探索的財務ビッグデータ解析

—PG-Strom によるデータラングリングの並列化—

地　道　正　行

要　旨

本稿では，データベース Orbis から抽出された世界の2,400万社を超える企業（一般事業会社）に関する財務データを前処理したものを，**PG-Strom** を利用することによって並列処理し，データラングリングを行う時間を短縮することを試みる．また，探索的データ解析の端緒として，データの要約と可視化も行い，さらに，この工程を自動実行することによって，再現可能性を確保する方法についても述べる．

キーワード：探索的財務ビッグデータ解析（Exploratory Financial Big Data Analysis），前処理（Preprocessing），データラングリング（Data Wrangling），並列化（Parallelization），再現可能性（Reproducibility）

I　はじめに

地道（2020）では，粗データ（raw data）を「読み込めるファイル形式」に変換する工程を「前処理」（preprocessing）と呼び，それを GNU parallel（Tange, 2018）を用いて並列化することによって，処理速度（velosity）を改善した[1)]．本稿では，このように処理されたデータファイルをデータ解析環境に読み込み，分析・解析できるオブジェクトに変換する工程を「データラングリング」（data wrangling）と呼び（Wickham and Grolemund, 2016 参

1）　GNU parallel を利用した並列処理については，Janssens（2014）の第 8 章が参考になる．

照)，この工程を並列化することによって高速化することを試みる（図１も参照のこと）.

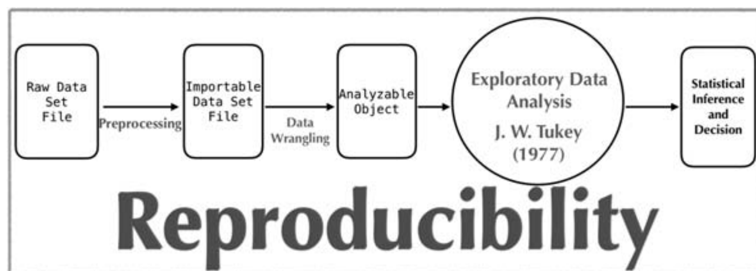


図１　探索的財務ビッグデータ解析の全工程（地道，2020，図１を再掲）

「データサイエンス」や「ビッグデータ」に関する文献等における経験則として，前処理とデータラングリングの工程は，データを処理・分析・解析する全工程の50％から90％を占めるともいわれることからも（cf. Patil (2012), p. 18)，この試みは重要な意味を持つ.

具体的には，財務関連の情報を含むデータベースから抽出された規模の大きい粗データファイルを，前処理を行うことによって CSV ファイル[2] に変換したもの（地道，2020参照）を，PostgreSQL[3] でデータベース化する．さらに **PG-Strom**[4] を利用することによって，GPGPU[5] で並列処理し，データラングリングを高速化することによって，処理速度を向上させることを試みる.

2） CSV（Comma Sepelated Values）ファイルとは，項目（カラム）間がコンマ区切りのテキスト形式のファイルである.

3） PostgreSQL（https://www.postgresql.org/）は，リレーショナル・データベース管理システム（Relational Database Management System: RDBMS）の一つである．本研究では，バージョン10.10を利用している.

4） **PG-Strom**（https://heterodb.github.io/pg-strom/ja/）は，海外浩平氏によってオープンソースとして開発されている PostgreSQL の拡張モジュールであり，GPGPU を使うことによって，高速に PostgreSQL サーバからデータを抽出する仕組みを提供する．海外（2019-a, b）を参照されたい.

5） GPGPU とは，General-Purpose computing on Graphics Processing Units の略語であり，画像処理を高速に実行する GPU（Graphics Processing Unit）の機能を，画像処理以外の用途に転用することである（IT 用語辞典 http://e-words.jp/w/GPGPU.html 参照）.

本稿の構成は次のようなものである．まず，Ⅱ節では，本稿で利用するデータベースとデータセット，そして前処理されたデータセットファイルについて述べ，Ⅲ節では，従来から行ってきた，Apache Spark™[6]（以下，Spark と略）を用いたデータラングリングについて述べる．次に，Ⅳ節では **PG-Strom** 環境を利用してデータラングリングを行い，Spark 環境を用いた場合と比較する．さらに，この応用として，Ⅴ節では，データラングリングの結果として抽出されたデータオブジェクトに対する探索的データ解析の端緒として，データの要約と可視化を行う．最後に，Ⅵ節では総括を行う．なお，本稿で利用した計算機環境についての情報を付録に与えている．

Ⅱ　データベースとデータセット

本研究では，Bureau van Dijk[7]（ビューロー・ヴァン・ダイク）社（以下 BvD と略）のデータベース Orbis（オービス）を利用する．このデータベースには，世界の全企業（データ抽出時点で約2.4億社以上を収録）の情報が国際比較可能な統一のフォームで収録されている．このデータベースから，データセットとして，主要財務情報（売上高，資産合計など）を最長10年分抽出した以下のようなものを利用する：

（1）連結（consolidated）財務諸表を優先的に抽出した24,014,352社
（2）非連結（un-consolidated）財務諸表を優先的に抽出した24,012,807社

データセットのファイルは，１個のサイズが５GB 程度の25個の TSV ファイル[8]（２セット）からなり，表１，表２には，それぞれ，データセットの仕様とサイズを与えている．

地道（2020）では，主に連結優先で抽出したデータセット DS-Orbis-C-2018 の前処理の工程を並列化によって効率化することが議論されている．

6） https://spark.apache.org/
7） BvD Web Page: https://www.bvdinfo.com/en-gb/
8） TSV（Tab Sepelated Values）ファイルとは，項目（カラム）間がタブ区切りのテキスト形式のファイルである．

表１　データセット：仕様

データセット名	抽出年度	データベース	上場情報	抽出主体	抽出期間	抽出指標数
DS-Orbis-C-2018	2018	Orbis	上場・非上場	連結優先	10年	82
DS-Orbis-U-2018	2018	Orbis	上場・非上場	非連結優先	10年	82

表２　粗データセット：サイズ

データセット名	社数	総行数	粗データファイル	トータルサイズ
DS-Orbis-C-2018	24,014,352社	264,157,872行	`01_orbis_2018.asc,..., 025_orbis_2018.asc`	約 127 GB
DS-Orbis-U-2018	24,012,807社	264,140,888行	`01_orbis_2018_LU.asc,..., 025_orbis_2018_LU.asc`	約 125 GB

なお，処理後のファイル（CSV 形式）のサイズは，DS-Orbis-C-2018（連結優先），DS-Orbis-C-2018（連結優先）のそれぞれの場合に対して表３のようなものである．

表３　前処理後のデータセット：サイズ

データセット名	社数	総行数	CSV ファイル	サイズ
DS-Orbis-C-2018	24,014,352社	240,143,521行	`firmfinBC2018.csv`	約 124.5 GB
DS-Orbis-U-2018	24,012,807社	240,128,071行	`firmfinBU2018.csv`	約 124.8 GB

Ⅲ　Spark によるデータラングリング

Spark は，高速かつ汎用的なクラスタ・コンピューティング・システムの一つであり，本稿で利用するデータ解析環境 R と連携して利用するためのパッケージも提供されている．地道（2018-a）では，BvD のデータベース Osiris から世界157カ国の全上場企業を対象に抽出されたデータセットを前処理したもの（約 1.3 GB の CSV ファイル）を Spark と R，そして SparkR，sparklyr パッケージを利用してラングリングを行ったが，SparkR パッケージによって利用可能となる Spark DataFrame オブジェクト形式は，ここで扱うような 100 GB を超える規模のデータセットにも対応している．

ここでは，東京大学情報基盤センターの専有利用型リアルタイムデータ解析ノード（FENNEL）上で，Spark 環境を利用して，DS-Orbis-C-2018（連結優先）のデータセットファイル `firmfinBC2018.csv` を R へ読み込み，

オブジェクトへ変換（データラングリング）することを試みる．なお，計算機環境については付録を参照されたい．

Spark の環境設定として，R 上で以下のように入力することによって SparkR パッケージを利用した：

スクリプト 1　SparkR パッケージを利用するための R スクリプト

```
1 > Sys.setenv(SPARK_HOME = "/home/masa/spark/spark-2.2.0-bin-hadoop2.7")
2 > Sys.setenv(JAVA_HOME = "/usr/lib/jvm/java-8-oracle")
3 > library(SparkR, lib.loc = c(file.path(Sys.getenv("SPARK_HOME"), "R" , "lib
    ","")))
4 > sparkR.session(master = "local[*]", sparkConfig = list(spark.driver.
    memory = "12g"), spark.debug.maxToStringFields = "200")
```

SparkR パッケージに付属する関数 read.df を以下のように利用することによって，CSV ファイルを R のオブジェクト（Spark DataFrame オブジェクト）として読み込むことができる．

