データベース負荷テストツールまとめ(5)
(公開版)

JPOUG> SET EVENTS 20120721

2012/07/21 平塚 貞夫

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＝8：2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

- 以下のブログエントリの続きです。
  - 第1回 http://d.hatena.ne.jp/sh2/20090802
  - 第2回 http://d.hatena.ne.jp/sh2/20090816
  - 第3回 http://d.hatena.ne.jp/sh2/20100112
  - 第4回 http://d.hatena.ne.jp/sh2/20100510

- 過去4回分は、実は今回のための伏線でした。

- 伏線を張ったまま2年ほど忘れていたので、回収しに来ました。

前回までの復習

負荷テストツールのカテゴリ

- 負荷テストツールを以下のようにカテゴリ分けしてみました。

A：実際のウェブアプリケーションに対して負荷テストを行うツール
- Apache Bench
- Apache JMeter
- HP LoadRunner

B：モデル化されたウェブアプリケーションに対して負荷テストを行うツール
- SPECjbb2005

C：実際のアプリケーションをもとにしてデータベースの負荷テストを行うツール
- Oracle Real Application Testing (Database Replay)

D：モデル化されたアプリケーションに対してデータベースの負荷テストを行うツール
- SysBench
- pgbench
- tpcc-mysql

- 仕事で一番役に立つのは、カテゴリAのツールだと思います。
- 本日はカテゴリDのツールについて話をします。

カテゴリDのツールの利用用途

- カテゴリDのツールをシステム開発やウェブサービス開発の現場で使うことは少ないと思いますが、意外といろいろな場面で役に立ちます。

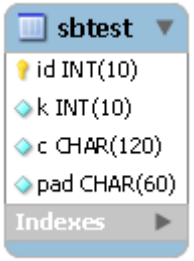
- DBサーバを最新機種に更改したときの性能を調べる
- ストレージをSSDに入れ替えた場合の性能を調べる
- 仮想化における性能低下率を調べる

- MySQLを5.5から5.6にアップグレードしたときの性能を調べる
- Oracle DatabaseをPostgreSQLにリプレイスしたときの性能を調べる
- パラメータチューニングによってどの程度性能が向上するかを調べる

- どの程度負荷をかけるとレプリケーションが遅延するのか調べる
- オンラインバックアップ取得中の性能低下率を調べる

- 負荷テスト中のStatspack/AWRレポートを取得し、性能分析手法を学習する
- 負荷テスト中のスロークエリログを取得し、性能分析手法を学習する

SysBench



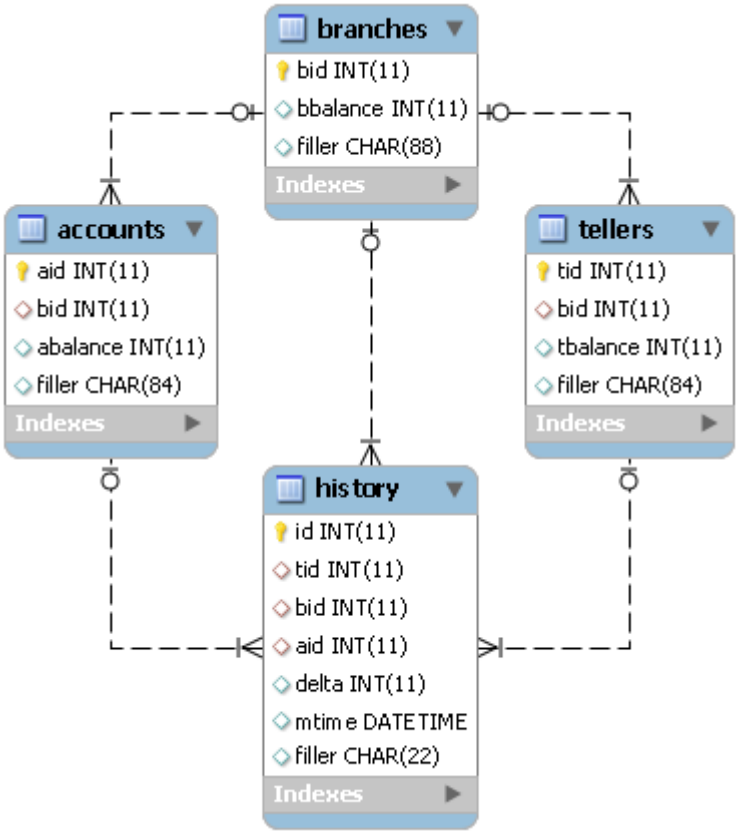
- 対応RDBMS：Oracle Database、MySQL、PostgreSQL
- 対応OS：Linuxなど
- 言語：C
- 作者：Alexey Kopytov氏
- ライセンス：GPLv2
- トランザクション仕様：
  独自、およびユーザ指定(0.5以降)

