OpenSource/Linux技術文書

Hadoop HiveとMySQLの利用例

～ Hadoop HiveとMySQLのパフォーマンスを比較してみる ～

日本ヒューレット・パッカード株式会社
ESSNプリセールス統括本部
エンタープライズサーバー・ストレージ技術第一本部
Linuxソリューション部
古賀　政純

2012年2月29日

目次

図表目次

[本ドキュメントについて]

- 本ドキュメントでは、NameNode サーバーのみでの作業を(NN)、DataNode サーバーのみでの作業を(DN)、ファイルサーバーでの作業を(FS)、NameNode サーバーと DataNode サーバー両方での作業を(NN,DN)と記すことにします。
  - 例 1)

  「ファイルをコピーします。（NN）」と記載してあるものは、NameNode だけでファイルをコピーするという意味になります。
  - 例 2)

  「rpm コマンドでパッケージをインストールします（NN,DN）」と記載してあるものは、NameNode と DataNode の両方で rpm コマンドを使ってインストールを行うという意味になります。

- コマンドラインでの入力が長く紙面の都合で折り返して記載する場合は、下記のように「¥」記号を挿入して複数行にわたって記載しています。複数行にわたって記載されていても実際には１行で入力するものは、その記述の最後に「（実際には１行で入力）」を挿入しています。
  - 例 3)

  ```
  # alternatives --install /etc/hadoop-0.20/conf hadoop-0.20-conf ¥
  /etc/hadoop-0.20/conf.hp001 20　（実際には１行で入力）
  ```

- 本ドキュメントの内容については充分チェックをしておりますが、その正確性を保証する物ではありません。また、将来、予告なしに変更することがあります。
- 本ドキュメントの使用で生じるいかなる結果も利用者の責任となります。日本ヒューレット・パッカード株式会社は、本ドキュメントの内容に一切の責任を負いません。
- 本ドキュメントの技術情報は、ハードウェア構成、OS、アプリケーションなど使用環境により大幅に数値が変化する場合がありますので、十分なテストを個別に実施されることを強くお薦め致します。
- 本ドキュメント内で表示・記載されている会社名・サービス名・商品名等は各社の商標又は登録商標です。
- 本ドキュメントで提供する資料は、日本の著作権法、条約及び他国の著作権法にいう著作権により保護されています。

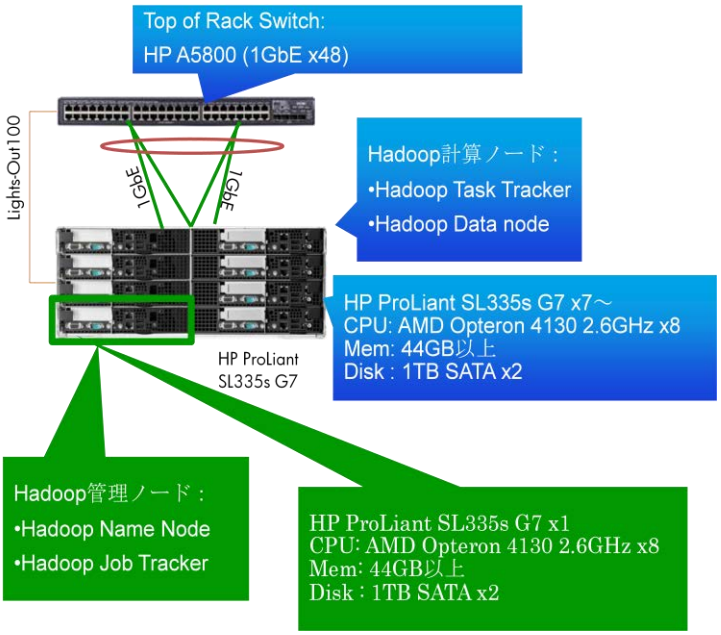
本ドキュメントはCloudera distribution for Hadoop 3U2 (通称CDH3U2)のHadoop HiveとMySQLの処理時間を比較するためのガイドです。

システム構成

以下にHadoopが稼働する環境を示します。

ハードウェア　　: HP ProLiant SL6500 (Server: HP ProLiant SL335s G7 x8)
OS　　: Red Hat Enterprise Linux 6.1 x86-64 (NameNodeおよびDataNode)
JDK　　: 1.6.0u29 x64 (Oracle社が提供するパッケージを利用)
CDH　　: CDH3U2　(Red Hat系OSに対応したRPMパッケージを利用)

以下にハードウェア外観を示します。今回の構成ではNameNodeの可用性は考慮していないため、NameNodeの障害時のデータロストが発生するSPOFが存在する点にご注意ください。



- Proof Of Concept 構成
- Hadoop の Name Node は 1 台構成
- Name Node は、Job Tracker を兼用
- Name Node の可用性はなし
- システムの可用性は HDFS のレプリカで担保できるが、ロストしてもよいレベルの動作確認用途
- 容量を確保するため、SATA のディスクを搭載
- 障害による交換頻度を下げたい場合は、SAS を検討
- Top of Rack スイッチはギガビットのポート数を多数もつものを用意
- HP A5800 では 48 ポートが利用可能

図 1. Cloudera Distribution for Hadoop on HP ProLiant SL6500 のシステム構成例

上記の8台のHadoopノードとは別に、hiveコマンドによりHadoopクラスターにジョブを投入するHiveクライアントとなるサーバーを1台用意します。以下、Hiveクライアントとなるサーバーの環境です。