スクリプト 2　**read.df** によるデータの読み込み

```
1 > firmfinBC.sdf <- read.df("../CSV/firmfinBC2018.csv", sourc = "csv",
    header = TRUE, inferSchema = "true", na.strings = "NA")
```

この入力の工程において，CPU の稼働率を htop[9] コマンドによってモニタリングした結果を図 2 に与える．この図から，全ての CPU コアがほぼ100％で稼働していることがわかり，Spark はデフォルトで並列処理を行っていることがわかる．

読み込まれた Spark DataFrame オブジェクト（firmfinBC.sdf）は，そのままでは可視化や統計モデリングを行うことに適していないため，SparkR パッケージに付属する filter 関数と select 関数を用いて適当な条件のものとで行と列の選択を行った後，collect 関数で R で標準的に扱われるオブジェクト形式（R の data.frame オブジェクト）に変換する．その際，データ処理の流れ[10] の視認性を向上させるために，パイプ演算子

9） http://hisham.hm/htop/
10）パイプ演算子 %>% を使って，データの処理をつなげることは，「データパイプライン」（data pipeline）と呼ばれることがある．

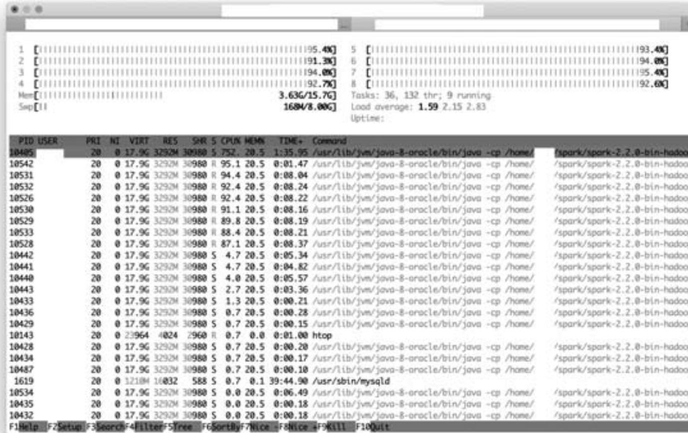


図２　**htop** コマンドによる CPU の稼働率のモニタリング：図の上部のプログレスバーが CPU コアの稼働状態をあらわしており，８個の CPU コアが全てほぼ100％で稼働していることが読み取れる

`%>%` を利用している．なお，これまでに行ってきたデータ解析のためのスクリプトとの整合性を保つために，関数 `colnames` を用いて列名を変更している．

スクリプト３　オブジェクト変換と列名の変更

```
1 > library (magrittr)
2 > firmfinBC2015 <- firmfinBC.sdf %>%
3       filter(firmfinBC.sdf$year == "2015" & firmfinBC.sdf$sales > 0 &
           firmfinBC.sdf$employees > 0 & firmfinBC.sdf$assets_total > 0 &
           firmfinBC.sdf$month == 12) %>%
4       select(firmfinBC.sdf$firmID, firmfinBC.sdf$country, firmfinBC.sdf$
           cons, firmfinBC.sdf$listed, firmfinBC.sdf$exchange, firmfinBC.sdf$
           InfoProv, firmfinBC.sdf$sales, firmfinBC.sdf$employees, firmfinBC.
           sdf$assets_total) %>%
5       collect()
6  > colnames(firmfinBC2015) <- c("firmID", "country", "cons", "listed", "
     exchange", "infoprov", "sales", "employees", "assets.total")
```

ここでは，行選択の基準として，決算年を2015年（`firmfinBC.sdf$year`

== "2015")，売上高，従業員数，資産合計を正値（firmfinBC.sdf$sales > 0, firmfinBC.sdf$employees > 0, firmfinBC.sdf$assets_total > 0)，さらに決算月数を12カ月のものに限定（firmfinBC.sdf$month ==12）している．また，以下の列を抽出対象とした：

表 4 Orbis データセットファイルからの抽出対象列（1）

列名	説明
firmID	企業名と BvD 企業 ID の結合
country	所在国
cons	連結情報
listed	上場情報
exchange	主取引所
InfoProv	情報提供元
sales	売上高
employees	従業員数
assets_total	資産合計

以上のデータラングリングの工程をイメージ化したものが図 3 である．

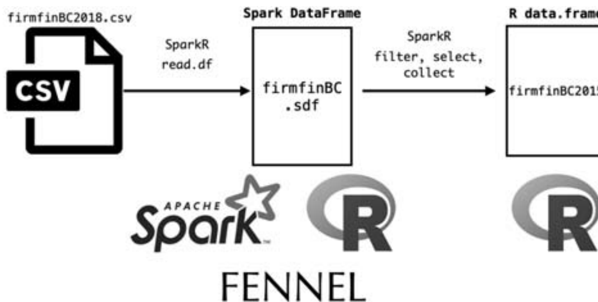


図 3 FENNEL 上での Spark，R，SparkR パッケージによるデータラングリング

以上のデータラングリングの工程を計測した結果として，約30分（1765秒）かかることがわかった[11)]．なお，複数回の実行において数秒単位のばらつきはあるものの，一回のラングリングに，このような時間がかかるという

結果は，探索的データ解析を行う上で，インタラクティブ性が損なわれることを表している．以上の考察から，この工程の処理速度を向上させることが本研究の目的である．

Ⅳ　PG-Strom によるデータラングリング

本節では，**PG-Strom** 環境を利用することによって，ラングリングの高速化を検討する．

1　環境設定

PG-Strom 環境の利用にあたっては，以下の設定が必要となる：

（PG-1）PostgreSQL データベースの構築
（PG-2）**PG-Strom** の設定

ここでは，設定（PG-1）に関する簡単な説明を行い，設定（PG-2）に関する詳細は割愛する[12]．PostgreSQL データベースの構築にあたっては，一旦，サーバー上に PostgreSQL データベース（`jhpcn`）を作成後，2 種類のテーブル（`orbis2018c`，`orbis2018u`）を作成し，表 3 におけるデータセットファイル `firmfinBC2018.csv`，`firmfinBU2018.csv` から，それぞれ，テーブル `orbis2018c`，`orbis2018u` にコピーする．これらの工程の流れを図 4 に与える．

図 4 について簡単に補足すると，まず，データセットの CSV ファイル `firmfinBC2018.csv`，`firmfinBU2018.csv` を，それぞれ，ファイル `tmpc.csv`，`tmpu.csv` へ一旦変換しており，この段階での処理は，

11）`tictoc` パッケージに付属する `tic`，`toc` 関数を用いて計測を行った
12）本研究で利用する **PG-Strom** 環境は，JHPCN プロジェクト（`https://jhpcn-kyoten.itc.u-tokyo.ac.jp/abstract/jh191002-NWJ`）の一環として，実験的に期間限定（2019年11月15日－12月 6 日）で用意された環境を利用した．なお，**PG-Strom** のインストールについては，`https://heterodb.github.io/pg-strom/ja/install/` を参照されたい．

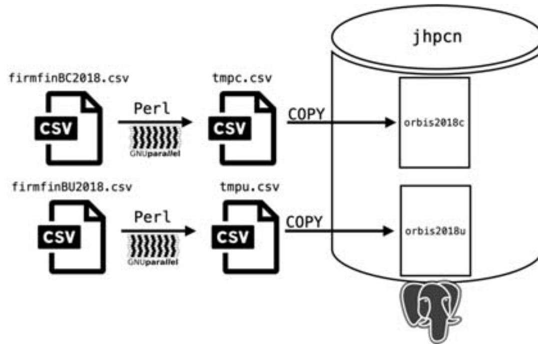


図 4　**PostgreSQL データベース jhpcn のテーブル orbis2018c, orbis2018u へのデータのコピー**

PostgreSQL データベースへデータファイルをロード（コピー）を行うために文字列の置換をおこなっている[13]．次に，これらの変換された CSV ファイル tmpc.csv, tmpu.csv を PostgreSQL の COPY コマンドを利用してテーブル orbis2018c, orbis2018u へコピーしている．

2　PostgreSQL 環境のもとでのデータラングリング

通常，R から RDBMS へ接続するためには，専用の R パッケージを利用する．たとえば，R から PostgreSQL のデータベースへ接続するためには，DBI パッケージと RPostgreSQL パッケージを利用することによって可能となる（図 5 参照）．

RPostgreSQL パッケージを利用し，PostgreSQL データベースに接続するための設定は，以下のようなものである：

スクリプト 4　RPostgreSQL パッケージを利用するための設定

```
1 > library(RPostgreSQL)
2 > drv.PG <- dbDriver("PostgreSQL")
3 > con <- dbConnect(drv=drv.PG, host = "xxxx", port = 5432, user = "yyyy",
     password = "********", dbname = "jhpcn")
```