- SysBenchはファイルI/Oやメモリ転送などいくつかの測定をすることができ、機能の一つとしてデータベースの負荷テストを行うことができます。

- Facebookのエンジニア、MariaDBのエンジニアが好んで使っている印象です。

- Launchpadからバージョン0.5の開発版を持ってくることをおすすめします。0.5でLuaスクリプトに対応しています。

pgbench



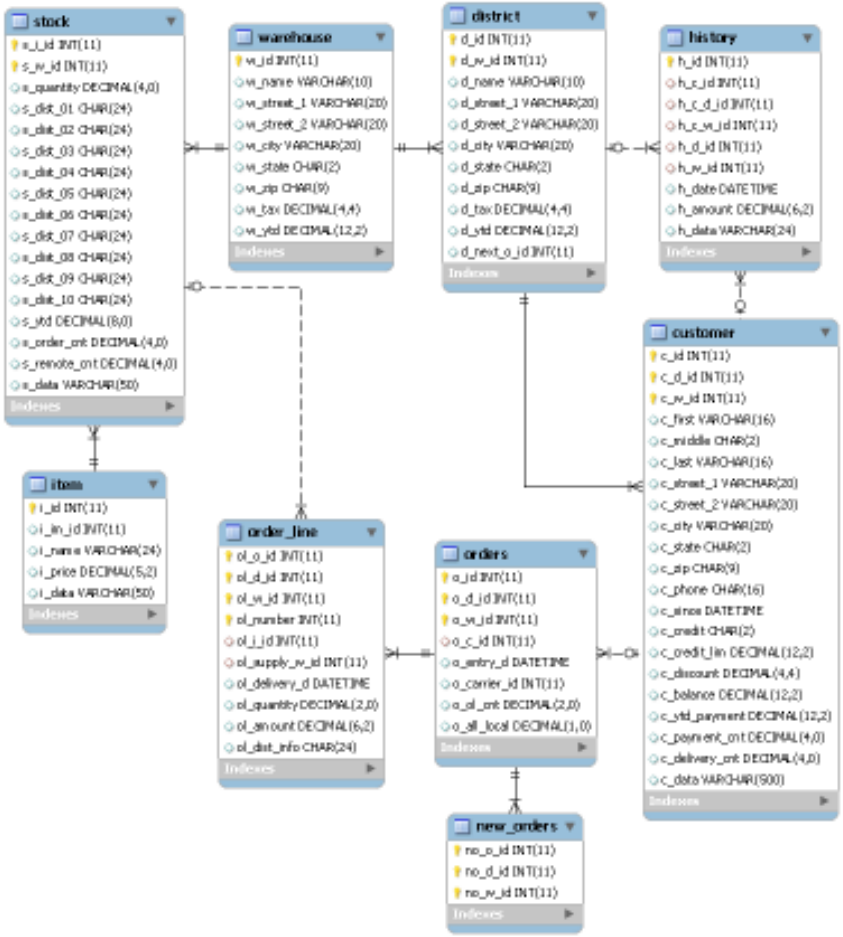
- 対応RDBMS：PostgreSQL
- 対応OS：Linuxなど
- 言語：C
- 作者：石井達夫氏
- ライセンス：BSDに類似
- トランザクション仕様：
  TPC-Bベース、およびユーザ指定

- accounts、tellers、branchesテーブルをUPDATEしてhistoryテーブルにINSERTするというトランザクションを繰り返し実行します。

- コミュニティでよく使われているもので、PostgreSQL 9.2における性能改善にも役立ったそうです。

- mysqlbenchというMySQLへの移植版が存在します。

tpcc-mysql



- 対応RDBMS：MySQL
- 対応OS：Linuxなど
- 言語：C
- 作者：木下靖文氏
- ライセンス：不明
- トランザクション仕様：TPC-Cベース

- 注文、支払いなど5種類のトランザクションをミックスして実行します。

- 当時のMySQLの性能の低さを白日の下に晒した極めて重要なツールで、InnoDB Plugin以降の性能改善に大きく貢献しました。

- 過去Oracle Database版とPostgreSQL版も作られたのですが、一般には公開されていません。

既存ツールの課題

- 多くのツールがLinuxをターゲットにしています。
  - Windowsでも利用したい
  - Solaris、HP-UX、AIXでも利用したい

- 多くのツールが一つのRDBMSだけをターゲットにしています。
  - pgbenchをOracle Databaseで動かしたい
  - tpcc-mysqlをOracle DatabaseやPostgreSQLで動かしたい
  - SysBenchは複数のRDBMSに対応しているが、PostgreSQLのコードにFIXMEと書いてある

- 特に、TPC-Cの実装でtpcc-mysql以外にこれといって良いものがありません。
  - 第2回を参照

JdbcRunner - 汎用データベース負荷テストツール

JdbcRunner - 汎用データベース負荷テストツール

JdbcRunner - 汎用データベース負荷テストツール

Overview

JdbcRunnerは各種データベースを対象とした負荷テストツールです。
スクリプトでトランザクションを定義して多重実行し、スループットとレスポンスタイムを測定することができます。

また、JdbcRunnerはOracle、MySQL、PostgreSQLを対象とした以下のテストキットが付属しており、
ユーザが独自にスクリプトを作成する以外に、これらを用いた負荷テストを行うことも可能となっています。

- Tiny SysBench - SysBench OLTPベンチマークの簡易版
- Tiny TPC-B - TPC-Bの簡易実装
- Tiny TPC-C - TPC-Cの簡易実装

Download

JdbcRunnerは、Vectorソフトライブラリからダウンロードできます。

- Vector: JdbcRunner (その他 / JAVA・JAVA Script)