ハードウェア　　: HP ProLiant DL160 G5 x1
CPU　　: Intel(R) Xeon X5472 3.00GHz x8
メモリ　　: 6GB
内蔵ディスク　　: SATA 750GB x1、SATA 250GB x1、SATA 2TB x2
OS　　: Red Hat Enterprise Linux 6.1 x86-64

CPU については、HP ProLiant SL335s G7 および DL160 G5 ともにコア数を示します。SL6500 の各ノード HP ProLiant SL335s G7 は、HP ProLiant DL160 G5 と HP A5800 を通じて同一 LAN セグメントで疎通できる環境です。ユーザーは ProLiant DL160 G5 で hive コマンドによりジョブを投入し、Hadoop のジョブは SL335s G7 の 8 台で実行されます。

Hadoop のインストール(FS)

ジョブ投入用マシンとなる ProLiant DL160 G5 に Hadoop をインストールします。

```
# rpm -vhi hadoop-0.20-0.20.2+923.142-1.noarch.rpm
```

Hadoop ノード 8 台の構築に利用した Java Runtime ライブラリー（jre-6u29-linux-amd64.rpm）を rpm コマンドで Hive クライアントにもインストールします。Hadoop が提供する/etc/hadoop ディレクトリ配下の各種設定、Java Runtime ライブラリーのインストール等については日本 HP の Linux 技術情報サイトにて提供している技術文書「Cloudera Distribution for Hadoop インストール手順及び利用例」を参照し設定して下さい。

「Cloudera Distribution for Hadoop インストール手順及び利用例」の入手先：

https://www.hpe.com/jp/ja/servers/linux/technical/edlin.html

ただし、Hive クライアントでは、NameNode や DataNode 等の Hadoop の Master/Slave ノードの各種サービスのインストールや起動は不要ですので、Hadoop の/etc/hadoop ディレクトリ配下の環境設定までを行ってください。

Hadoop Hive のインストール(FS)

Hive クライアントとなる ProLiant DL160 G5 に Hadoop Hive をインストールします。ユーザーが hive コマンドを利用するために必要です。

```
# rpm -vhi hadoop-hive-0.7.1+42.27-2.noarch.rpm
```

Hadoop Hive への入力用のデータ作成(FS)

販売店テーブル(47 レコード/ファイルサイズ 約 2KB)として、都道府県毎に 1 支店のデータ shop.csv を作成します。ファイルに日本語が含まれる場合は、ターミナルエミュレーター等での日本語表示を考慮し、LANG 環境変数が ja_JP.UTF-8 になっているか確認してください。レコード数（この場合 47）は wc -l で確認できます。

```
# cat /etc/sysconfig/i18n/
LANG="ja_JP.UTF-8"

# mkdir -p /data/
# chmod -R 777 /data
# cd /data/
# vi shop.csv
1,札幌支店,北海道,札幌市
2,青森支店,青森県,青森市
3,盛岡支店,岩手県,盛岡市
4,仙台支店,宮城県,仙台市
5,秋田支店,秋田県,秋田市
6,山形支店,山形県,山形市
7,福島支店,福島県,福島市
8,水戸支店,茨城県,水戸市
9,宇都宮支店,栃木県,宇都宮市
10,前橋支店,群馬県,前橋市
11,さいたま支店,埼玉県,さいたま市
12,千葉支店,千葉県,千葉市
13,東京本店,東京都,新宿区
14,横浜支店,神奈川県,横浜市
15,新潟支店,新潟県,新潟市
16,富山支店,富山県,富山市
17,金沢支店,石川県,金沢市
18,福井支店,福井県,福井市
```

19,甲府支店,山梨県,甲府市
20,長野支店,長野県,長野市
21,岐阜支店,岐阜県,岐阜市
22,静岡支店,静岡県,静岡市
23,名古屋支店,愛知県,名古屋市
24,津支店,三重県,津市
25,大津支店,滋賀県,大津市
26,京都支店,京都府,京都市
27,大阪支店,大阪府,大阪市
28,神戸支店,兵庫県,神戸市
29,奈良支店,奈良県,奈良市
30,和歌山支店,和歌山県,和歌山市
31,鳥取支店,鳥取県,鳥取市
32,松江支店,島根県,松江市
33,岡山支店,岡山県,岡山市
34,広島支店,広島県,広島市
35,山口支店,山口県,山口市
36,徳島支店,徳島県,徳島市
37,高松支店,香川県,高松市
38,松山支店,愛媛県,松山市
39,高知支店,高知県,高知市
40,福岡支店,福岡県,福岡市
41,佐賀支店,佐賀県,佐賀市
42,長崎支店,長崎県,長崎市
43,熊本支店,熊本県,熊本市
44,大分支店,大分県,大分市
45,宮崎支店,宮崎県,宮崎市
46,鹿児島支店,鹿児島県,鹿児島市
47,那覇支店,沖縄県,那覇市

会員情報テーブル(1000 レコード/ファイルサイズ 約 36KB)を作成します。以下 URL のサイトでランダムな人名・電話番号のデータを生成します。出力形式を CSV、出力数を 1000 件、データの項目に、名前、性別、年齢、電話番号を指定しデータを生成します。ファイルを保存する際に拡張子を「.csv」にします。
http://kazina.com/dummy/

次ページのような CSV ファイル member.csv を得るために上記サイトで得られた CSV ファイルを編集します。CSV ファイルの行頭の連番数字の生成と 1 行目の「名前　性別　年齢　電話番号」という行の削除をマイクロソフト社の Office 2007 の Excel で行います。以下に連番の生成例を示します。まず Excel で CSV ファイルを開き、連番を生成したいセルに数字の 1 を入れ、Excel のウィンドウから「編集」→「フィル」→「連続データの作成」を選択し、「連続データ」ウィンドウで範囲に「列」、種類に「加算」、増分値に「1」、停止値に「1000」を入力して「OK」を選択すると 1～1000 の連番が生成できます。