13）実際には，データファイルに存在する欠測値を表す文字列（NA）を置換するための処理を Perl と GNU parallel を用いて行っている．

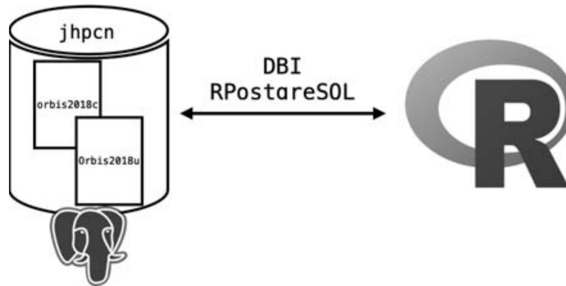


図５　DBI，RPostgreSQL パッケージによる **Postgre SQL** と R の接続

ここで，`library(RPostgreSQL)` によって，RPostgreSQL パッケージを読み込んでいる．なお，このパッケージの読み込みと同時に読み込まれる DBI パッケージには，`dbConnect` 関数が収められており，この関数を使って，データベースドライバを指定（`dbDriver("PostgreSQL")`）し，その結果をオブジェクト `drv.PG` に付置している．さらに，DBI パッケージに付属する `dbConnect` 関数に引数として，ドライバ名（`drv=drv.PG`），ホスト名（RDBMS サーバ名）（`host="xxxx"`），接続ポート番号（`port=5432`），ユーザ名（`user= "yyyy"`），パスワード（`password="********"`），データベース名（`dbname="jhpcn"`）を与え，その結果を `con` オブジェクトに付値している．この後は，データベースとの通信は `con` オブジェクトを介して行う[14]．

以上の設定のもとで，PostgreSQL データベース `jhpcn` から R を利用してデータラングリングを行う（図６参照）．

その際，Spark を利用して行った一連のデータラングリング（スクリプト２，３）と同等の条件で行うために，スクリプト５を利用する．

14）セキュリティの関係上，ここでは，ホストアドレス，ユーザ名，パスワードは省略している．

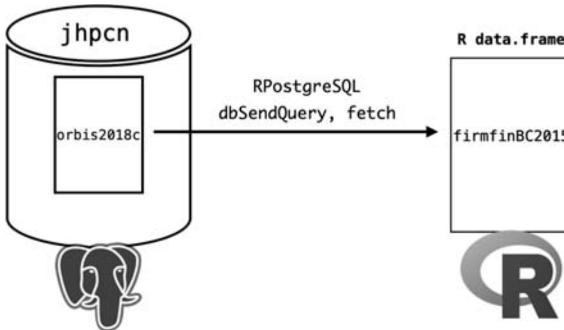


図 6　RPostgreSQL による **PostgreSQL** データベースからのデータラングリング

スクリプト 5　RPostgreSQL パッケージによるデータラングリング

```
1 > sql <- "select␣firmID,␣country,␣cons,␣listed,␣exchange,␣InfoProv,␣sales,␣
      employees,␣assets_total␣from␣orbis2018c␣␣where␣year␣=␣2015␣and␣sales␣>␣0
      ␣and␣employees␣>␣0␣and␣assets_total␣>␣0␣and␣month␣=␣12"
2 > rs <- dbSendQuery (con, sql)
3 > firmfinBC2015 <- fetch(rs, n = -1)
```

スクリプト 5 の 1 行目でデータ抽出を行うための SQL スクリプトを定義したものを sql に付値しており，この情報とスクリプト 4 の 3 行目で定義されているデータベースとの接続情報 con を関数 dbSendQuery に引数として与えることによって，SQL クエリーをデータベースエンジンにサブミットするための情報を定義し，rs に付置している．なお，この段階では，データ抽出はまだ行われておらず，最終的には 3 行目で関数 fetch の引数に与えることによって，データの抽出を行っている．ただし，引数 n=-1 は先頭行を読み飛ばすための指定である．

このラングリング時間の計測結果は，約640秒（10分40秒）であり，この結果は Spark を用いた場合（約30分）からは改善（約 3 分の 1 に短縮）されている．しかしながら，1 回のラングリングに，これだけの時間がかかるということは，対話型の処理を基本とする探索的データ解析にとって快適な環境とはいえない．この問題に対して，**PG-Strom** 環境によって改善するこ

とを試みる．

3　PG-Strom 環境のもとでのデータラングリング

Spark を R から利用するときは，スクリプト 1 で与えたように，Sys.setenv 関数を利用した環境変数の設定や，SparkR パッケージのロード，さらに sparkR.session 関数を使って Spark セッションをスタートさせる等の専用の環境設定を行うことが必要となる．一方，**PG-Strom** が適切に設定された環境のもとで，R を用いてデータラングリングを行うための（これ以上の）特別な設定は必要ない．この理由は，**PG-Strom** が PostgreSQL に対するプラグインとして実装され，ハードウェア（GPGPU）に強く依存したシステムであることにある[15]．つまり，**PG-Strom** 環境のもとでデータラングリングを行うためには，スクリプト 4，5 をそのまま実行すればよい（図 7 参照）．

スクリプト 4，5 を実行する工程で，見かけ上は PostgreSQL の環境下で実行した場合と見分けがつかないが，GPGPU の稼働率を nvtop[16] コマンド

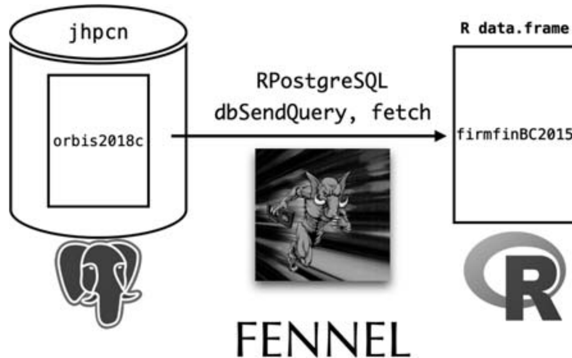


図 7　**PG-Strom** 環境のもとでの RPostgreSQL による **PostgreSQL** データベース **jhpcn** のテーブル **orbis2018c** からのデータラングリング

15）http://www.elsa-jp.co.jp/products/products-top/heterodb/pg-strom/pg-stoem/hetero-db-pg-strom/ 参照．

によってモニタリングした結果（図８）から，GPU 稼働率が36％であることがわかり，**PG-Strom** が GPU コアを並列的に利用して処理を行っていることがわかる[17]．

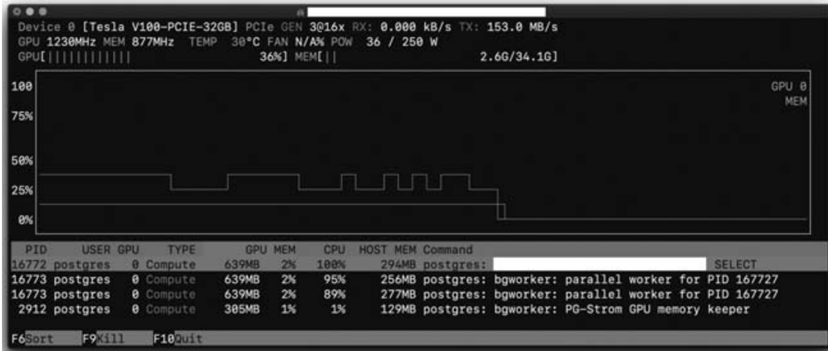


図８　**nvtop** コマンドによる **GPGPU** の稼働率のモニタリング：図の上部のプログレスバーが **GPU** コアの稼働状態をあらわしており，**GPU** 稼働率が36％であることが読み取れる．また，**postgres** のプロセスが並列的に働いている（**parallel worker**）ことも同様にわかる．

このラングリング時間の計測結果は，約86秒（１分26秒）であり，この結果は Spark を用いた場合（約30分）から飛躍的な改善（約20分の１に短縮）がなされており，PostgreSQL 環境のみを用いた場合（10分40秒）からも十分に改善（９分の１に短縮）されていることがわかる．

このように１回のラングリングに対する計測だけでは不十分と思われるため，100回ラングリングを繰り返し，計測した結果を可視化することを考える[18]．

16）https://github.com/Syllo/nvtop/
17）今回利用した NVIDIA® Tesla® Volta（V100）（PCIe），32 GB のコア（CUDA コア）数は5120である．
18）データラングリングの時間を計測することに関するシステム間の差異を比較するために公平性を期するためには，本来であれば，Spark 環境と PostgreSQL のネイティブ環境での計測も複数回行って比較する必要があろうが，**PG-Strom** 環境以外のシステムでは，１回の計測にかかる時間が30分または10分程度かかることから割愛している．

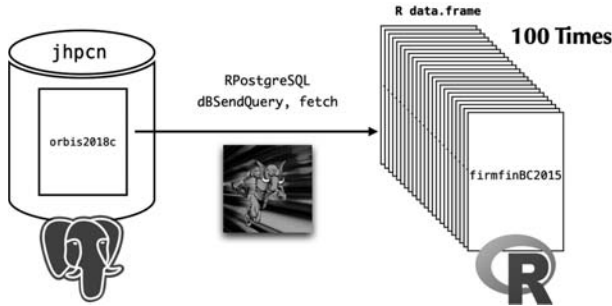


図 9　**PG-Strom** 環境のもとでの RPostgreSQL による **PostgreSQL** データベースからのデータラングリングを100回繰り返した場合