Manual

本体に付属しているマニュアルと同じものを、オンラインで提供しています。

- JdbcRunner v1.2 documentation
  - 1. JdbcRunner の概要
  - 2. チュートリアル
  - 3. 負荷テストの流れ
  - 4. スクリプトの書き方
  - 5. 設定パラメータ
  - 6. スクリプトAPIリファレンス
  - 7. その他の付属ツール
  - 8. テストキット Tiny SysBench
  - 9. テストキット Tiny TPC-B
  - 10. テストキット Tiny TPC-C

Blog

JdbcRunnerと関連の深いはてなダイアリーのエントリです。

- データベース負荷テストツールまとめ(1) - SHOの日記
  TPC-B、TPC-Wベースのツールについて紹介しています。
- データベース負荷テストツールまとめ(2) - SHOの日記
  TPC-Cベースのツールについて紹介しています。
- データベース負荷テストツールまとめ(3) - SHOの日記
  TPC-Hベースのツールについて紹介しています。
- データベース負荷テストツールまとめ(4) - SHOの日記
  TPC-Eベースのツールについて紹介しています。

- というわけで、作りました。
  http://dbstudy.info/jdbcrunner/
- 修正済みBSDライセンスです。
- 現在の最新バージョンは、1.2です。

OSとRDBMSに依存しない

- Javaで実装したので、OSに依存しません。
  - Java SE 6以降に対応

- JDBCドライバさえあれば、RDBMSを選びません。
  - ただし、ツールがSQLの方言を吸収してくれるわけではない
  - メインターゲットはOracle Database、MySQL、PostgreSQL

スクリプトによるシナリオの記述

- 負荷テストのシナリオをJavaScriptで記述します。
- JavaScript自体にSQLを発行する機能はないので、裏でJavaが頑張っています。
- pgbenchなら15行で書けます。

```
function run() {
    var tid = random(1, TID_SCALE * scale);
    var bid = Math.floor((tid - 1) / TID_SCALE) + 1;
    var aid = random(AID_SCALE * (bid - 1) + 1, AID_SCALE * bid);
    var delta = random(-999999, 999999);

    execute("UPDATE accounts SET abalance = abalance + $int WHERE aid = $int", delta, aid);
    query("SELECT abalance FROM accounts WHERE aid = $int", aid);
    execute("UPDATE tellers SET tbalance = tbalance + $int WHERE tid = $int", delta, tid);
    execute("UPDATE branches SET bbalance = bbalance + $int WHERE bid = $int", delta, bid);
    execute("INSERT INTO history (tid, bid, aid, delta, mtime, filler) "
        + "VALUES ($int, $int, $int, $int, $timestamp, $string)",
        tid, bid, aid, delta, new Date(), FILLER);
    commit();
}
```

- スクリプトですので、「このSQLにヒント句を付けたい」といったときにもその場ですぐに修正ができます。

Mozilla Rhinoの利用

- JavaScriptエンジンとしてRhino(ライノー、動物のサイ)を利用しています。

- TPC-Cを実装するためにif文やfor文などの制御構造を使いたかったというのが元の動機です。負荷テストのシナリオをXMLなどで定義する、あるいは独自のDSLを作るといった案がある中で、試行錯誤した結果いまの形に落ち着きました。

- Java VM上で別のスクリプト言語を動作させる試みは2006年のJava SE 6のころから流行りはじめたそうで、現在Groovy(独自言語)、Rhino(JavaScript実装)、JRuby(Ruby実装)、Jython(Python実装)などを利用することができます。

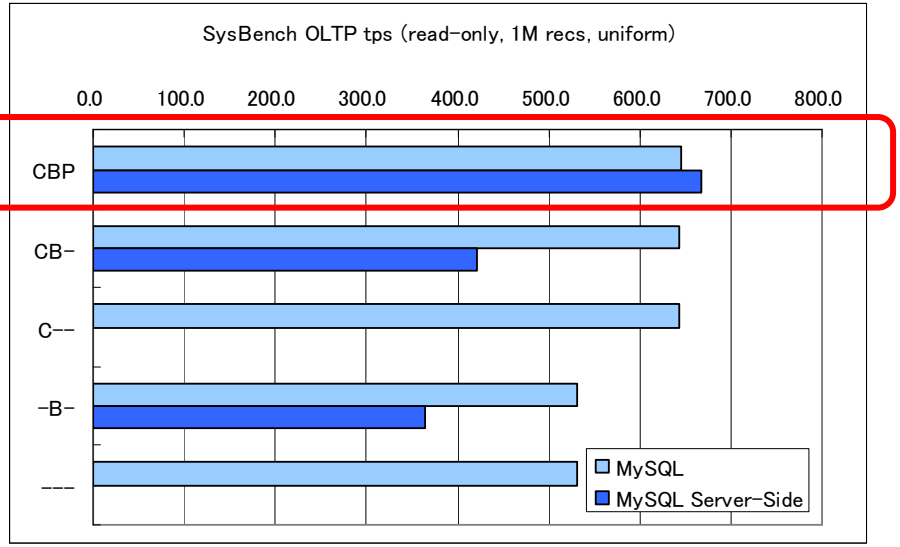
- これらのスクリプトエンジンの中からRhinoを選んだのは、当時この中ではRhinoが最も性能が良かったのと、JavaScriptなら読み書きできる人が多いだろうと考えたためです。

パラメータのバインド機構を利用

- Oracle Databaseは、いくつかの設計条件を満たさないとOLTPにおいてまるで性能が出ないことが知られています。
  - コネクションプールを利用する
  - パラメータのバインド機構を利用する
  - PreparedStatementプール機能を利用する