図 2. Excel での連番付与のウィンドウ

Excel でファイルを CSV 形式で保存します。今回はファイル名を member_temp1.csv とします。Linux のファイルサーバーに member_temp1.csv をコピーします。さらにコピーした CSV ファイルを Linux の UTF-8 形式に変換するため、ファイルサーバーに、nkf パッケージをインストールします。

nkf RPM パッケージ
http://mirror.centos.org/centos/6/os/x86_64/Packages/nkf-2.0.8b-6.2.el6.x86_64.rpm

```
# rpm -vhi nkf-2.0.8b-6.2.el6.x86_64.rpm
```

コピーした CSV ファイルのファイル形式を確認いておきます。

```
# cd /root/
# file member_temp1.csv
member_temp1.csv: Non-ISO extended-ASCII text, with CRLF, NEL line terminators
```

nkf によって、Linux の UTF-8 形式のファイルに変換します。

```
# nkf -w80 -Lu -d member_temp1.csv > member_utf8.csv
```

file コマンドで変換後の形式を確認します。

```
# file member_utf8.csv
member.csv: UTF-8 Unicode text
```

nkf コマンドによって変換された member_utf8.csv ファイルの形式が「UTF-8 Unicode text」と表示されるかを必ず確認してください。また、less コマンド等により文字化け表示がないことを確認してください。次に、member_utf8.csv ファイル中の氏名と電話番号、性別にダブルクォーテーションを付加します。これには awk コマンドを利用します。

```
#awk -F"," '{ print $1 ",¥"" $2 "¥",¥"" $3 "¥"," $4 ",¥"" $5 "¥"" }' member_utf8.csc ¥
> member.csv    ←実際には一行で入力
```

```
# pwd
/data/
```

```
# head -5 member.csv
1,"山田 太郎","090-xxxx-yyyy",72,"M"
2,"山田 花子","090-xxxx-yyyy",75,"M"
3,"田中 一郎","080-xxxx-yyyy",63,"M"
4,"鈴木 一朗","080-xxxx-yyyy",56,"F"
5,"鈴木 二郎","090-xxxx-yyyy",52,"F"
```

```
# wc -l member.csv
1000 member.csv
```

品目情報テーブル(150 レコード/ファイルサイズ 約 3KB)を表示します。

```
# head -5 item.csv
1,品目 1,100,L,M,S
2,品目 2,100,L,M,S
3,品目 3,100,L,M,S
4,品目 4,100,L,M,S
5,品目 5,100,L,M,S
```

```
# wc -l item.csv
150 item.csv
```

売上明細テーブルを作成します。売上明細テーブルの生成用スクリプト mkdata.py を作成します。

```
#!/usr/bin/python
# -*- encoding:utf-8 -*-
from datetime import date, timedelta
import sys
import random  # for random number
initial_date = date(2010,11,1)
argvs = sys.argv
if len(argvs) != 2:
   print "Usage: mkdata start_no"
   sys.exit()
start_no = int(argvs[1]) + 1
end_no = start_no + 1000000000
# Main loop
for i in xrange(start_no,end_no):
    # Get random date
    rnd_day = random.randint(1,365)    # from 2010/11/01 to 2011/11/31
    hd_date_tmp = initial_date + timedelta(days=rnd_day)
    hd_date = hd_date_tmp.strftime('%Y-%m-%d')

    # get empno(random 1..10)
    hd_empno = str(random.randint(1,10))

    # get shopno(random 1..47)
    hd_shopno = str(random.randint(1,47))

    # get regno(random 1..10)
    hd_regno = str(random.randint(1,10))

    # get memberno(random 1..1000)
    hd_memberno = str(random.randint(1,1000))

    # get itemno(random 1..150)
    hd_itemno_tmp = random.randint(1,150)
    hd_itemno = str(hd_itemno_tmp)

    # get price(gitem table)
    hd_price = str(((hd_itemno_tmp/10)*100)+100)

    # get num(random 1..10)
    hd_num = str(random.randint(1,10))

    # Main proc
    if i%1000000 == 0:
        sys.stderr.write(str(i)+"¥n")
```

```
    outstr = str(i) +  "," + hd_date + "," + hd_empno + "," + hd_shopno + "," + ¥
             hd_regno + "," + hd_memberno + "," + hd_itemno + "," + hd_price + ¥
             ","  + hd_num
    print outstr
```

10 億レコードの売上明細テーブルの生成用スクリプト mkdata.py を使って、売上明細テーブルの CSV ファイル data00.csv を生成します。mkdata.py スクリプトの使用例を以下に挙げます。

例1)
```
# ./mkdata.py 1 > data00.csv
# head -2 data00.csv
1,2011-07-15,9,41,7,368,41,500,1
2,2011-01-28,2,44,1,1000,118,1200,6

# tail -2 data00.csv
999999999,2011-04-14,3,44,2,479,33,400,1
1000000000,2010-11-11,1,33,9,587,8,100,10
```

例2)
```
# ./mkdata.py 1000000001 > data01.csv
# head -2 data01.csv
1000000001,2011-09-20,3,5,10,663,68,700,1
1000000002,2011-10-23,6,10,7,938,35,400,2

# tail -2 data01.csv
1999999999,2011-01-15,5,37,7,645,141,1500,7
2000000000,2011-04-14,8,8,8,376,76,800,10
```