このラングリングを実行するための関数 bm を定義し，この関数を100回実行した結果を計測し，CSV ファイルに書き出す（ベンチマークする）ためのスクリプトファイル bm.R（スクリプト 6 参照）を作成した．

スクリプト 6　ベンチマーク用 R スクリプトファイル **bm.R**

```
 1  # Define benchmark function
 2  bm <- function(){
 3        require(RPostgreSQL)
 4        con <- dbConnect(drv=drv.PG, host="xxxx", port = 5432, user = "yyyy",
              password = "********", dbname = "jhpcn")
 5        sql <- "select␣firmID,␣country,␣cons,␣listed,␣exchange,␣InfoProv,␣
             sales,␣employees,␣assets_total␣from␣orbis2018c␣where␣year␣=␣2015␣
             and␣sales␣>␣0␣and␣employees␣>␣0␣and␣assets_total␣>␣0␣and␣month␣=␣
             12"
 6        rs <- dbSendQuery(con, sql)
 7        firmfinBC2015 <- fetch(rs, n = -1)
 8        dbDisconnect(con)
 9  }
10  # Excute benchmark
11  queue <- matrix(nrow = 100, ncol = 3);
12  for (i in 1:100){
13        t1 <- proc.time(); bm(); t2 <- proc.time() - t1
14        queue[i,1] <- t2[1]; queue[i,2] <- t2[2]; queue[i,3] <- t2[3]
15  }
16  # Save R object to CSV file
17  write.csv(queue, "benchmark-gpu2-100.csv")
```

スクリプト 6 における 2 行目から 9 行目で定義されている関数 bm は，スクリプト 4 による RPostgreSQL を利用するための設定と，スクリプト 5 によ

るラングリングを関数化したものであり，この関数を実行することによって，図９におけるラングリングを１回実行するものである．また，11行目から14行目で，実際に100回のラングリングを実行し，各実行時間の計測結果をオブジェクト `queue` に格納している．さらに，16行目で，オブジェクト `queue` を CSV ファイル `benchmark-gpu2-100.csv` に出力している．

以上の処理を，手動ではなく `make` コマンドで自動実行するために，`Makefile`（スクリプト７）を作成し，ターゲット `bm` を用意した．

スクリプト７　make ファイル Makefile：ターゲット bm

```
bm:
	date > start -bm.txt
	Rscript bm.R
	date > end -bm.txt
```

スクリプト７の１行目から４行目で定義されているターゲット `bm` を以下のようにターミナルで実行することによって，ベンチマークの結果が CSV ファイル `benchmark-gpu2-100.csv` に出力される（図10も参照）．

ターゲット bm の実行

```
$ make bm
```

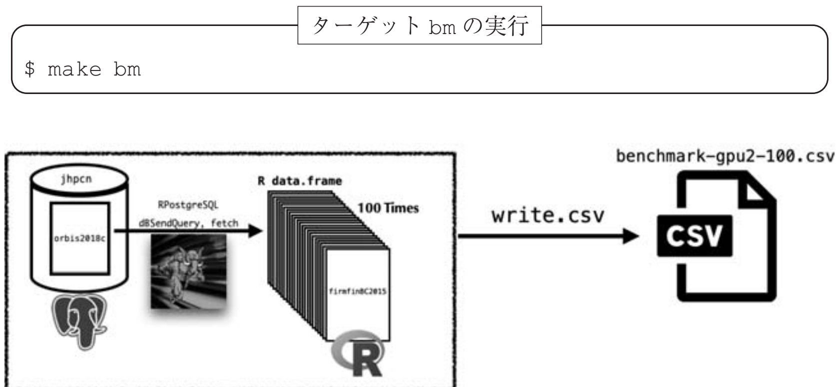


図10　ターゲット **bm** に対する **make** の実行によって実行される処理

以上の処理は，FENNEL 環境のもとで実行しており，この工程にかかった時間の計測結果は，以下のようなものである：

ベンチマーク実行の計測時間

```
$ cat start-bm.txt
2019年 11月 22日 金曜日 11:37:42 JST
$ cat end-bm.txt
2019年 11月 22日 金曜日 13:42:02 JST
```

ここでは，ターゲット bm の実行開始時と終了時に出力される時間情報が記録されたファイル（スクリプト７参照）を表示しており，この時間の差分をとることによってベンチマークの実行時間がわかる．この結果から，２時間４分20秒かかったことがわかる．

次に，この結果を可視化するためには，ローカル環境で行った方が便利なため sftp コマンドを使って CSV ファイルを転送後，実行時間のデータの可視化を行った（図11参照）．

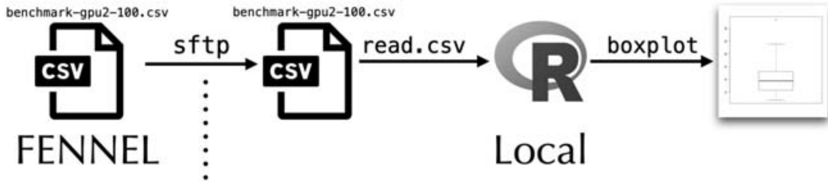


図11　ベンチマークの実行時間の CSV ファイルのローカル環境への転送と，R による可視化

ここで，ベンチマーク（100回のデータラングリング）の実行時間の計測結果をボックスプロットによって可視化しており，実際にはローカルの R 環境のもとで以下のように入力して実行している：

ベンチマーク結果の読み込みと可視化

```
> bm100 <- read.csv("benchmark-gpu2-100.csv")
> boxplot(bm100$V3, ylim = c(70, 90))
```

実際のボックスプロットは，図12で与えられる．

この結果から，全ての処理は90秒以内であり，中央値が76秒程度であるこ

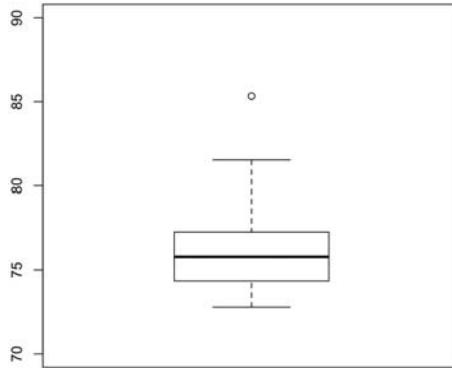


図12　ベンチマーク実行時間の計測結果のボックスプロット

とから，**PG-Strom** 環境は対話型の処理を基本とする探索的データ解析を実行可能なものといえる．次節では，これまでに Spark 環境では難しかったラングリングによる規模の大きなデータの探索的データ解析（要約，可視化）を実行する．なお，ここで考察したベンチマークの全工程については，図13を参照されたい．

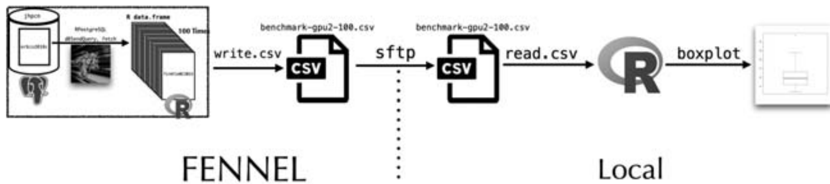


図13　ベンチマークの全工程のイメージ

V　PG-Strom 環境のもとでの探索的データ解析

1　総資産利益率と実効税率の散布図の描画：2015年の場合

ここでは，データベース Orbis から非連結財務諸表を優先に抽出したデータセット DS-Orbis-U-2018 にもとづいて作成された PostgreSQL のテーブル `orbis2018u`（図 4 参照）から，以下のような R スクリプトを実行するこ

とによってラングリングを行った（図14も参照）．なお，一見すると特別な設定は行っていないよう見えるが，**PG-Strom** 環境で実行している．

スクリプト８　PG-Strom 環境でのテーブル orbis2018u からのデータラングリング

```
1 > sql2 <- "select␣firmID,␣year,␣country,␣month,␣cons,␣listed,␣exchange,␣
      InfoProv,␣interest_paid,␣costs_employees,␣tax,␣PL_after_tax,␣PL_before_
      tax,␣assets_total␣from␣orbis2018u␣where␣year␣=␣2015␣and␣month␣=␣12␣and␣(
      cons␣=␣'U1'␣or␣cons␣=␣'U2')"
2 > rs2 <- dbSendQuery(con, sql2)
3 > firmfinBU2015U <- fetch(rs2, n = -1)
```

ここでは，行選択の基準として，決算年を2015年（`year = 2015`）とし，決算月数を12カ月（`month = 12`），さらに非連結のものに限定（`(cons = 'U1' or cons = 'U2')`）している．また，列として表５のようなものを抽出対象とした：