- 世の中にはバインド機構を利用していないツールが多いので、注意が必要です。

- JdbcRunnerの場合、普通にスクリプトを書けばこれらの条件が満たされるようになっています。



常にこの条件で測定できます。

テストキット

- 負荷テストのシナリオを三種類バンドルしています。

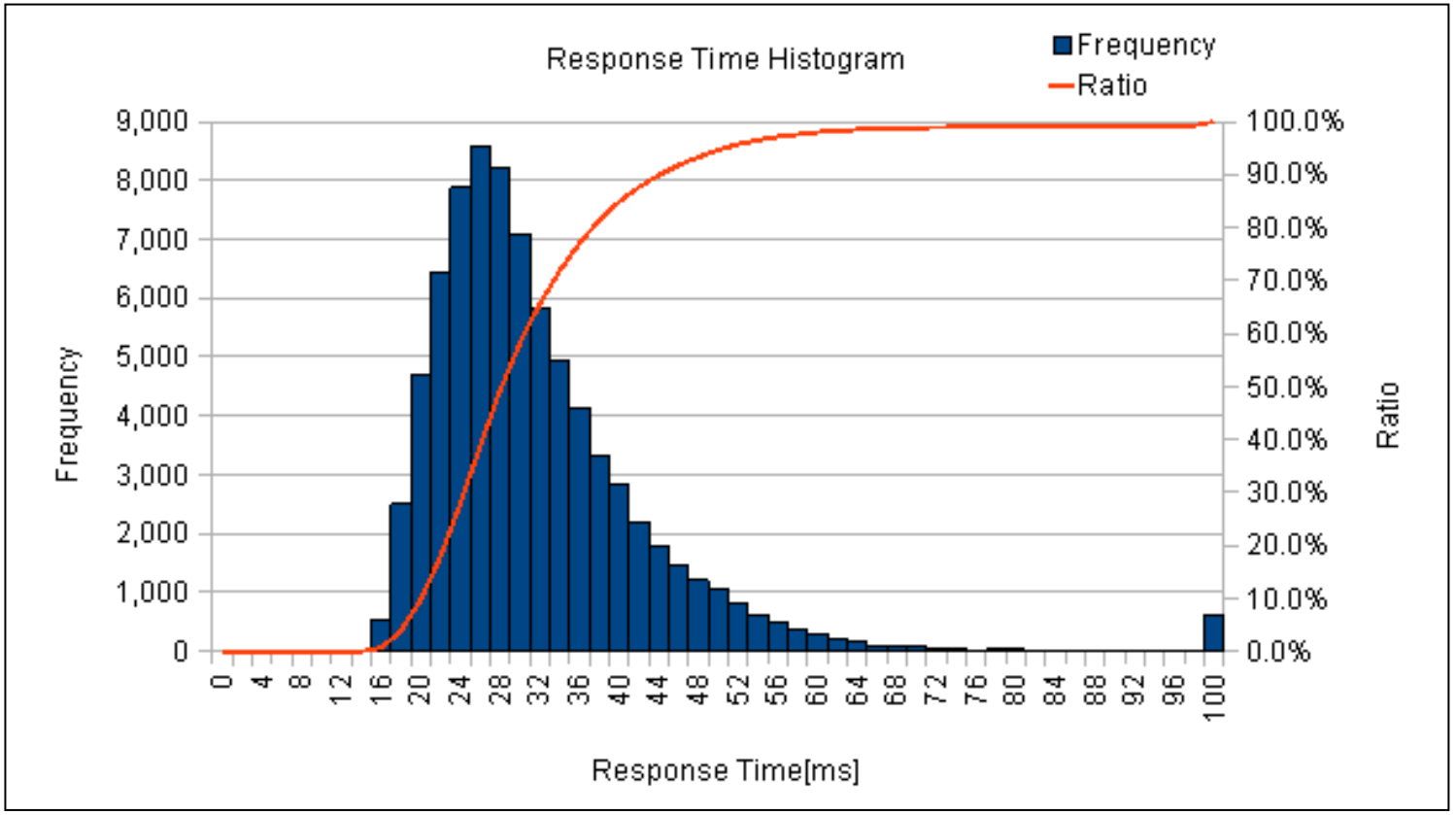
- Tiny SysBench
  - SysBench OLTPテストの移植版
  - Oracle Database、MySQL、PostgreSQLでテスト済み
  - PostgreSQLでFIXMEと書いてあった箇所は修正済み

- Tiny TPC-B
  - TPC-Bの簡易実装
  - Oracle Database、MySQL、PostgreSQLでテスト済み
  - pgbenchとほぼ同じ

- Tiny TPC-C
  - TPC-Cの簡易実装
  - Oracle Database、MySQL、PostgreSQLでテスト済み
  - tpcc-mysqlとは異なる

レスポンスタイムのヒストグラム

- レスポンスタイムを真面目に計測しています。
- 平均値ではなくて、中央値を出力します。
- レスポンスタイムのヒストグラムを作成するCalcシートをバンドルしています。



スロットル機能

- どのツールでも最大限の負荷をかけて限界性能を測定することはできるのですが、実際のシステムで常にCPU使用率が100%ということはありません。

- 冒頭で分類したカテゴリAのツールでは、処理ごとに思考遅延時間や入力待ち時間を設けて負荷を調節することが多いと思います。

- JdbcRunnerでもトランザクションごとのスリープ時間を指定することができるようになっています。ですが検証用のツールですのでもう少し工夫して、スループットの方を固定できるようにも作ってあります。

