Techniki eksploracji danych

Krzysztof Gajowniczek

Rok akademicki: 2020/2021

Section 1

Teoretyczne podstawy przetwarzania równoległego

- Przetwarzanie dużych ilości danych za pomocą złożonych modeli może być czasochłonne.
- Nowe rodzaje wykrywania oznaczają, że skala gromadzenia danych jest obecnie ogromna, a modelowane wyniki mogą być również duże.
- Jeśli coś zajmuje mniej czasu, jeśli odbywa się poprzez **przetwarzanie równoległe**, dlaczego nie zrobić tego i zaoszczędzić czas?
- Nowoczesne laptopy i komputery PC mają obecnie **procesory wielordzeniowe** z wystarczającą ilością dostępnej pamięci, którą można wykorzystać do szybkiego generowania wyników.

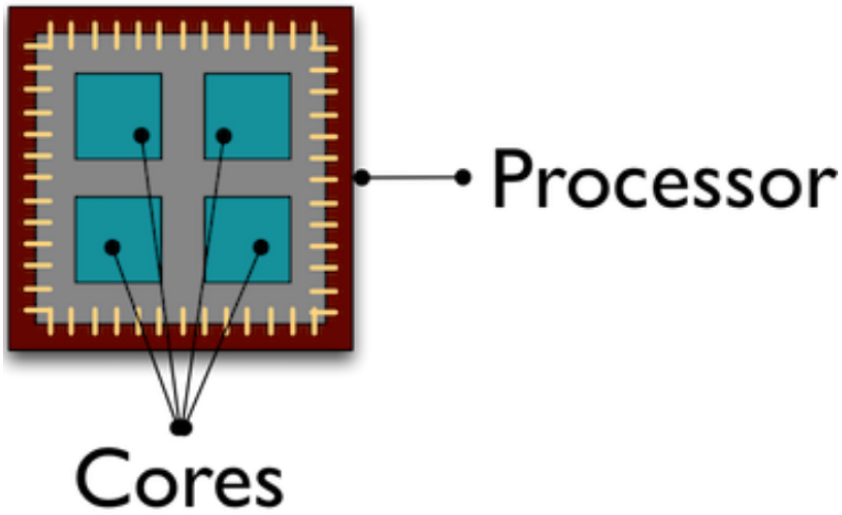
- Zrównoleglenie kodów ma swoje liczne zalety.
- Zamiast czekać kilka minut lub godzin na zakończenie zadania, można podmienić kod, uzyskać dane wyjściowe w ciągu kilku sekund lub minut i jednocześnie sprawić, by było to wydajne.
- Efektywność kodu jest obecnie jedną z najbardziej poszukiwanych umiejętności w branży i niewiele osób jest w stanie z niej korzystać.
- Kiedy już nauczysz się zrównoleglania kodu, będziesz tylko żałować, że nie nauczyłeś się go wcześniej.

- Wiele programów R działa szybko i dobrze na jednym procesorze. Ale czasami obliczenia mogą być:

- **cpu-bound**: zajmuje zbyt dużo czasu procesora.
- **memory-bound**: zajmuje zbyt dużo pamięci.
- **I/O-bound**: odczyt/zapis z dysku zajmuje zbyt dużo czasu.
- **network-bound**: przesyłanie zajmuje zbyt dużo czasu.

- Sercem każdego komputera jest nowoczesny procesor CPU (ang. Central Processing Unit).
- Podczas gdy tradycyjne komputery miały jeden procesor, nowoczesne komputery posiadają wiele procesorów, z których każdy może zawierać wiele rdzeni.
- Te procesory i rdzenie są dostępne do wykonywania obliczeń.

- Komputer z jednym procesorem może nadal mieć 4 rdzenie (quad-core), co pozwala na jednoczesne wykonywanie 4 obliczeń.



Section 2

Podejścia do równoległości w R

- R może korzystać z kilku struktur/narzędzi/pakietów, aby umożliwić zrównoleglenie:

- parallel jest prawdopodobnie najbardziej rozwiniętym i najbardziej dojrzałym i opiera się na pracy z pakietem multicore i snow (z których pierwszy został usunięty z CRAN, ponieważ został całkowicie wchłonięty przez parallel), z których drugi może być używane do zrównoleglania obliczeń na jednym lub wielu komputerach.
- foreach umożliwia zrównoleglenie poprzez rozszerzenia foreach pętlifor, pakiet doMC i słowo kluczowe %dopar%. Może korzystać z frameworka multicore, z tymi samymi ograniczeniami systemu operacyjnego wymienionymi powyżej, lub frameworka snow.