表５　Orbis データセットファイルからの抽出対象列（2）

列名	説明
firmID	企業名と BvD 企業 ID の結合
country	所在国
cons	連結情報
listed	上場情報
exchange	主取引所
InfoProv	情報提供元
interest_paid	支払利息
costs_employees	人件費
tax	税金
PL_after_tax	税引後利益
PL_before_tax	税引前利益
assets_total	資産合計

なお，このラングリングに要した時間は約３分[19]であり，この程度の時間であれば複数年にわたってラングリングを行うことも可能である．

次に，このラングリングによって抽出されたオブジェクト `firmfinBU`

19）正確には176.755秒であり，`tictoc` パッケージの `tic`，`toc` 関数を用いて測定した．

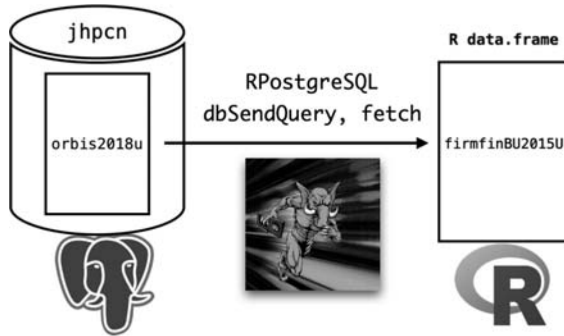


図14　**PG-Strom** 環境のもとでの RPostgreSQL による **PostgreSQL** データベース **jhpcn** のテーブル **orbis2018u** からのデータラングリング

2015U を使って要約と可視化を行う．まず，ラングリングを実行し，その結果のオブジェクト firmfinBU2015U を RDATA ファイル[20] に保存するためのスクリプトファイル DataWranglingOrbis2015u.R を作成した（スクリプト 9 参照）．

スクリプト 9　データラングリングのための R スクリプトファイル **DataWranglingOrbis2015u.R**

```
1 # Data wrangling from table orbis2018u: 2015
2 sql2 <- "select␣firmID,␣year,␣country,␣month,␣cons,␣listed,␣exchange,␣
     InfoProv,␣interest_paid,␣costs_employees,␣tax,␣PL_after_tax,␣PL_before_
     tax,␣assets_total␣from␣orbis2018u␣where␣year␣=␣2015␣and␣month␣=␣12␣and␣(
     cons␣=␣'U1'␣or␣cons␣=␣'U2')"
3 rs2 <- dbSendQuery(con, sql2)
4 firmfinBU2015U <- fetch(rs2, n = -1)
5 # Save R objects
6 save.image("PG-Storm.RData")
```

次に，このスクリプトを自動実行するためのターゲット DW-2015-u を Makefile に以下のように記述した：

20）RDATA ファイルは，R の作業空間（workspace）を保存するためのバイナリー形式のファイルであり，拡張子は，.RData で表される．Windows，macOS，Linux などの各種 OS 間でバイナリー互換である．

スクリプト10 **make** ファイル **Makefile**: ターゲット **DW-2015-u**

```
1 DW-2015-u:
2       date > start-DW-2015-u.txt
3       Rscript DataWranglingOrbis2015u.R
4       date > end-DW-2015-u.txt
```

スクリプト10の1行目から4行目で定義されているターゲット DW-2015-u を以下のようにターミナルで実行することによって，ラングリングの結果が RData ファイル PG-Storm.RData に出力される（図15も参照）.

ターゲット DW-2015-u の実行

```
$ make DW-2015-u
```

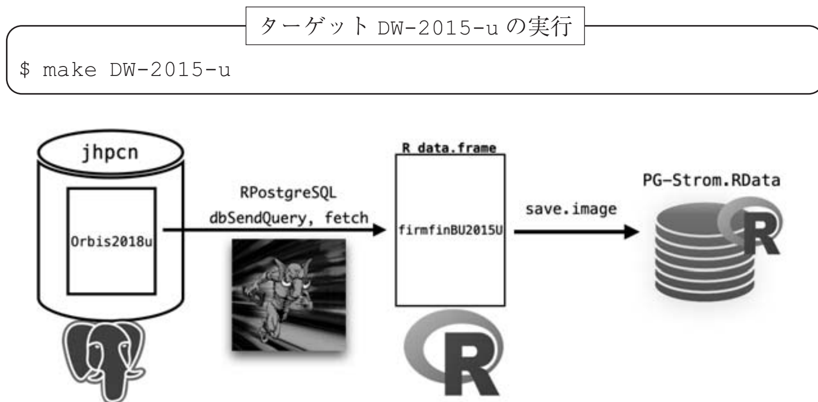


図15 ターゲット **DW-2015-u** に対する **make** の実行

以上の処理は，FENNEL 環境のもとで実行しており，この結果を可視化するためには，ローカル環境で行った方が便利なため sftp コマンドを使って RDATA ファイルを転送後，要約と可視化を行った（図16参照）.

ここで，RDATA ファイル PG-Storm.RData は以下のように読み込んでいる.

RDATA ファイル PG-Storm.RData の読み込み

```
> load ("PG-Strom.RData")
```

また，オブジェクト firmfinBU2015U の要約は以下のような入力によって行った.

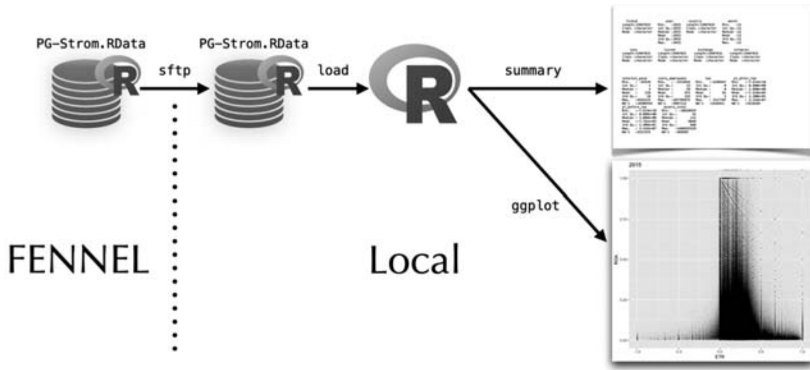


図16　RDATA ファイルのローカル環境への転送と，R による要約と可視化

オブジェクト firmfinBU2015U の要約

```
> library(dplyr)
> firmfinBU2015U %>% summary()
    firmid                year         country               month
 Length:13687832    Min.   :2015   Length:13687832     Min.   :12
 Class :character   1st Qu.:2015   Class :character    1st Qu.:12
 Mode  :character   Median :2015   Mode  :character    Median :12
                    Mean   :2015                       Mean   :12
                    3rd Qu.:2015                       3rd Qu.:12
                    Max.   :2015                       Max.   :12
    cons               listed            exchange            infoprov
 Length:13687832    Length:13687832    Length:13687832    Length:13687832
 Class :character   Class :character   Class :character   Class :character
 Mode  :character   Mode  :character   Mode  :character   Mode  :character
 interest_paid       costs_employees         tax              pl_after_tax
 Min.   :   -45594   Min.   : -1414656   Min.   :-2288447   Min.   :-7.614e+10
 1st Qu.:        1   1st Qu.: 13         1st Qu.:       0   1st Qu.: 0.000e+00
 Median :        4   Median : 59         Median :       0   Median : 2.000e+00
 Mean   :      239   Mean   : 975        Mean   :      81   Mean   :-7.918e+03
 3rd Qu.:       18   3rd Qu.: 233        3rd Qu.:       5   3rd Qu.: 2.300e+01
 Max.   : 4593225    Max.   :207705875   Max.   : 5217707   Max.   : 3.316e+07
 NA's   :10306934   NA's   :9097122     NA's   :5428452    NA's   :4418266
 pl_before_tax assets_total
 Min.   :-7.614e+10     Min.   : -10628929
 1st Qu.: 0.000e+00     1st Qu.:        16
 Median : 3.000e+00     Median :       122
 Mean   :-7.762e+03     Mean   :      8040
 3rd Qu.: 2.900e+01     3rd Qu.:       680
 Max.   : 3.316e+07     Max.   :1468925328
 NA's   :4321555        NA's   :405095
```

ここで，dplyr パッケージを読み込むことによって利用可能となるパイプ演算子 %>% を使って，データフレーム firmfinBU2015U を要約するための関数 summary に引き渡している．この結果から，欠測値（Not Available: NA）が含まれていることがわかる．

次に，可視化を行うためにまずは以下のように要約を行っている．

スクリプト11 企業の ROA と ETR を計算するための R スクリプト

```
1 > firmfin.ROA.ETR.2015.firm.summary <- firmfinBU2015U %>%
2         filter(!is.na(tax)) %>%
3         filter(!is.na(pl_before_tax)) %>%
4         filter(!is.na(assets_total)) %>%
5         filter(pl_before_tax > 0) %>%
6         group_by(firmid) %>%
7         summarize(ROA = pl_before_tax/assets_total,
8                   ETR = tax/pl_before_tax)
```