- 内部的には、指定したスループットになるようトランザクションごとのスリープ時間を逆算して、頑張っています。

測定例

NECさま＆日本HPさま事例

- HP ProLiant Server＋InfiniBand＋IOアクセラレータ＋CLUSTERPRO検証報告
  http://www.nec.co.jp/pfsoft/clusterpro/clp/download.html#wp

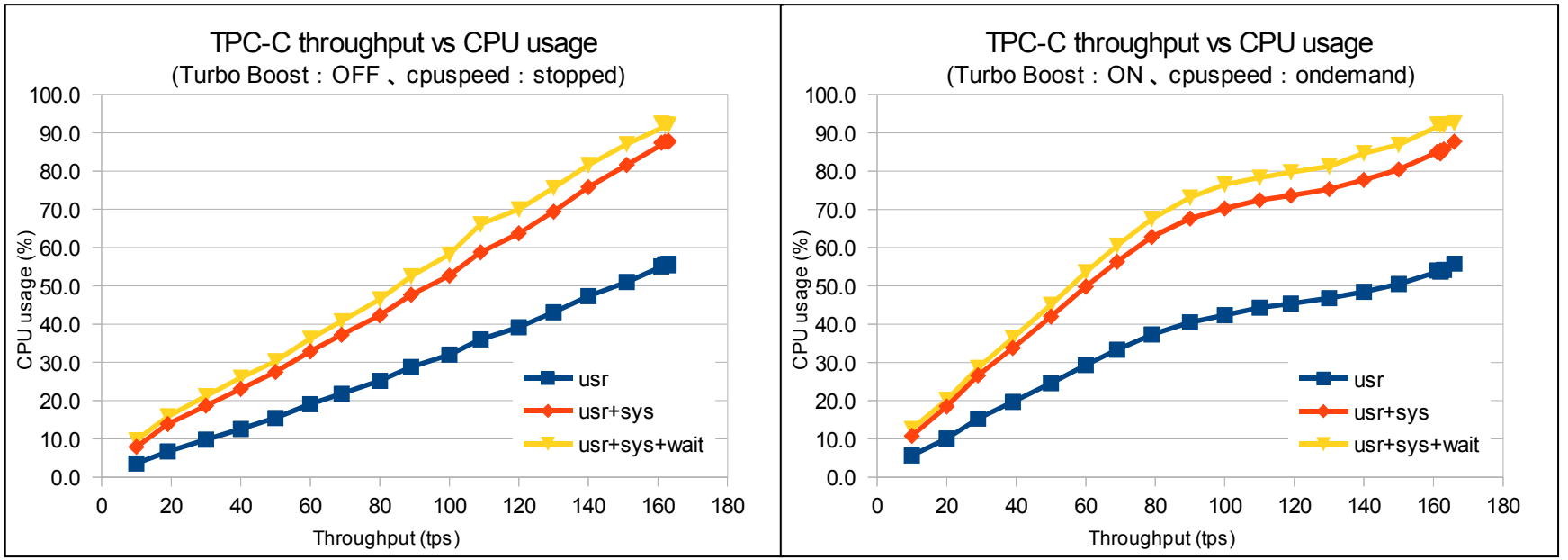
(報告書より引用)

- DBサーバにFusion-io社のioDriveを搭載し、CLUSTERPROのミラーリング機能を用いて冗長化したものです。

- JdbcRunnerのTiny TPC-Cを利用し、PostgreSQL 9.0とMicrosoft SQL Server 2008 R2で測定を行っています。

- 私はJdbcRunnerをSQL Serverで動かしたことは一度もなかったので、このレポートを見てびっくりしました。動くのですね。

スループットとCPU使用率の関係



- JdbcRunnerのスロットル機能を利用して、横軸にスループット、縦軸にそのときのCPU使用率をプロットしたグラフです。
- cpuspeedが有効化されていると低～中負荷のときにクロック周波数が下がるため、CPU使用率が相対的に高く見えてしまいます。
- 今年は節電のためサーバでもcpuspeedを有効にすることがあるかと思いますが、その際のCPU使用率監視には注意する必要があります。

正しい測定結果を得るためのポイント

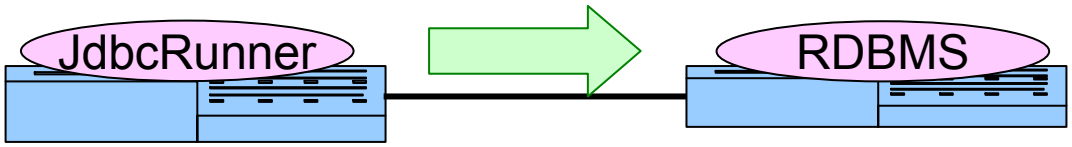
Javaにおけるベンチマークの難しさについて

- IBM developerWorksに素晴らしい記事がありますので、ご一読ください。特に仕事でJdbcRunnerを使う場合は必ずお読みください。

- Javaの理論と実践: 動的コンパイルとパフォーマンス測定 - IBM developerWorks
  http://www.ibm.com/developerworks/jp/java/library/j-jtp12214/

