (`o_w_id`, `o_d_id`, `o_id`),
CONSTRAINT `order_line_fk2`
  FOREIGN KEY (`ol_supply_w_id`, `ol_i_id`)
  REFERENCES `stock` (`s_w_id`, `s_i_id`)
```



測定環境

- 以下の環境を使用しました。
  - CPU：Intel Core i5-2400S (Quad-Core、2.50GHz、Max 3.30GHz)
  - SSD：CSSD-S6T512NHG5Q (TOSHIBA HG5d)
  - ホストOS：Scientific Linux 6.4 x86_64
  - ゲストOS：Scientific Linux 6.4 x86_64
  - ゲストCPU割り当て：2Cores
  - ゲストメモリ割り当て：8GBytes

- MySQLは以下の3バージョンを使用しました。
  - MySQL 5.1.69 (Scientific Linux 6.4バンドル版)
  - MySQL 5.5.34 (オラクル公式RPM版)
  - MySQL 5.6.14 (オラクル公式RPM版)

- 主なパラメータを以下に示します。
  - innodb_buffer_pool_size = 1G
  - innodb_log_file_size = 128M
  - log_bin = OFF

測定パターン

- 5つのパターンで測定を行いました。

- パターン1：単純ロード
  TBL作成→PK作成→IX作成→FK作成→ロード

- パターン2：＆FKを無効化
  TBL作成→PK作成→IX作成→FK作成→**FK無効化**→ロード→**FK有効化**

MySQL 5.1最速

- パターン3：＆IXを後作成
  TBL作成→PK作成→FK作成→FK無効化→ロード→FK有効化→**IX作成**