ここで，1行目から4行目の入力で，オブジェクト firmfinBU2015U における税金（tax），税引前利益（pl_before_tax），資産合計（assets_total）における欠測値を dplyr パッケージに付属する関数 filter と !is.na を使って取り除いている．また，5行目では税引前利益（pl_before_tax）が正の値のみに限定（filter(pl_before_tax > 0)）している．さらに，6行目で group_by 関数を用いて会社毎にグループ分けした後，7，8行目で各会社の総資産利益率（Return On Asset: ROA）と実効税率（EffectiveTax Rate: ETR）を関数 summarize を使って求めている．なお，総資産利益率は税引前利益を資産合計で割ったもの（ROA = pl_before_tax/assets_total）で定義され，実効税率は税金を税引前利益で割ったもの（ETR = tax/pl_before_tax）で定義されている．要約の結果として得られるオブジェクト firmfin.ROA.ETR.2015.firm.summary は，各会社の社名と ROA，ETR の値を列として持つデータ・フレーム・オブジェクトであり，具体的には以下のようなものである：

スクリプト12 オブジェクト firmfin.ROA.ETR.2015.firm.summary

```
1 > firmfin.ROA.ETR.2015.firm.summary
2 # A tibble: 5,163,037 x 3
```

```
3     firmid                                                                ROA      ETR
4     <chr>                                                               <dbl>    <dbl>
5    1 _SEATECHRIM RU77245522                                             0.0625  0
6    2 - - LTD SD SD BG010329873                                          0.295   0
7    3 - -2 OOD BG103708518                                               0.499   0.103
8    4 - 04 OOD BG115830330                                               0.159   0.0994
9    5 - 06 EOOD BG160096481                                              0.286   0.25
10   6 - 1 LTD EOOD BG200012549                                           0.118   0
11   7 - 1 ZHITLOVO-EKSPLUATATSIYNOI KONTORI-13 ZHITLOVO-BUDIVELNIH … 0.312   0
12   8 - 11 LTD EOOD BG203183473                                          0.5     0
13   9 - 15 LTD OOD BG203534214                                           0.941   0.125
14  10 - 18 O FR480571280                                                 0.318   0.322
15  # … with 5,163,027 more rows
```

オブジェクト firmfin.ROA.ETR.2015.firm.summary に対して，以下のようなスクリプトを実行することによって ETR を横軸，ROA を縦軸とする散布図を描く（Saka, *et al.*, 2019 参照）．

スクリプト13　企業の ROA と ETR の散布図を描画するための R スクリプト

```
1 > library(ggplot2)
2 > p2015 <- firmfin.ROA.ETR.2015.firm.summary %>%
3         ggplot(aes(ETR, ROA)) +
4         geom_point(size = 0.01, alpha = 0.01) +
5         xlim(-1, 1) + ylim(0, 1) + labs(title = 2015)
6 > png("ROA-ETR-2015.png")
7 > print(p2015)
8 > dev.off()
```

このスクリプトにおいて，１行目で ggplot2 パッケージを読み込んでおり，２行目から５行目でオブジェクト firmfin.ROA.ETR.2015.firm.summary をパイプラインでつなぎ，ggplot 関数を使って散布図を描くためのレイヤーを重ねた結果をプロットオブジェクト p2015 に付値している．さらに，６行目から８行目で，プロットオブジェクト p2015 を関数 png を使って PNG[21] ファイル ROA-ETR-2015.png に書き出している．図17はこのファイルを描画したものである．

なお，本小節でのデータラングリングから可視化までの全工程を図18に与える．

ここで与えた単年のデータに対する可視化は，時間をかければ３節で述べ

21）PNG は，Portable Network Graphics の略であり，画像データを圧縮して記録するファイル形式の一つである（IT 用語時点 http://e-words.jp/ 参照）．

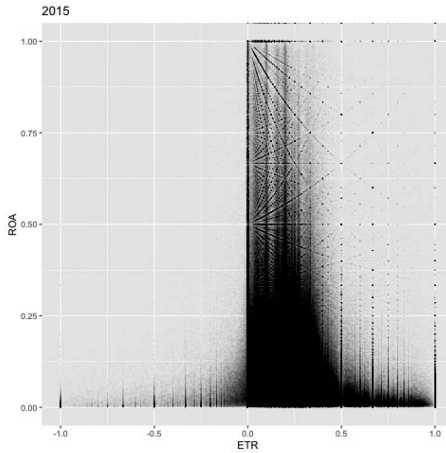


図17　**ROA**（縦軸）と **ETR**（横軸）の散布図：2015年

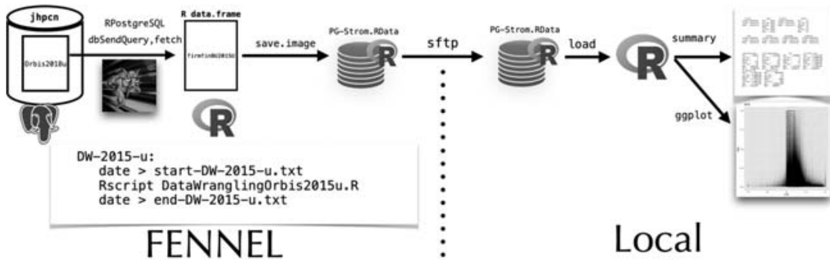


図18　2015年のデータラングリング，要約，可視化の全工程のイメージ

た Spark 環境と R 環境を利用することによっても実現可能であったが，複数年にわたるデータ（例えば，Orbis データには10年分のデータが収録されている）に対して同様の可視化を実行することは，実行時間の観点から現実的ではなかった．この問題に対して，本小節で行った **PG-Strom** 環境を利用することによって，利用可能な全期間にわたるデータを使って可視化を行うことを次に考える．

2　総資産利益率と実効税率の散布図の描画：2008年から2017年の場合

前小節では，2015年におけるデータにもとづいて，総資産利益率と実効税率の散布図を描く手順について詳しく見たが，ここでは，期間を2008年から2017年の10年間に拡大し，データラングリングと可視化について make コマンドによって自動実行する方法を考える．

まず，10年分のデータラングリングを行うために，Makefile にスクリプト14に与えられているようなターゲット DW-u を用意した．

スクリプト14　**make** ファイル **Makefile**：ターゲット **DW-u**

```
1  DW-u:
2  	date > start-DW-u.txt
3  	Rscript DataWranglingOrbis2018u.R
4  	date > end-DW-u.txt
```

ここで，R スクリプト DataWranglingOrbis2018u.R は以下のようなものである：

スクリプト15　R スクリプトファイル **DataWranglingOrbis2018u.R**

```
1  library(RPostgreSQL)
2  drv <- dbDriver("PostgreSQL")
3  con <- dbConnect(drv = drv.PG, host = "xxxx", port=5432, user= "yyyy", password
       = "********", dbname = "jhpcn")
4  # Data Wrangling from orbis2018u
5  # 2008
6  sql2008 <- "select␣firmID,␣year,␣country,␣month,␣cons,␣listed,␣exchange,␣
     InfoProv,␣interest_paid␣costs_employees,␣tax,␣PL_after_tax,␣PL_before_tax,
     ␣assets_total␣from␣orbis2018u␣where␣year␣=␣2008␣and␣month␣=␣12␣and␣(cons␣=␣
     'U1'␣or␣cons␣=␣'U2')"
7  rs2008 <- dbSendQuery(con, sql2008)
8  firmfinBU2008U <- fetch(rs2008, n = -1)
9  :
10 : (中略)
11 :
12 # 2017
13 sql2017 <- "select␣firmID,␣year,␣country,␣month,␣cons,␣listed,␣exchange,␣
     InfoProv,␣interest_paid,␣costs_employees,␣tax,␣PL_after_tax,␣PL_before_tax,
     ␣assets_total␣from␣orbis2018u␣where␣year␣=␣2017␣and␣month␣=␣12␣and␣(cons␣=␣
     'U1'␣or␣cons␣=␣'U2')"
14 rs2017 <- dbSendQuery(con, sql2017)
15 firmfinBU2017U <- fetch(rs2017, n = -1)
16 # Save R object
17 save.image(file = "DataWranglingOrbis2018u.RData")
```

スクリプト14の1行目から4行目で定義されているターゲット DW-u を以下のようにターミナルで実行することによって，ラングリングされたオブジェクトが RDATA ファイル DataWranglingOrbis2018u.RData に出力される（図19も参照）．