上記の例 1 を実行します。データはファイルサーバーの/data に出力します。10 億レコードのファイルサイズは約 39GB となりました。
```
# pwd
/data
# ./mkdata.py 1 > data00.csv
# ls -lh data00.csv
-rw-r--r--. 1 root root 39G  9 月 21 23:50 2011 data00.csv
```

Hive 動作確認用の数レコードからなる売上明細テーブル small.csv を作成します。今回は 5 レコードのデータを作成しました。
```
# head -5 data00.csv > small.csv
# cat small.csv
1,2011-07-15,9,41,7,368,41,500,1
2,2011-01-28,2,44,1,1000,118,1200,6
3,2011-04-02,2,20,10,828,37,400,2
4,2011-08-07,2,42,4,395,94,1000,1
5,2011-09-12,7,41,4,157,98,1000,8

#pwd
/data
# ls -lF
-rw-r--r--. 1 root root 41577044585  9 月 21 23:50 2011 data00.csv
```

```
-rw-r--r--  1 root root        4645  1月 23 15:35 2012 item.csv
-rw-r--r--  1 root root       42227  1月 23 15:35 2012 member.csv
-rwxr-xr-x  1 root root        1466  1月 23 15:35 2012 mkdata.py*
-rw-r--r--  1 root root        2004  1月 23 15:35 2012 shop.csv
-rw-r--r--  1 root root         178  1月 16 12:25 small.csv
```

Hadoop Hive の DB 作成用 SQL 文を保存したファイルを作成。

Hadoop Hive でアクセスできるデータベース名は hadooptest とします。Hadoop Hive では MySQL と同様に create database 文を利用できます。

```
# cd/data/
# vi hd_db.sql
DROP DATABASE hadooptest;
CREATE DATABASE hadooptest;
SHOW DATABASES;
```

上記で作成した SQL 文を Hive にロードし、データベースを作成します。

```
# hive < hd_mkdb.sql
```

Hadoop Hive のテーブル作成用 SQL 文を保存したファイルを作成

```
# vi hd_table.sql
USE hadooptest;
CREATE TABLE gdata
(
  transno  int,
  datet    string,
  empno    int,
  shopno   int,
  regno    int,
  memberno int,
  itemno   int,
  price    int,
  num      int
) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' stored AS TEXTFILE;

CREATE TABLE gitem
(
  itemno  int,
  name    string,
  price   int,
  l       string,
  m       string,
  s       string
) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' stored AS TEXTFILE;

CREATE TABLE gmember
(
  memberno  int,
  name      string,
  tel       string,
  age       int,
```

```
  sex       string
) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' stored AS TEXTFILE;

CREATE TABLE gshop
(
  shopno  int,
  name      string,
  state     string,
  city      string
) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' stored AS TEXTFILE;
```

上記で作成した SQL 文を Hive にロードし、hadooptest データベースにテーブルを作成します。

```
# hive < hd_table.sql
```

Hadoop Hive のコマンドプロンプトでテーブルが作成されているかを show tables で確認します。

```
# hive
hive> show databases;
OK
default
hadooptest
Time taken: 2.971 seconds

hive> use hadooptest;
OK
Time taken: 0.016 seconds
hive> show tables;
OK
gdata
gitem
gmember
gshop
Time taken: 0.372 seconds
hive>quit;
#
```

作成した Hadoop Hive のテーブルに CSV ファイルのデータをセットする SQL 文が書かれたファイルを用意します。動作確認用の 5 レコードのデータをテストする場合は、small.csv を SQL 文が書かれたファイル hd_loadsmallcsv.sql に記述します。

```
# pwd
/data
# vi hd_loadsmallcsv.sql
USE hadooptest;
LOAD DATA LOCAL INPATH '/data/item.csv'    INTO TABLE gitem;
LOAD DATA LOCAL INPATH '/data/member.csv'  INTO TABLE gmember;
LOAD DATA LOCAL INPATH '/data/shop.csv'    INTO TABLE gshop;
LOAD DATA LOCAL INPATH '/data/small.csv'   INTO TABLE gdata;
```

メンテナンス用にテーブル削除用の SQL 文を記述したファイル hd_droptable.sql も用意しておきます。

```
# vi hd_droptable.sql
USE hadooptest;
DROP TABLE gitem;
DROP TABLE gmember;
DROP TABLE gshop;
DROP TABLE gdata;
```

上記の hd_loadsmallcsv.sql ファイルを使って、Hadoop Hive 上に作成した hadooptest データベースの各テーブル gitem、gmember、gshop、gdata に値をセットします。

```
# hive < hd_loadsmallcsv.sql
Hive history file=/tmp/root/hive_job_log_root_201201161313_710741549.txt
hive> USE hadooptest;
OK
Time taken: 2.851 seconds
hive> LOAD DATA LOCAL INPATH '/data/item.csv'    INTO TABLE gitem;
Copying data from file:/data/item.csv
Loading data to table gitem
OK
Time taken: 1.053 seconds
hive> LOAD DATA LOCAL INPATH '/data/member.csv' INTO TABLE gmember;
Copying data from file:/data/member.csv
Loading data to table gmember
OK
Time taken: 0.178 seconds
hive> LOAD DATA LOCAL INPATH '/data/shop.csv'    INTO TABLE gshop;
Copying data from file:/data/shop.csv
Loading data to table gshop
OK
Time taken: 0.15 seconds
hive> LOAD DATA LOCAL INPATH '/data/small.csv'  INTO TABLE gdata;
Copying data from file:/data/small.csv
Loading data to table gdata
OK
Time taken: 0.152 seconds
hive>
#
```