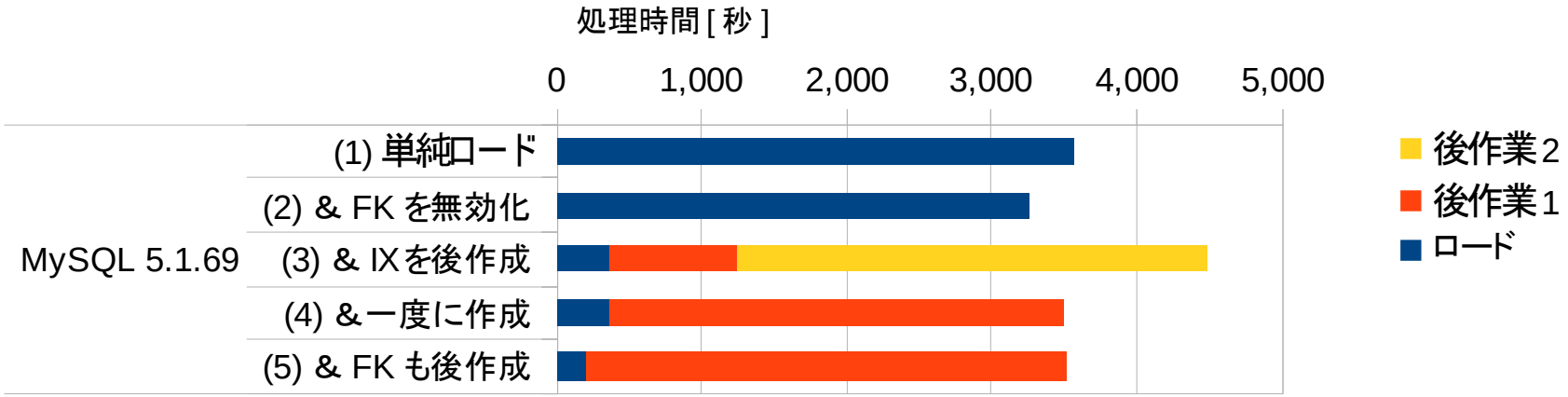
- パターン4：＆一度に作成
  TBL作成→PK作成→FK作成→FK無効化→ロード→FK有効化→**IXを一度に作成**

MySQL 5.5最速

- パターン5：＆FKも後作成
  TBL作成→PK作成→ロード→FK無効化→**IXとFKを一度に作成**→FK有効化

MySQL 5.6最速

MySQL 5.1の測定結果

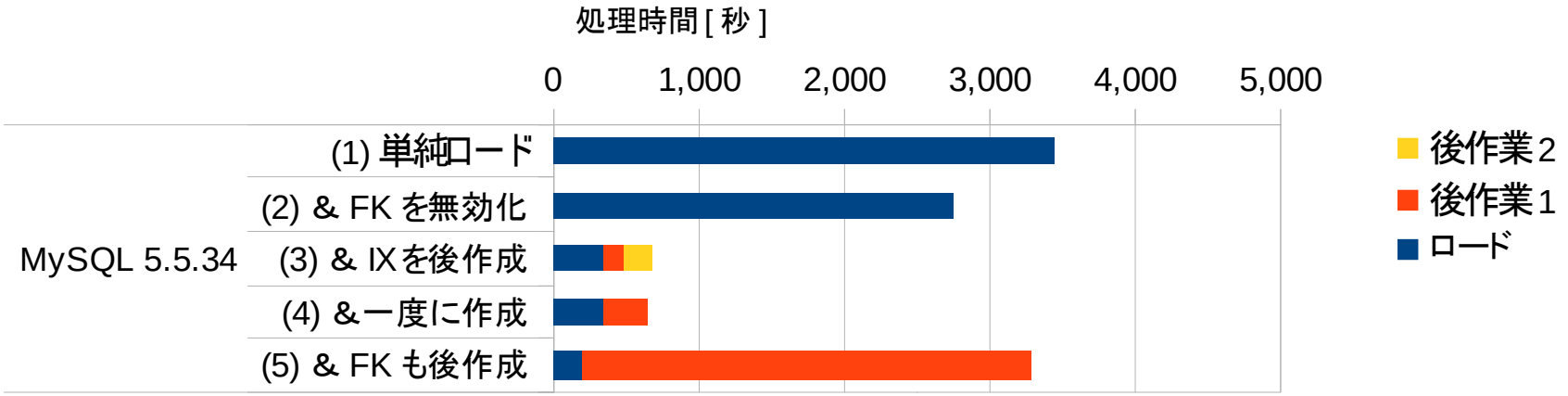


- パターン2が最も速いです。

- パターン3の後作業1は1つ目のセカンダリインデックス作成、後作業2は2つ目のセカンダリインデックス作成です。MySQL 5.1のセカンダリインデックス作成はテーブルコピーを伴うため、実質的にデータロードを3回行ってしまっています。

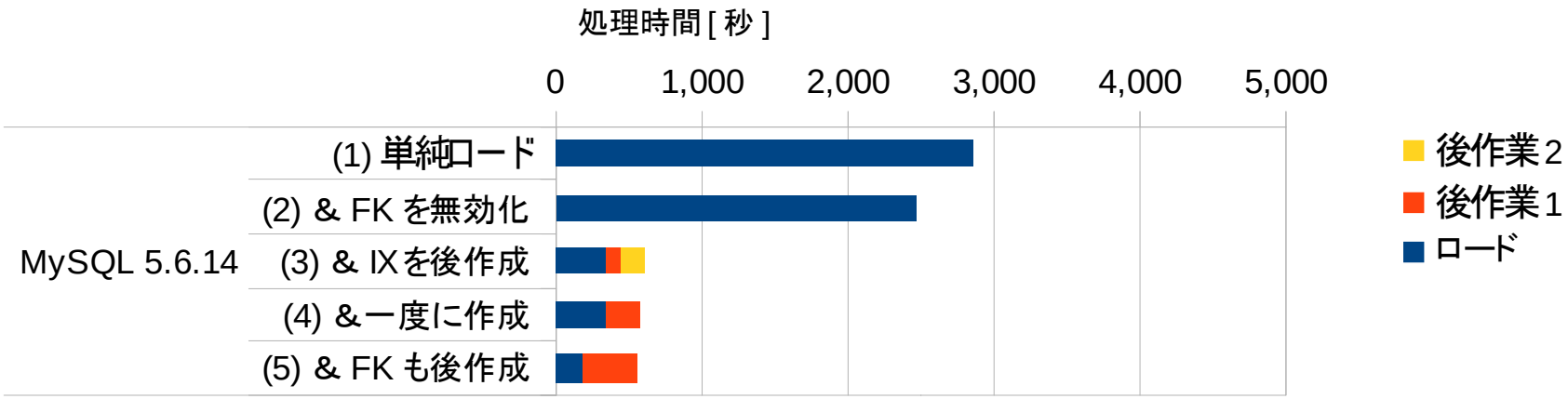
- パターン4のようにセカンダリインデックス作成を一度に行うことで、テーブルコピーの回数を減らすことが可能です。これはパターン3のオレンジ部分がなくなった状態と同じです。

MySQL 5.5の測定結果



- パターン4が最も速いです。MySQL 5.5のFast Index Creationがよく効いています。
- Fast Index Creationが効く場合でも、パターン4のようにセカンダリインデックス作成を一度に行うことは有効です。これは、セカンダリインデックス作成の最初に行うテーブルフルスキャンを1回に抑えることができるためです。
- MySQL 5.5の場合、外部キー制約の付与は残念ながらテーブルコピーを伴います。

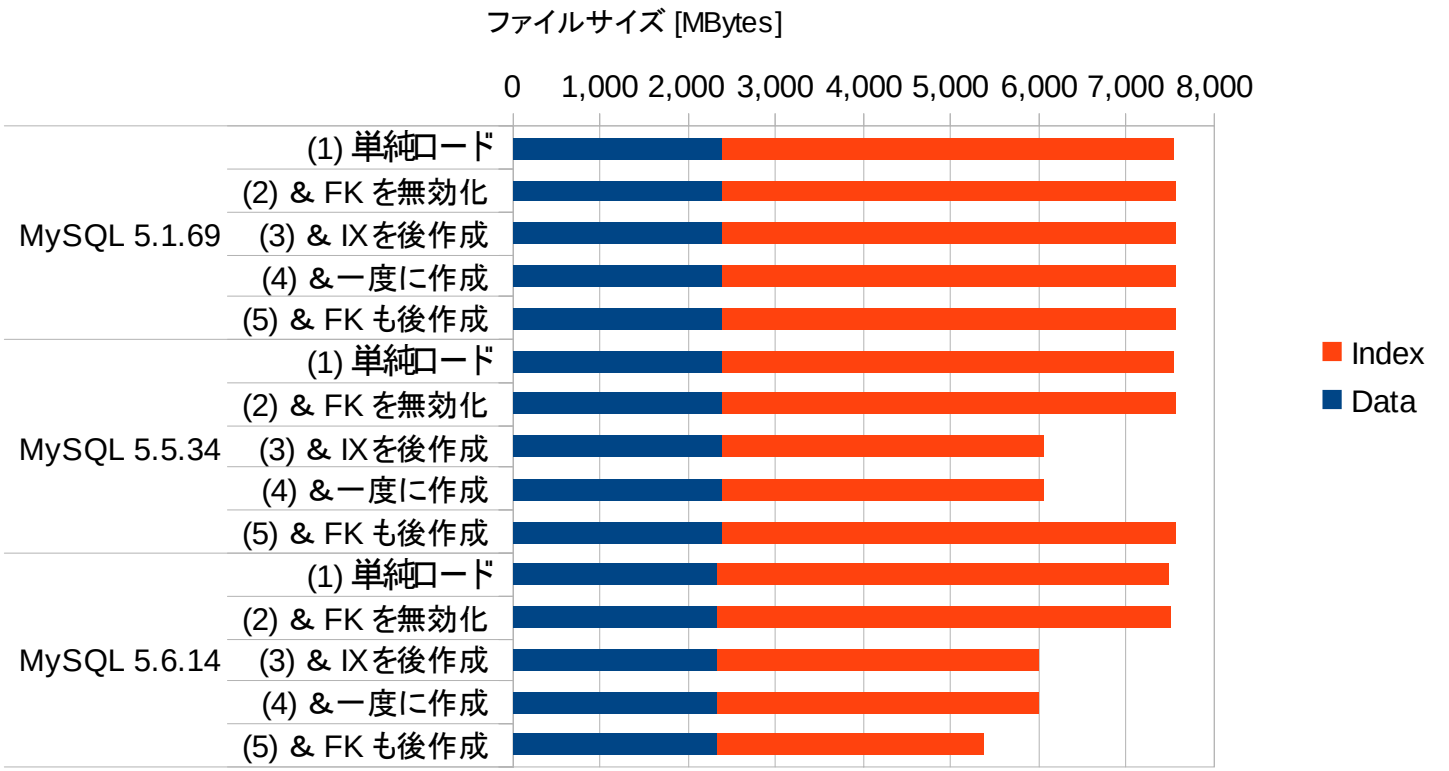
MySQL 5.6の測定結果