ターゲット DW-u の実行

```
$ make DW-u
```

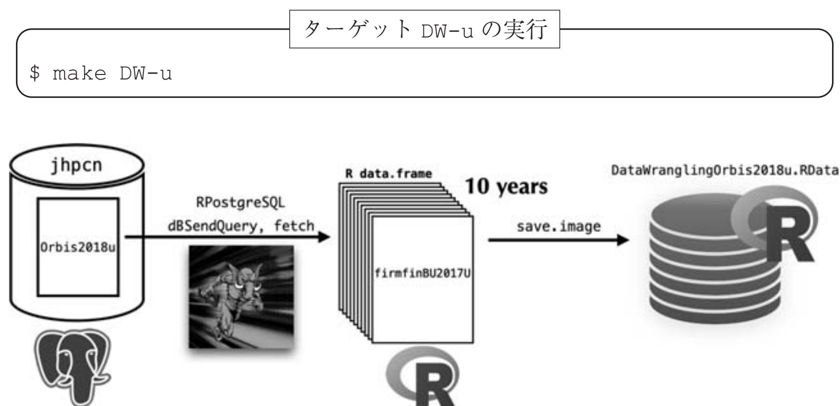


図19　ターゲット **DW-u** に対する **make** の実行によって実行される処理

以上の処理は，FENNEL 環境のもとで実行しており，この工程にかかった時間の計測結果は，以下のようなものである：

ターゲット DW-u に対する make の実行時間

```
$ cat start-DW-u.txt
2019年 12月 3日 火曜日 17:13:52 JST
$ cat end-DW-u.txt
2019年 12月 3日 火曜日 17:54:47 JST
```

この結果から，40分55秒かかったことがわかる．

この処理によって得られた RDATA ファイル DataWranglingOrbis2018u.RData の容量は，2.75 GB であり，一旦ファイルを sftp コマンドを利用してローカル転送した後，可視化を行う（図20参照）．

次に，可視化を行うために Makefile にスクリプト16に与えられているようなターゲット png を用意した．

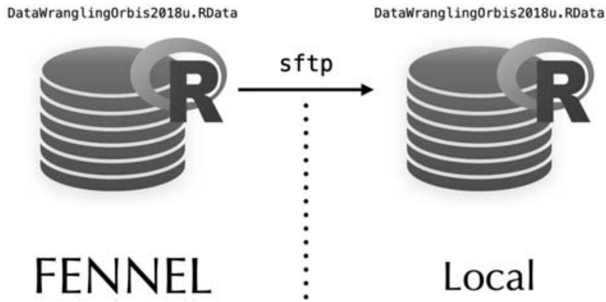


図20　RDATA ファイル **DataWranglingOrbis2018u.RData** のローカルへの転送

スクリプト16　**make** ファイル **Makefile**: ターゲット **png** と **animation**

```
1  png:
2  	date > start-png.txt
3  	Rscript makepng.R
4  	date > end-png.txt
```

ここで，R スクリプト `makepng.R` は可視化した結果を PNG ファイルとして書き出すためのものでありスクリプト17で定義される.

スクリプト17　R スクリプトファイル **makepng.R**

```
1  load("./DataWranglingOrbis2018u.RData")
2  mkpng <- function(df, year)
3  {
4    require(ggplot2)
5    require(dplyr)
6    p <- df %>%
7      filter(!is.na(tax)) %>%
8      filter(!is.na(pl_before_tax)) %>%
9      filter(!is.na(assets_total)) %>%
10     filter(pl_before_tax > 0) %>%
11     group_by(firmid) %>%
12     summarize( ROA = pl_before_tax/assets_total,
13                ETR = tax/pl_before_tax) %>%
14     ggplot(aes(ETR, ROA)) +
15     geom_point(size = 0.01, alpha = 0.01) +
16     xlim(-1, 1) + ylim(0, 1) + labs(title=year)
17   png(paste("ROA-ETR-", year, ".png", sep = ""))
18   print(p)
19   dev.off()
20 }
21 mkpng(firmfinBU2008U, year = 2008)
22 mkpng(firmfinBU2009U, year = 2009)
```

```
23 mkpng(firmfinBU2010U, year = 2010)
24 mkpng(firmfinBU2011U, year = 2011)
25 mkpng(firmfinBU2012U, year = 2012)
26 mkpng(firmfinBU2013U, year = 2013)
27 mkpng(firmfinBU2014U, year = 2014)
28 mkpng(firmfinBU2015U, year = 2015)
29 mkpng(firmfinBU2016U, year = 2016)
30 mkpng(firmfinBU2017U, year = 2017)
```

スクリプト17の１行目で，RDATA ファイル DataWranglingOrbis2018u.RData を読み込んでおり，２行目から20行目で散布図を PNG ファイルへ書き出すための関数を定義している．なお，21行目から30行目で実際に2008年から2017年のデータを利用した散布図を PNG ファイルへ書き出す関数を実行している．

ターゲット png を以下のようにターミナルで実行することによって，スクリプト16の２行目から４行目で定義されているシェルスクリプトが実行され，10年分の PNG ファイル ROA-ETR-2008.png ～ ROA-ETR-2017.png が出力される（図21参照）．

ターゲット png の実行

```
$ make png
```

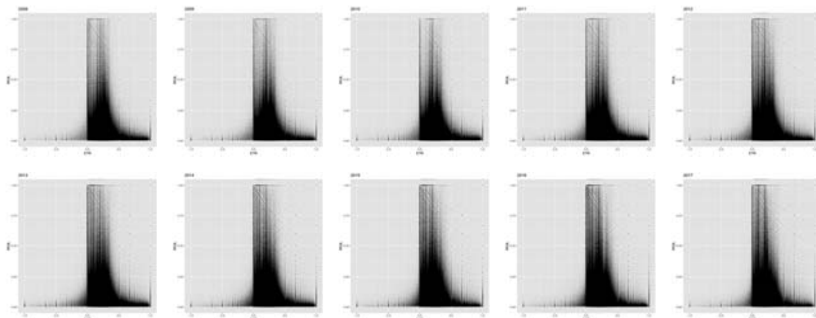


図21　ETR（横軸）と ROA（縦軸）の散布図：2008年～2017年

以上の処理は，ローカル環境のもとで実行しており，この工程（10個の PNG ファイルの出力）にかかった時間の計測結果は，以下のようなもので

ある：

ターゲット png に対する make の実行時間

```
% cat start-png.txt
2020年 5月 1日 金曜日 16時10分44秒 JST
% cat end-png.txt
2020年 5月 1日 金曜日 16時39分11秒 JST
```

この結果から，28分27秒かかっていることがわかる．

さらに，10年分の PNG ファイル ROA-ETR-2008.png ～ ROA-ETR-2017.png をまとめてアニメーション GIF[22] ファイルに変換するために Makefile にスクリプト18に与えられているようなターゲット animation を用意した．

スクリプト18　make ファイル Makefile：ターゲット animation

```
1 animation:
2 	date > start-animation.txt
3 	convert -layers optimize -loop 0 -delay 40 ROA-ETR-????.png animation.gif
4 	date > end-animation.txt
```

ターゲット animation を以下のようにターミナルで実行することによって，スクリプト18の２行目から４行目で定義されているシェルスクリプトが実行され，10年分の PNG ファイル ROA-ETR-2008.png ～ ROA-ETR-2017.png が convert[23] コマンドによってアニメーション GIF ファイル animation.gif へ変換される（紙面では動きを表現できないので図は割愛する）．

ターゲット animation の実行

```
$ make animation
```

22）GIF は，Graphics Interchange Format の略であり，画像データを圧縮して記録するファイル形式の一つである．アニメーション GIF はアニメーション機能を持つように拡張された仕様の一つで，動画を保存することができる形式である．（IT 用語時点 http://e-words.jp/ 参照.）

23）convert は，ImageMagick（https://imagemagick.org/index.php）に付属する画像ファイルを変換するためのコマンドである．

以上のターゲット png，animation を実行することによって得られる可視化の工程のイメージを図22に与える．

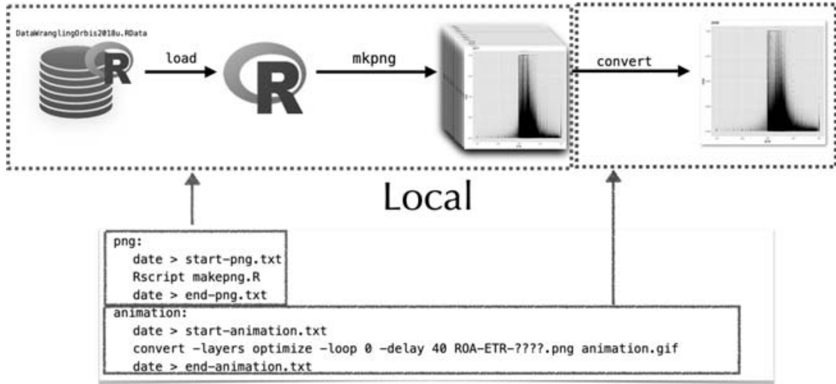


図22　可視化の工程のイメージ

さらに本小節で行ったデータラングリングから可視化までの全工程を図23に与える．なお，これらの実行時間は，FENNEL 環境で行った処理に約41分，ローカル環境では約28分であり，（転送時間を除いて）全体で 1 時間強である．