- 簡単にまとめます。
  - JavaバイトコードはHotSpotによって動的にコンパイルされるため、コンパイルされる前の遅い状態での測定結果は除外し、コンパイルされた後の測定結果のみを取得する必要がある
  - そのため対象の処理を指定回数繰り返し、経過時間を測定するという方法は適切ではない (JdbcRunnerは対象の処理を指定回数ではなく指定時間繰り返し、測定結果の前半を除外します)
  - 測定時間が短い場合、測定結果が一回のガベージコレクションの影響をより大きく受けてしまう。そのためガベージコレクションが複数回発生するだけの十分な測定時間を確保し、ガベージコレクションの影響を平準化するべきである

クライアント機とサーバ機を分ける



- JdbcRunnerを動作させるクライアント機とRDBMSを動作させるサーバ機を分けて、ネットワーク経由で負荷テストを行ってください。

- JdbcRunner自体がかなりCPUを使うので、同一マシンで負荷テストを行うと一体何を測定しているのか分からなくなってしまいます。

- 負荷テストのシナリオに依存しますが、可能であれば同じスペックのマシンを2台用意してください。最低でもサーバ機に対して半分以上の性能を持ったクライアント機を用意してください。

- CPUコア数の多いマシンにおいて、ネットワーク処理が一つのCPUコアに集中してボトルネックになることがあります。できるだけ新しいOS、新しいネットワークカード、新しいネットワークドライバをご利用ください。
  http://ja.community.dell.com/techcenter/b/weblog/archive/2010/12/20/rhel6.aspx

サーバVMを利用する

- Java VMには二種類あります。
  - クライアントVM
    起動時間とメモリ消費量を優先したもの。-clientオプションをつけて起動する
  - サーバVM
    性能を優先したもの。-serverオプションをつけて起動する

- 必ずサーバVMを利用してください。性能に約2倍の差があります。

- 特にWindowsのJREにはクライアントVMしか入っていないので、必ずJDKをインストールしておいてください。

- オプションを指定しなかったときにどちらのJava VMが利用されるかは、環境によって異なります。JDKインストール先のjvm.cfgというファイルに優先順位が記載されています。

```
-client KNOWN
-server KNOWN
-hotspot ALIASED_TO -client
-classic WARN
-native ERROR
-green ERROR
```

HotSpotコンパイルが収束するまでの時間を調べておく

- 前述したように、HotSpotコンパイルが収束してから実際の測定に入る必要があります。あらかじめJava VMに-XX:+PrintCompilationオプションを付与してHotSpotコンパイルが収束するまでの時間を調べておき、JdbcRunnerの-warmupTimeオプションにそれ以上の値を指定してください。

```
$ java -server -XX:+PrintCompilation JR tpcc.js

...

   8966 447       org.mozilla.javascript.NativeArray::js_sort (146 bytes)
   9087 448       org.mozilla.javascript.NativeNumber::execIdCall (345 bytes)
10:11:19 [INFO ] [Warmup] -292 sec, 94,94,10,10,9 tps, (496,498,50,50,49 tx)
   9467 449       org.mozilla.javascript.gen.tpcc_js_6::getParamOrVarName (940 bytes)
   9470 450       org.mozilla.javascript.gen.tpcc_js_6::getParamOrVarConst (808 bytes)
   9730 451       java.lang.StringBuilder::toString (17 bytes)
10:11:20 [INFO ] [Warmup] -291 sec, 93,95,8,10,10 tps, (589,593,58,60,59 tx)
  10357 452       oracle.jdbc.driver.T4CTTIrxd::unmarshalBVC (158 bytes)
  10486 453       sun.util.calendar.Gregorian$Date::getNormalizedYear (5 bytes)
  10581 454  !    oracle.jdbc.driver.OraclePreparedStatement::setDouble (29 bytes)

...

 81962 584  !    org.apache.commons.pool.impl.GenericObjectPool::borrowObject (908 bytes)
10:12:32 [INFO ] [Warmup] -219 sec, 89,90,10,10,8 tps, (7314,7317,732,732,731 tx)
10:12:33 [INFO ] [Warmup] -218 sec, 93,90,8,10,9 tps, (7407,7407,740,742,740 tx)
10:12:34 [INFO ] [Warmup] -217 sec, 101,103,11,8,11 tps, (7508,7510,751,750,751 tx)
10:12:35 [INFO ] [Warmup] -216 sec, 95,93,9,10,9 tps, (7603,7603,760,760,760 tx)
10:12:36 [INFO ] [Warmup] -215 sec, 94,95,10,9,10 tps, (7697,7698,770,769,770 tx)
```

HotSpotコンパイルをフォアグラウンドで行う

- -nAgents 200など多重度の高い設定では、HotSpotコンパイルが収束するまでかなりの時間を要することがあります。

- Java VMはデフォルトでHotSpotコンパイルをバックグラウンドで行います。メインの処理をインタプリタモードで実行しながら、HotSpotコンパイルを別のスレッドが実行します。ここでスレッド数が非常に多い場合、HotSpotコンパイルを行うスレッドがなかなかCPUを割り当ててもらえず、長時間にわたってインタプリタモードのまま動作してしまうことがあります。

- Java VMに-Xbatchオプションを付与すると、メインの処理を停止させてHotSpotコンパイルを行うように動作が変更されます。これによってHotSpotコンパイルが収束するまでの時間を大幅に短縮することができます。