- パターン5が最も速いです。外部キー制約の付与についてテーブルコピーを抑制できています。

- 最適なデータロード手順においても1,958MBytesのファイルをロードするのに560秒かかっているので、スループットは毎秒3.5MBytesということになります。バージョンアップごとに改善されてきてはいるのですが、他のRDBMSと比較するとまだまだ遅いです。

ファイルサイズの比較



- プライマリインデックスでソート済みのファイルをロードしているので、テーブル本体はいずれも断片化していない状態です。
- MySQL 5.5ではセカンダリインデックス、MySQL 5.6では加えて外部キー制約を後作成することで、セカンダリインデックスの断片化を抑制することが可能です。

大量データロード時の考慮点

tmpdirに要求される容量について

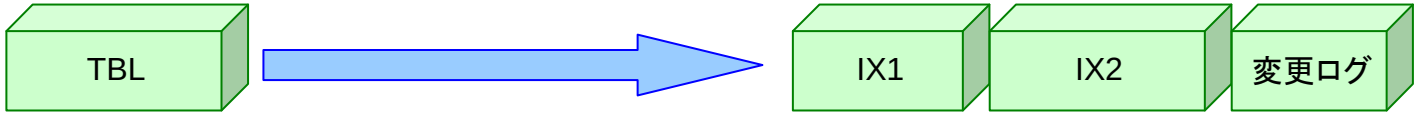
- データロード、セカンダリインデックス作成や外部キー制約付与などの処理において、パラメータtmpdirで指定したディレクトリに作業用の一時ファイルが作成されます。tmpdirに要求される容量について説明します。

- データロードの際は、データロードに伴って出力されるバイナリログサイズ分の容量がtmpdirに要求されます。トランザクション実行中のバイナリログはパラメータbinlog_cache_sizeに収まる間はメモリ上に保持され、それを超えるとtmpdir上の一時ファイルに保存されます。MySQLのLOAD DATA文は単一トランザクションで実装されており、ロードするファイルとほぼ同じサイズのバイナリログがtmpdirに出力されることになります。

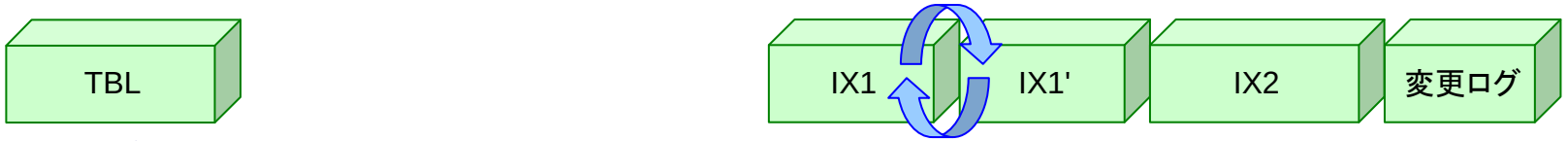