データがロードされているかを確認します。ここではテーブルの上から 3 行のみを表示させる SQL 文が書かれたファイル hd_disp3.sql を用意して hive にロードします。

```
# vi hd_disp3.sql
USE hadooptest;
SELECT * FROM gitem   limit 3;
SELECT * FROM gmember limit 3;
SELECT * FROM gshop   limit 3;
SELECT * FROM gdata   limit 3;
```

```
# hive < hd_disp3.sql
# hive < hd_disp3.sql
Hive history file=/tmp/root/hive_job_log_root_201201231910_1036982238.txt
hive> USE hadooptest;
```

```
OK
Time taken: 1.325 seconds
hive> SELECT * FROM gitem   limit 3;
OK
1       "品目1" 100     "L"     "M"     "S"
2       "品目2" 100     "L"     "M"     "S"
3       "品目3" 100     "L"     "M"     "S"
Time taken: 0.449 seconds
hive> SELECT * FROM gmember limit 3;
OK
1       "富田 夏空"     "062-812-xxxx"  21      "F"
2       "首藤 照生"     "083-978-yyyy"  26      "M"
3       "谷村 一徳"     "089-661-zzzz"  70      "M"
Time taken: 0.112 seconds
hive> SELECT * FROM gshop   limit 3;
OK
1       "札幌支店"      "北海道"        "札幌市"
2       "青森支店"      "青森県"        "青森市"
3       "盛岡支店"      "岩手県"        "盛岡市"
Time taken: 0.091 seconds
hive> SELECT * FROM gdata   limit 3;
OK
1       2011-07-15      9       41      7       368     41      500     1
2       2011-01-28      2       44      1       1000    118     1200    6
3       2011-04-02      2       20      10      828     37      400     2
Time taken: 0.1 seconds
hive>
#
```

Hadoop Hive による 5 レコードの POS データの処理

支店別売上合計の計算を行う SQL 文を記述したファイル hd_total.sql を作成し、hive で実行します。

```
# cd /data/
# vi hd_total.sql
USE hadooptest;
set mapred.map.tasks=8;
set mapred.reduce.tasks=35;
select name, sum(num*price) total from gshop gp join gdata gd on gp.shopno = gd.shopno
group by name;
```

```
# hive < hd_total.sql
Hive history file=/tmp/root/hive_job_log_root_201201231926_277186080.txt
hive> USE hadooptest;
OK
Time taken: 1.68 seconds
hive> set mapred.map.tasks=8;
hive> set mapred.reduce.tasks=35;
hive> select name, sum(num*price) total from gshop gp join gdata gd on gp.shopno = gd.shopno
group by name;
Total MapReduce jobs = 2
```

```
Launching Job 1 out of 2
Number of reduce tasks not specified. Defaulting to jobconf value of: 35
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_201201201658_0003, Tracking URL =
http:// hd01:50030/jobdetails.jsp?jobid=job_201201201658_0003
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=hd01:8021 -kill
job_201201201658_0003
2012-01-23 19:26:38,662 Stage-1 map = 0%,  reduce = 0%
2012-01-23 19:26:40,675 Stage-1 map = 50%,  reduce = 0%
2012-01-23 19:26:42,686 Stage-1 map = 100%,  reduce = 0%
2012-01-23 19:26:47,712 Stage-1 map = 100%,  reduce = 1%
2012-01-23 19:26:49,725 Stage-1 map = 100%,  reduce = 10%
2012-01-23 19:26:50,731 Stage-1 map = 100%,  reduce = 31%
2012-01-23 19:26:51,738 Stage-1 map = 100%,  reduce = 77%
2012-01-23 19:26:52,745 Stage-1 map = 100%,  reduce = 86%
2012-01-23 19:27:44,015 Stage-1 map = 100%,  reduce = 88%
2012-01-23 19:27:45,021 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201201201658_0003
Launching Job 2 out of 2
Number of reduce tasks not specified. Defaulting to jobconf value of: 35
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_201201201658_0004, Tracking URL =
http:// hd01:50030/jobdetails.jsp?jobid=job_201201201658_0004
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=hd01:8021 -kill
job_201201201658_0004
2012-01-23 19:27:50,721 Stage-2 map = 0%,  reduce = 0%
2012-01-23 19:27:52,730 Stage-2 map = 17%,  reduce = 0%
2012-01-23 19:27:53,735 Stage-2 map = 100%,  reduce = 0%
2012-01-23 19:27:59,761 Stage-2 map = 100%,  reduce = 3%
2012-01-23 19:28:00,766 Stage-2 map = 100%,  reduce = 49%
2012-01-23 19:28:01,772 Stage-2 map = 100%,  reduce = 87%
2012-01-23 19:28:56,199 Stage-2 map = 100%,  reduce = 93%
2012-01-23 19:28:57,204 Stage-2 map = 100%,  reduce = 100%
Ended Job = job_201201201658_0004
OK
"大分支店"      7200
"佐賀支店"      8500
"長野支店"      800
"長崎支店"      1000
Time taken: 144.135 seconds
```