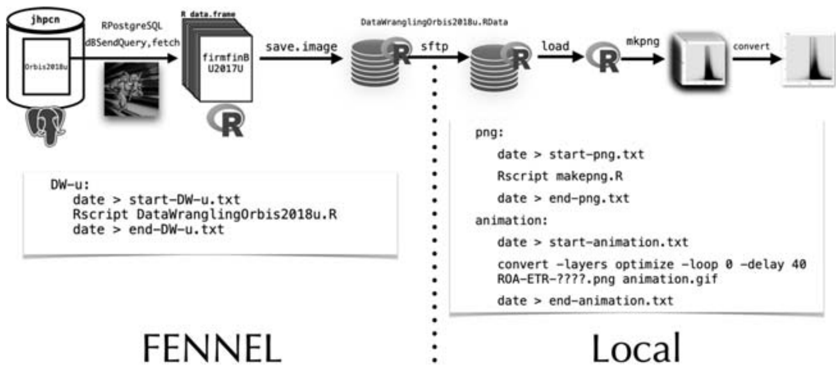


図23　2008年～2017年のデータラングリング，可視化の全工程のイメージ

VI　おわりに

本稿では，**PG-Strom** 環境を利用したデータラングリングにもとづく処理速度の向上と，得られたデータの要約と可視化を行うことによって探索的データ解析の実行可能性について検討した．なお，`Makefile` にターゲットを作成することができる工程は，`make` を用いて自動実行することによって，再現可能性をもたせた．

また，本稿で行ったデータの可視化は，R の作業空間のイメージを RDATA ファイルに出力することによって保存し，ローカル環境に一旦転送したものを再度読み込み，R を用いて行った．この理由としては，ネットワークを超える環境でリアルタイムで可視化することが速度などの観点から難しいためであるが，RDATA ファイルの「書き出し」，「転送」，「読み込み」にかかる時間は無視できないものがある（図24の破線内の工程を参照）．

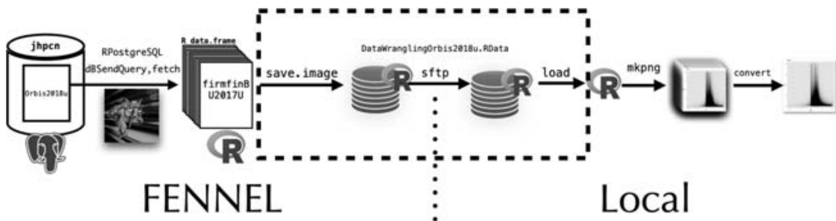


図24　RDATA ファイルの「書き出し」，「転送」，「読み込み」

この問題点をふまえ，今後統計モデリングを含む探索的データ解析をさらに本格的に実行するために検討しなければならないこととしては以下のことがあげられる：

（M1）シリアライズの再検討
（M2）データラングリングのさらなる高速化

ここで，「書き出し」，「転送」，「読み込み」の時間を改善するための簡単な方法としては，（M1）で指摘しているように R の作業空間をシリアライ

ズ（serialize）するファイル形式を見直すことであろう．つまり，R の標準的なシリアライズのファイル形式である RDATA ファイルではなく，Apache Arrow[24] や Apache Parquet[25] などの形式でシリアライズすることによってファイルサイズを縮小したり，書き出し・読み込みにかかる時間を大幅に短縮できる可能性がある．このことは，ファイル転送の時間短縮にもつながることから，次の課題として検討する予定である．

また，（M2）については，**PG-Strom** と列指向データストア（Arrow Fdw）[26] を併用をすることによって，さらなる時間短縮を実現できることが報告されている[27]．このことも，今後の課題としたい．

（筆者は関西学院大学商学部教授）

参考文献

［1］ Janssens, J.（2014）*Data Science at the Command Line*, O'Reilly Media.（太田満久，下田倫大，増田泰彦監訳，長尾高弘訳（2015）『コマンドラインではじめるデータサイエンス―分析プロセスを自在に進めるテクニック―』，オライリー・ジャパン.）

［2］ 地道正行（2018-a）『探索的財務ビッグデータ解析―前処理，データラングリング，再現可能性―』，商学論究，第66巻，第1号，pp. 1-31，関西学院大学商学研究会.

［3］ 地道正行（2018-b）『探索的財務ビッグデータ解析―データ可視化，統計モデリング，モデル選択，モデル評価，動的文書生成，再現可能研究―』，商学論究，第66巻，第2号，pp. 1-41，関西学院大学商学研究会.

［4］ 地道正行（2020）『探索的財務ビッグデータ解析―前処理の並列化―』，商学論究，第67巻，第3号，pp. 1-19，関西学院大学商学研究会.

［5］ 海外浩平（2019-a）『PostgreSQL は最新ハードウェアでどこまでやれるのか？ GPU と NVME で実現する超高速ログデータ処理基盤』，https://www.sraoss.co.jp/event_seminar/2019/20190418_SRAOSS_seminar_PGStrom_on_ArrowFdw.pdf

［6］ 海外浩平（2019-b）『Arrow Fdw ―PostgreSQL で大量のログデータを処理するためのハードウェア最適化アプローチ―』，20191115-PGconf.Japan，https://www.slideshare.net/kaigai/20190314-pgstrom-arrowfdw/

24）https://arrow.apache.org/

25）https://parquet.apache.org/

26）https://heterodb.github.io/pg-strom/ja/arrow_fdw/

27）https://kaigai.hatenablog.com/ の2019年10月31日の記事『秒速で10億レコードを処理する話』を参照されたい．

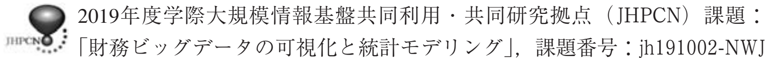
[７] Patil, DJ（2012）*Data Jujitsu: The Art of Turning Data into Product,* An O’Reilly Radar Report, O’Reilly.
[８] Saka, C., T. Oshika, and M. Jimichi（2019）Visualization of tax avoidance and tax rate convergence: Exploratory analysis of world-scale accounting data, *Meditari Accountancy Research,* Vol. 27 No. 5, pp. 695-724, Emerald Publishing Limited.
[９] Tange, Ole,（2018）*GNU Parallel 2018*, ISBN: 9781387509881, DOI: 10.5281/zenodo.1146014, URL: `https://doi.org/10.5281/zenodo.1146014`, Mar, 2018.
[10] Wickham, H. and G. Grolemund（2016）*R for Data Science,* O’Reilly.

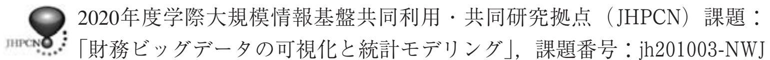
謝辞

本研究の一部は以下の助成を得ている．さらに，東京大学の宮本大輔氏には **PG-Strom** 環境を提供して頂き，BvD の増田歩氏にはデータの抽出に関して多大なるご協力いただいた．ここに感謝の意を表する．

科研費 科学研究費基盤研究 C：「グラフィカル・データ・アナリシスによる格差研究と社会環境会計による解決方法の提案」（2016年～2018年），課題番号：16K04022

科研費 科学研究費基盤研究 C：「共有価値創造（CSV）のための社会環境会計の構築」（2019年～2021年），課題番号：19K02006


2019年度学際大規模情報基盤共同利用・共同研究拠点（JHPCN）課題：「財務ビッグデータの可視化と統計モデリング」，課題番号：jh191002-NWJ


2020年度学際大規模情報基盤共同利用・共同研究拠点（JHPCN）課題：「財務ビッグデータの可視化と統計モデリング」，課題番号：jh201003-NWJ

関西学院大学図書館図書費 B，研究設備費（III），個人研究費

付録　計算機環境

本研究のために利用した環境としては，東京大学情報基盤センター[28]に設置された専有利用型リアルタイムデータ解析ノード（FENNEL）を利用した．１ノードの簡単な仕様を以下に与える[29]：

28）`https://www.itc.u-tokyo.ac.jp/`
29）今回利用しているノードは，GPU がある環境が２ノードであり，GPU がない環境が２ノードの合計４である．

OS: Ubuntu 16.04

CPU: Intell® Xeon® プロセッサー E5-2620 v4, 8 Cores

Main Memory: 16 GB

Storage: 1 TB

GPU Memory: 8 GB

また，本研究で利用した **PG-Strom** 環境は，実験的に FENNEL 環境にネットワーク接続した以下の環境を利用させていただいた：

OS: CentOS 7.7

CPU: Intel® Xeon® Bronze 3104, 6 Cores

Main Memory: 128 GB

Storage: 600 GB

GPU Unit and Memory: NVIDIA® Tesla® V100, 32 GB

なお，ローカル環境は以下のようなものである：

OS: macOS Catalina (10.15.4)

CPU: Intel® Core™ i9-8950HK, 6 Cores

Main Memory: 32 GB

Storage: 4 TB