- セカンダリインデックス作成、外部キー制約付与の際は、作成されるセカンダリインデックスサイズの2倍にパラメータinnodb_online_alter_log_max_sizeで指定した値を加えた容量がtmpdirに要求されます。
  - セカンダリインデックスに応じた容量が要求されるのは、Fast Index Creationを行う場合です。テーブルコピーを伴う場合は必要ありません。
  - セカンダリインデックスを一度に複数作成すると、消費容量が増えます。
  - innodb_online_alter_log_max_sizeは、Online DDL中に変更されたデータを一時的に保存しておくログファイルです。デフォルト値は128MBytesです。

Fast Index Creationの内部動作

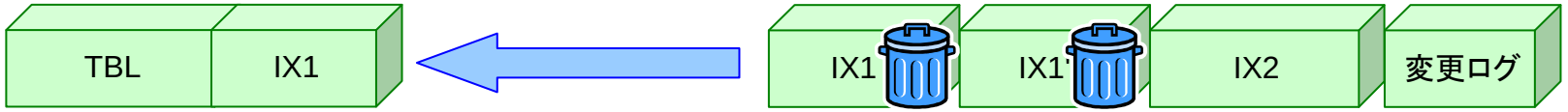
1. それぞれのインデックスについて、対象カラムのデータをtmpdirに書き出します。



2. データをソートしてインデックスを作成します。このとき2倍の容量を消費します。

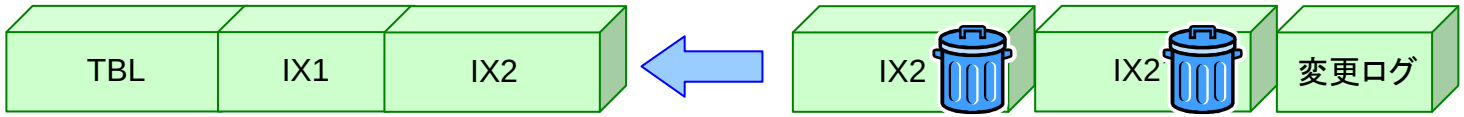


3. インデックスの作成が終わったらdatadirに書き戻し、一時ファイルを削除します。

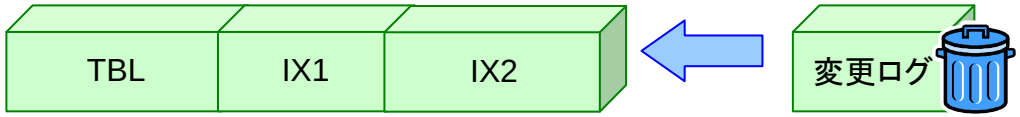


4. 同様に、2つ目のインデックスを作成します。

5. datadirに書き戻し、一時ファイルを削除します。



6. インデックス作成中に変更されたデータを反映し、変更ログを削除します。



datadir　　tmpdir

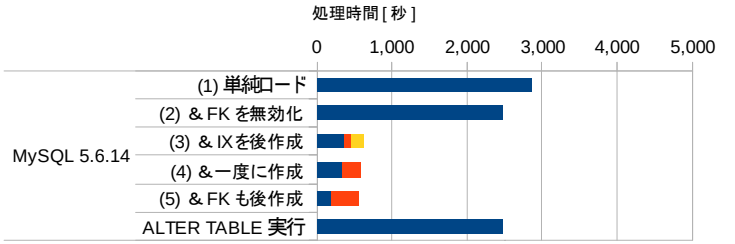
tmpdirの設定方針

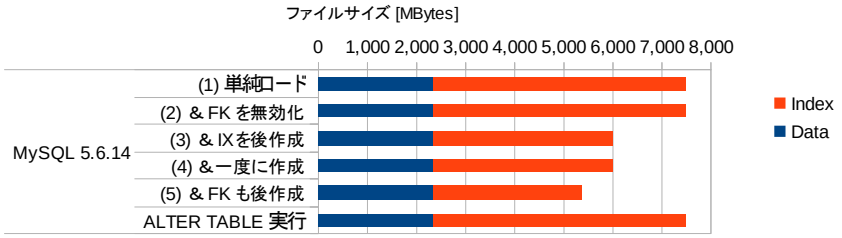