- メインの処理を停止させてしまう点については、JdbcRunnerの-warmupTimeに十分な値を指定することで、測定結果から除外することが可能です。

Solarisに関する注意事項

- SolarisとJava SE 6の組み合わせで、測定終了時に例外java.sql.SQLRecoverableExceptionが発生することがあります。

- Java VMの不具合です。
  http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=4385444

- Java SE 7を利用するか、Java VMに-XX:-UseVMInterruptibleIOオプションを付与してください。

今後の予定

JdbcRunner 1.3

- 今日お披露目するつもりで作っていたのですが、間に合いませんでしたm(_ _)m
- 開発環境、各種ライブラリのアップデートと、細かな不具合の修正をしています。これまでの測定結果をご破算にするような大きな変更はありません。
- リポジトリを、自宅サーバのSubversionからGitHubへ移行しました。
  https://github.com/sh2/jdbcrunner
- あと10人日ぐらいでできると思います。

tpcc-mysql互換ローダ

- tpcc-mysqlを使ったことのある方はご存知かと思いますが、最初に行うテストデータのロードがかなり遅いです。

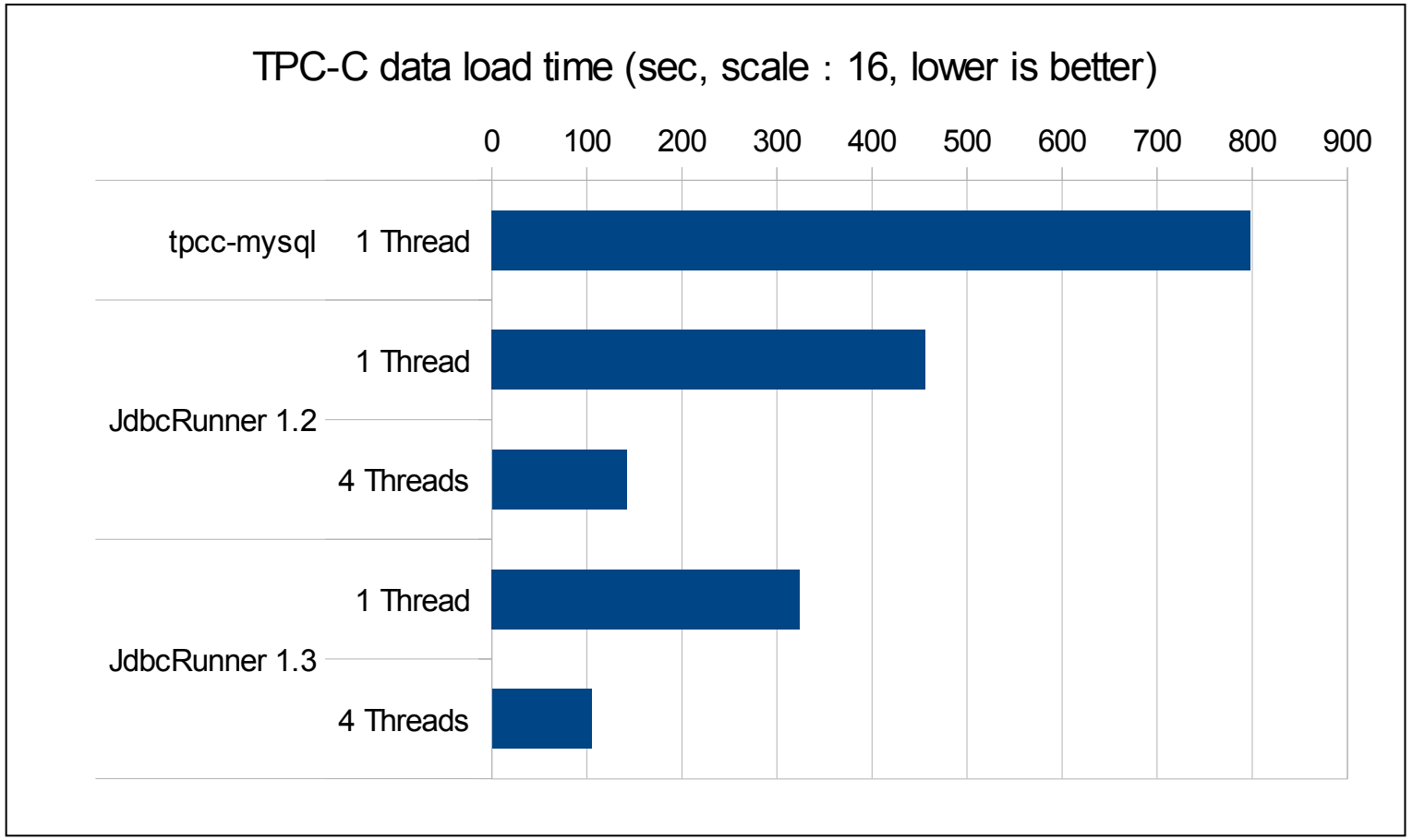
- JdbcRunnerのtpcc_load.jsの方が速いので、こちらをベースにしてテーブル、インデックス設計をtpcc-mysqlに合わせたバージョンを作っています。