Hadoop Hive による 10 億レコードの POS データの処理

動作確認用の 5 レコードの POS データを処理した時と全く同様に、10 億レコードの POS データに対して、Hive を使って Hadoop にロードする時間を計測します。

```
# vi hd_loadcsv.sql
USE hadooptest;
LOAD DATA LOCAL INPATH '/data/item.csv'    INTO TABLE gitem;
LOAD DATA LOCAL INPATH '/data/member.csv' INTO TABLE gmember;
LOAD DATA LOCAL INPATH '/data/shop.csv'    INTO TABLE gshop;
LOAD DATA LOCAL INPATH '/data/data00.csv'  INTO TABLE gdata;

# hive < hd_droptable.sql
# hive < hd_table.sql
# hive < hd_loadcsv.sql
Hive history file=/tmp/root/hive_job_log_root_201201231932_1025946026.txt
hive> USE hadooptest;
OK
Time taken: 1.45 seconds
hive> LOAD DATA LOCAL INPATH '/data/item.csv'    INTO TABLE gitem;
Copying data from file:/data/item.csv
Copying file: file:/data/item.csv
Loading data to table hadooptest.gitem
OK
Time taken: 0.977 seconds
hive> LOAD DATA LOCAL INPATH '/data/member.csv' INTO TABLE gmember;
Copying data from file:/data/member.csv
Copying file: file:/data/member.csv
Loading data to table hadooptest.gmember
OK
Time taken: 0.273 seconds
hive> LOAD DATA LOCAL INPATH '/data/shop.csv'    INTO TABLE gshop;
Copying data from file:/data/shop.csv
Copying file: file:/data/shop.csv
Loading data to table hadooptest.gshop
OK
Time taken: 0.273 seconds
hive> LOAD DATA LOCAL INPATH '/data/data00.csv'  INTO TABLE gdata;
Copying data from file:/data/data00.csv
Copying file: file:/data/data00.csv

Loading data to table hadooptest.gdata
OK
Time taken: 439.464 seconds
hive>
#
```

10 億レコードの POS データの行数をカウントする実行時間を Hadoop Hive で計測します。

```
# vi hd_count_5.sql
USE hadooptest;
SELECT count(*) FROM gdata;
```

```
# hive < hd_count.sql
Hive history file=/tmp/root/hive_job_log_root_201201171658_830212388.txt
hive> USE hadooptest;
OK
...
Ended Job = job_201201201658_0005
OK
1000000000
Time taken: 60.375 seconds
hive>
#
```

10 億レコードの POS データの売上明細テーブルの単一カラムの合計を Hadoop Hive で処理します。

```
# vi hd_sum_price.sql
USE hadooptest;
SELECT sum(price) FROM gdata;

# hive < hd_sum_price.sql
OK
Time taken: 2.817 seconds
hive>  select sum(price) from gdata;
...
Ended Job = job_201201201658_0006
OK
809992357200
Time taken: 475.728 seconds
```

10 億レコードの POS データの支店別売上合計の処理を Hadoop Hive で処理します。

```
# cat hd_total.sql
USE hadooptest;
set mapred.map.tasks=8;
set mapred.reduce.tasks=35;
select name, sum(num*price) total from gshop gp join gdata gd on gp.shopno = gd.shopno
group by name;

# hive < hd_total.sql
...
2012-01-23 19:59:19,718 Stage-2 map = 0%,  reduce = 0%
2012-01-23 19:59:21,725 Stage-2 map = 17%,  reduce = 0%
2012-01-23 19:59:22,729 Stage-2 map = 100%,  reduce = 0%
2012-01-23 19:59:29,753 Stage-2 map = 100%,  reduce = 26%
2012-01-23 19:59:30,757 Stage-2 map = 100%,  reduce = 84%
2012-01-23 19:59:31,762 Stage-2 map = 100%,  reduce = 87%
2012-01-23 20:00:24,960 Stage-2 map = 100%,  reduce = 90%
2012-01-23 20:00:25,964 Stage-2 map = 100%,  reduce = 100%
Ended Job = job_201201201658_0008
OK
"熊本支店"      94845359000
```

"大分支店"	94740054000
"水戸支店"	94828957700
"京都支店"	94790626000
"富山支店"	94784917500
"神戸支店"	94783495000
"福島支店"	94779292600
"佐賀支店"	94808501800
"長野支店"	94796532000
"鹿児島支店"	94786416300
"松江支店"	94741984300
"仙台支店"	94778791000
"宮崎支店"	94776189000
"静岡支店"	94800274400
"宇都宮支店"	94776627700
"新潟支店"	94792672900
"鳥取支店"	94776696900
"千葉支店"	94756630100
"岐阜支店"	94791176300
"山形支店"	94762780300
"秋田支店"	94740499700
"前橋支店"	94771647200
"札幌支店"	94817384300
"奈良支店"	94790527100
"横浜支店"	94790473300
"大津支店"	94781880500
"盛岡支店"	94806840400
"那覇支店"	94759247600
"東京本店"	94761945600
"甲府支店"	94770497100
"和歌山支店"	94778966500
"徳島支店"	94828134600
"高知支店"	94778832500
"福岡支店"	94813980000
"福井支店"	94756154300
"山口支店"	94746770000
"青森支店"	94793225300
"広島支店"	94750516000
"さいたま支店"	94787052000
"名古屋支店"	94787270600
"大阪支店"	94804526400
"津支店"	94818122700
"岡山支店"	94818514600
"金沢支店"	94793197400
"長崎支店"	94788978600
"松山支店"	94772395300
"高松支店"	94807632500
Time taken: 390.437 seconds
hive>
#

MySQL での計測

Hadoop Hive と MySQL の比較を行うため､Hadoop のマスターノード(NameNode)である HP ProLiant SL335s G7 で MySQL サーバーを起動します。

```
# chkconfig mysqld on
# service mysqld start
```

事前に MySQL の root ユーザーで MySQL のデータベースに適切なパスワードでアクセスできるように設定します。今回、MySQL ユーザーは root、パスワードは「password」で設定します。

```
# mysqladmin -uroot password "password"
```