- Amazon EC2の場合、Amazon Linux AMIのボリュームサイズがデフォルトで8GBytesであること、tmpdirのデフォルト値が/tmpであることに注意してください。例えばMySQL用に100GBytesのEBSボリュームを追加したにも関わらずtmpdirがデフォルトのままになっていると、データロード、セカンダリインデックス作成や外部キー制約付与に失敗する可能性があります。

- Amazon RDSではtmpdirが適切に設定されているため、問題ありません。

- tmpdirに要求される容量を正確に見積もることは難しいため、基本的にdatadirと同じボリュームを使用することをおすすめします。

テーブルの再編成について





- 最適な手順でデータロードを行ったあとALTER TABLE order_line ENGINE = InnoDBとしてテーブルの再編成を行うと、長時間かかってしまううえにセカンダリインデックスがかえって断片化してしまいます。

- これはALTER TABLE文が内部的にパターン2で処理を行っているためです。本件について、FacebookのMark Callaghan氏がFeature Requestを提出しています。
  http://bugs.mysql.com/bug.php?id=57583

- サービスを停止できる場合は、以下の手順で再編成を行ってください。
  1. セカンダリインデックスと外部キー制約を削除
  2. ALTER TABLE文を実行
  3. foreign_key_checksをOFFに変更
  4. セカンダリインデックスと外部キー制約を再作成
  5. foreign_key_checksをONに戻す

宿題

1. 今回ご紹介したデータロード手順は、テーブルにデータが入っていないところから作業を開始するときのものです。例えばデータが3,000万件入っているところに1,000万件追加する場合は、どの手順が最も速いでしょうか。調べてみてください。