- 性能測定にはtpcc-mysqlを使いたいが、作業時間をできるだけ短縮したいという方におすすめです。

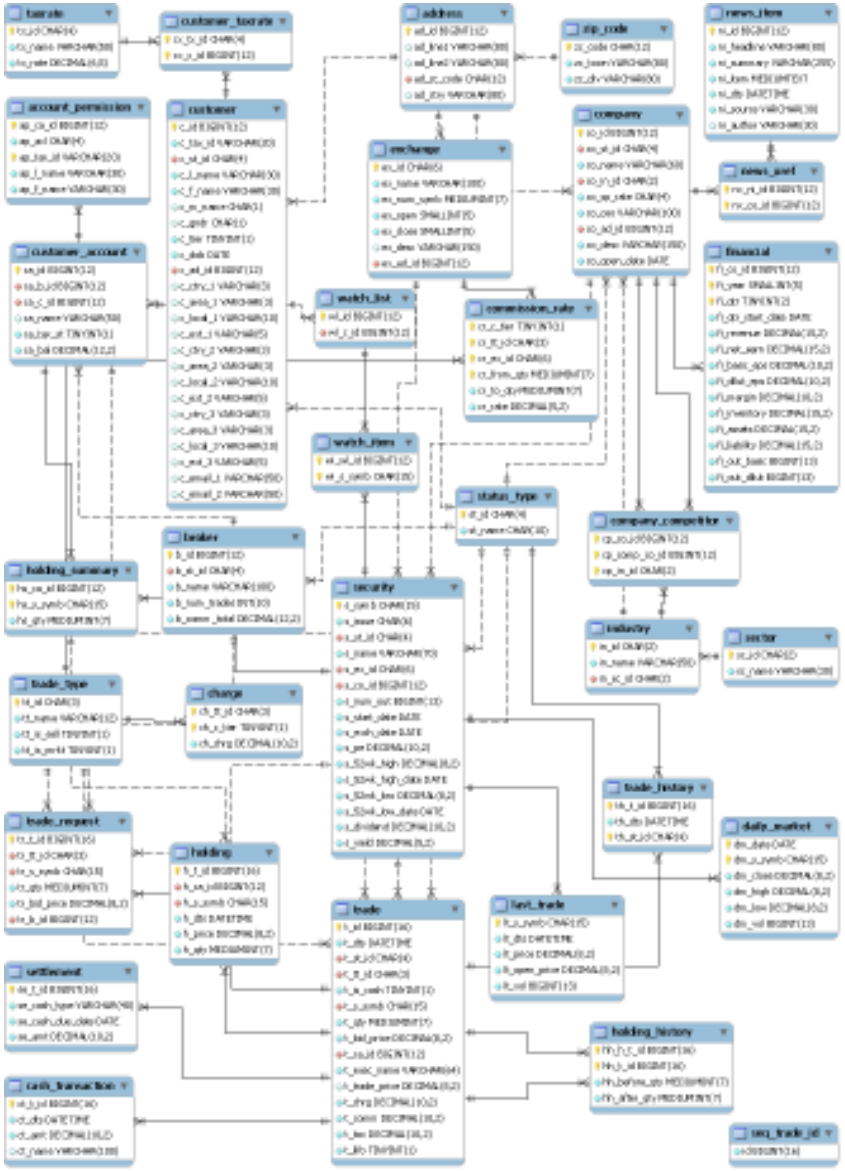
tpcc-mysql互換ローダ

- 現在のプロトタイプでtpcc-mysqlに比べて8倍速となっています。

- なぜ速いのかは、以下のエントリで解説しています。
  http://d.hatena.ne.jp/sh2/20090528



TPC-Eの実装



- 気の遠くなるようなER図ですが、いつかは実装したいと思っています。

Java SE 7/8対応

- Rhinoがあまり速くないというのが目下の課題です。

- プロファイリングをするとCPU負荷の半分がRhino、残りの半分がJDBCドライバといった状況です。RhinoはV8と比べると性能で10倍以上水をあけられているそうで、なんとも厳しいです。

- Java SE 7でinvokedynamicという命令が追加され、Java VM上で動作する動的言語のパフォーマンスを大きく向上させることができるようになりました。Rhinoは以下のリポジトリでinvokedynamicへの対応が進められているようです。
  https://github.com/mozilla/rhino/tree/invokedynamic

- Java SE 8に向けてNashorn(ナズホーン、ドイツ語でサイ)という新たなJavaScriptエンジンがオラクル社によって開発されています。個人的にはこちらを本命視しています。オープンソースソフトウェアとしてリリースされると伺っています。
  http://www.publickey1.jp/blog/12/javavmjavascriptecmascript5nashornjdk_8nodejs.html

英語ドキュメントの整備

- せっかく作ったので海外の方にも使っていただければと考えていますが、英語力の足りなさに苦慮しています。

- ツール本体は国際化対応済みです。

```
$ java JR
JdbcRunner 1.2
スクリプトファイルが指定されていません
```

```
$ LC_ALL=C java JR
JdbcRunner 1.2
Script file is not specified
```

- ツール本体のJavadocは日本語ですが、読む機会はあまりないかと思います。

- テストキットのコメントは英語で書いています。

- マニュアルが日本語版しかない上に分量が多いので、ここが課題です。

宿題

1. FreeBSD、HP-UX、AIXでの動作報告をお待ちしています。

2. DB2、HiRDB、Symfoware、Firebirdでの動作報告をお待ちしています。

3. Tiny TPC-Cですが、実はもう少し速くなります。チューニングに挑戦してみてください。