次に、MySQL のバージョンを確認しておきます。

```
# mysqladmin version -uroot -ppassword
mysqladmin  Ver 8.42 Distrib 5.1.52, for redhat-linux-gnu on x86_64
Copyright 2000-2008 MySQL AB, 2008 Sun Microsystems, Inc.
This software comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to modify and redistribute it under the GPL license

Server version          5.1.52
Protocol version        10
Connection              Localhost via UNIX socket
UNIX socket             /var/lib/mysql/mysql.sock
Uptime:                 3 days 3 hours 11 min 23 sec

Threads: 1  Questions: 3658  Slow queries: 0  Opens: 45  Flush tables: 1  Open tables:
38  Queries per second avg: 0.13
```

MySQL にデータベースを作成します。データベース名は hadooptest とします。

```
# mysqladmin -uroot -ppassword create hadooptest
```

MySQL にデータベース hadooptest が作成されているかを確認します。

```
# echo "show databases;" | mysql -uroot -ppassword
Database
information_schema
hadooptest
mysql
...
```

データベース hadooptest にテーブルをロードする SQL 文を記述したファイルを作成します。

```
# vi my_table.sql
USE hadooptest;
CREATE TABLE gdata
(
  transno  BIGINT(12),
  datet    VARCHAR(10) NOT NULL,
  empno    INT(11) NOT NULL,
  shopno   INT(11) NOT NULL,
  regno    INT(11) NOT NULL,
  memberno INT(11) NOT NULL,
  itemno   INT(11) NOT NULL,
  price    INT(11) NOT NULL,
```

```
  num       INT(11) NOT NULL,
  PRIMARY KEY(transno)
);

CREATE TABLE gitem
(
  itemno  INT(11),
  name    VARCHAR(16) NOT NULL,
  price   INT(11) NOT NULL,
  l       VARCHAR(8) NOT NULL,
  m       VARCHAR(8) NOT NULL,
  s       VARCHAR(1) NOT NULL,
  PRIMARY KEY(itemno)
);

CREATE TABLE gmember
(
  memberno  INT(11),
  name      VARCHAR(32) NOT NULL,
  tel       VARCHAR(16) NOT NULL,
  age       INT(11) NOT NULL,
  sex       VARCHAR(1) NOT NULL,
  PRIMARY KEY(memberno)
);

CREATE TABLE gshop
(
  shopno  INT(11),
  name      VARCHAR(32) NOT NULL,
  state     VARCHAR(16) NOT NULL,
  city      VARCHAR(16) NOT NULL,
  PRIMARY KEY(shopno)
);
```

作成したmy_table.sqlファイルをMySQLにロードし、データベースhadooptestにテーブルgdata、gitem、gmember、gshopを作成します。

```
# mysql -uroot -ppassword < my_table.sql
```

データベースhadooptestにテーブルgdata、gitem、gmember、gshopが作成されているかを確認します。

```
# echo "show tables;" |mysql -uroot -ppassword hadooptest
Tables_in_hadooptest
gdata
gitem
gmember
gshop
```

メンテナンス用にテーブル削除用のSQL文を記述したファイルmy_droptable.sqlも用意しておくとよいでしょう。

```
# vi my_droptable.sql
```

```
USE hadooptest;
DROP TABLE gitem;
DROP TABLE gmember;
DROP TABLE gshop;
DROP TABLE gdata;
```

Hadoop Hive のテストで使用した CSV ファイルを利用して計測します。CSV ファイルがあるかを確認します。

```
# cd /data/
# ls -lh *.csv
-rw-r--r-- 1 root root  39G  1月 24 00:33 2012 data00.csv
-rw-r--r-- 1 root root 4.6K  1月 24 00:19 2012 item.csv
-rw-r--r-- 1 root root  42K  1月 24 00:19 2012 member.csv
-rw-r--r-- 1 root root 2.0K  1月 24 00:19 2012 shop.csv
-rw-r--r-- 1 root root  171  1月 24 00:19 2012 small.csv
```

作成した MySQL のテーブルに CSV ファイルのデータをセットする SQL 文が書かれたファイルを用意します。10 億レコードのデータをテストする場合は、最下行の CSV ファイルで data00.csv を指定し、動作確認用の 5 レコードのデータをテストする場合は、small.csv を指定します。

```
# pwd
/data
# vi my_loadsmallcsv.sql
USE hadooptest;
LOAD DATA INFILE '/data/item.csv'   INTO TABLE gitem   FIELDS TERMINATED ¥
BY ',' ENCLOSED BY '"';     ←実際には１行
LOAD DATA INFILE '/data/member.csv' INTO TABLE gmember FIELDS TERMINATED ¥
BY ',' ENCLOSED BY '"';    ←実際には１行
LOAD DATA INFILE '/data/shop.csv'   INTO TABLE gshop   FIELDS TERMINATED ¥
BY ',' ENCLOSED BY '"';    ←実際には１行
LOAD DATA INFILE '/data/small.csv'  INTO TABLE gdata   FIELDS TERMINATED ¥
BY ',' ENCLOSED BY '"';    ←実際には１行
```

上記の my_loadsmallcsv.sql ファイルを使って、MySQL に作成した hadooptest データベースの各テーブル gitem、gmember、gshop、gdata に値をセットします。

```
# mysql -uroot -ppassword < my_loadsmallcsv.sql
#
```

データがロードされているかを確認します。ここではテーブルの上から 3 行のみを表示させる SQL 文が書かれたファイル my_disp3.sql を用意して hive にロードします。

```
# vi my_disp3.sql
USE hadooptest;
SELECT * FROM gitem   limit 3;
SELECT * FROM gmember limit 3;
SELECT * FROM gshop   limit 3;
SELECT * FROM gdata   limit 3;
```

```
# mysql -uroot -ppassword < my_disp3.sql
itemno  name    price   l       m       s
1       品目１  100     L       M       S
2       品目２  100     L       M       S
3       品目３  100     L       M       S
```

```
memberno        name    tel     age     sex
1       富田 夏空       062-812-0278    21      F
2       首藤 照生       083-978-8524    26      M
3       谷村 一徳       089-661-6355    70      M
shopno  name    state   city
1       札幌支店        北海道  札幌市
2       青森支店        青森県  青森市
3       盛岡支店        岩手県  盛岡市
transno datet   empno   shopno  regno   memberno        itemno  price   num
1       2011-07-15      9       41      7       368     41      500     1
2       2011-01-28      2       44      1       1000    118     1200    6
3       2011-04-02      2       20      10      828     37      400     2
4       2011-08-07      2       42      4       395     94      1000    1
5       2011-09-12      7       41      4       157     98      1000    8
```

支店別売上合計の計算を行う SQL 文を記述したファイル my_total.sql を作成し、MySQL で実行します。

```
# pwd
/data
# vi my_total.sql
USE hadooptest;
select name, sum(num*price) total from gshop,gdata where gshop.shopno = ¥
gdata.shopno group by name;  ←実際には１行
```

```
# mysql -uroot -ppassword < my_total.sql
name    total
長野支店        800
長崎支店        1000
大分支店        7200
佐賀支店        8500
```

MySQL による 10 億レコードの POS データの処理

動作確認用の 5 レコードの POS データを処理した時と全く同様に、10 億レコードの POS データに対して、MySQL にロードする時間を計測します。時間の計測には time コマンドを利用することができます。

```
# vi my_loadcsv.sql
USE hadooptest;
LOAD DATA INFILE '/data/item.csv'    INTO TABLE gitem   FIELDS TERMINATED ¥
BY ',' ENCLOSED BY '"'; ←実際には１行
LOAD DATA INFILE '/data/member.csv' INTO TABLE gmember FIELDS TERMINATED ¥
BY ',' ENCLOSED BY '"'; ←実際には１行
LOAD DATA INFILE '/data/shop.csv'    INTO TABLE gshop   FIELDS TERMINATED ¥
BY ',' ENCLOSED BY '"'; ←実際には１行
LOAD DATA INFILE '/data/data00.csv'  INTO TABLE gdata   FIELDS TERMINATED ¥
BY ',' ENCLOSED BY '"'; ←実際には１行
```

```
# mysql -uroot -ppassword < my_droptable.sql
# mysql -uroot -ppassword < my_table.sql
# time mysql -uroot -ppassword < my_loadcsv.sql
```

```
real    89m42.787s
user    0m0.009s
sys     0m0.005s
```

MySQL において 10 億レコードの POS データの行数をカウントする実行時間を計測します。

```
# vi my_count.sql
USE hadooptest;
SELECT count(*) FROM gdata;

# time mysql -uroot -ppassword < my_count.sql
count(*)
1000000000

real    0m0.013s
user    0m0.007s
sys     0m0.006s
```

10 億レコードの POS データの売上明細テーブルの単一カラムの合計を MySQL で処理します。

```
# vi hd_sum_price.sql
USE hadooptest;
SELECT sum(price) FROM gdata;

# time mysql -uroot -ppassword < my_sum_price.sql
sum(price)
809992357200

real    6m40.338s
user    0m0.009s
sys     0m0.005s
```

最後に 10 億レコードの POS データの支店別売上合計の処理を MySQL で処理します。

```
# pwd
/data
# cat my_total.sql
USE hadooptest;
select name, sum(num*price) total from gshop,gdata where gshop.shopno = ¥
gdata.shopno group by name;  ←実際には 1 行

# time mysql -uroot -ppassword < my_total.sql
# time mysql -uroot -ppassword < my_total.sql
name    total
さいたま支店    94787052000
熊本支店        94845359000
甲府支店        94770497100
盛岡支店        94806840400
神戸支店        94783495000
福岡支店        94813980000
福島支店        94779292600
福井支店        94756154300
秋田支店        94740499700
```

那覇支店　　94759247600
金沢支店　　94793197400
長野支店　　94796532000
長崎支店　　94788978600
青森支店　　94793225300
静岡支店　　94800274400
高知支店　　94778832500
高松支店　　94807632500
鳥取支店　　94776696900
鹿児島支店　　94786416300
前橋支店　　94771647200
千葉支店　　94756630100
名古屋支店　　94787270600
和歌山支店　　94778966500
大阪支店　　94804526400
大分支店　　94740054000
大津支店　　94781880500
奈良支店　　94790527100
宇都宮支店　　94776627700
宮崎支店　　94776189000
富山支店　　94784917500
山口支店　　94746770000
山形支店　　94762780300
岐阜支店　　94791176300
岡山支店　　94818514600
広島支店　　94750516000
徳島支店　　94828134600
新潟支店　　94792672900
札幌支店　　94817384300
東京本店　　94761945600
松山支店　　94772395300
松江支店　　94741984300
横浜支店　　94790473300
水戸支店　　94828957700
津支店　94818122700
京都支店　　94790626000
仙台支店　　94778791000
佐賀支店　　94808501800

real　61m58.305s
user　0m0.008s
sys　0m0.006s

以上