Subsection 1

Pakiet parallel

```
library(parallel)
no_cores <- detectCores()
clust <- makeCluster(no_cores)
parLapply(clust,1:3, function(x) c(x^2,x^3))
```

```
## [[1]]
## [1] 1 1
##
## [[2]]
## [1] 4 8
##
## [[3]]
## [1]  9 27
```

```
stopCluster(clust)
```

```
library(parallel)
no_cores <- detectCores()
clust <- makeCluster(no_cores)
base <- 4
clusterExport(clust, "base")
parSapply(clust, 1:5, function(exponent) base^exponent)
```

```
## [1]    4   16   64  256 1024
```

```
stopCluster(clust)
```

Subsection 2

Pakiety doParallel i foreach

```
library(doParallel)
```

```
## Loading required package: foreach
```

```
## Loading required package: iterators
```

```
cl <- makeCluster(2)
registerDoParallel(cl)
foreach(i=1:3) %dopar% sqrt(i)
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 1.414214
##
## [[3]]
## [1] 1.732051
```

- Pakiet doParallel umożliwia określenie różnych opcji podczas uruchamiania polecenia foreach, które są obsługiwane przez podstawową funkcję mclapply: "preschedule", "set.seed", "silent" i „cores".

```
mcoptions <- list(preschedule=FALSE, set.seed=FALSE)
foreach(i=1:3, .options.multicore=mcoptions) %dopar% sqrt(i
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 1.414214
##
## [[3]]
## [1] 1.732051
```

Section 3

Large memory and out-of-memory data

- R nie nadaje się dobrze do pracy ze strukturami danych większymi niż około 10-20% pamięci RAM komputera.
- Dane przekraczające 50% dostępnej pamięci RAM są zasadniczo bezużyteczne.
- Uważa się, że zbiór danych jest duży, jeśli przekracza 20% pamięci RAM na danej maszynie oraz ogromny, jeśli przekracza 50%.

- Wydajna reprezentacja danych w pamięci RAM skróci czas przetwarzania i pozwoli dopasować modele, które w innym przypadku wymagałyby ogromnej ilości pamięci RAM.
- Jednak nie wszystkie problemy są rzadkie (ang. sparse). Możliwe jest również, że dane nie mieszczą się w pamięci RAM, nawet jeśli są rzadkie.

- Istnieje kilka scenariuszy do rozważenia:

1. Dane mieszczą się w pamięci RAM, ale są zbyt duże, aby przeprowadzac na nich obliczenia - rozwiązaniem jest zastąpienie algorytmu, który jest używany.
2. Dane nie mieszczą się w pamięci RAM, ale mieszczą się w lokalnej pamięci (HD, SSD itp.).
3. Dane nie mieszczą się w lokalnej pamięci - jeśli dane nie mieszczą się w lokalnej pamięci, potrzeba będzie jakiegoś zewnętrznego rozwiązania do przechowywania, takiego jak rozproszony DBMS lub rozproszony system plików.

Subsection 1

Wydajne przetwarzanie z pamięci RAM

- Jeśli dane mieszczą się w pamięci RAM, ale nadal są zbyt duże, aby je przetwarzać (dopasowanie modelu wymaga około 5-10 razy więcej pamięci niż zapisanie go), istnieje kilka udogodnień do wykorzystania.
- Pierwsza to rzadka reprezentacja danych, która jest istotna, gdy używa się czynników (factors), które zazwyczaj są mapowane na rzadkie macierze modeli.
- Innym sposobem jest użycie *algorytmów pamięci zewnętrznej* (EMA).

- Funkcja `biglm::biglm` zapewnia EMA dla regresji liniowej. Poniższy przykład pochodzi z przykładu funkcji.

```
data(trees)
ff<-log(Volume)~log(Girth)+log(Height)

chunk1<-trees[1:10,]
chunk2<-trees[11:20,]
chunk3<-trees[21:31,]

library(biglm)
```

```
## Loading required package: DBI
```

```
a <- biglm(ff,chunk1)
a <- update(a,chunk2)
a <- update(a,chunk3)
coef(a)
```

```
## (Intercept)  log(Girth) log(Height)
##   -6.631617    1.982650    1.117123
```

- Rzeczy do zapamiętania:

- Dane zostały podzielone wierszami.
- Wstępne dopasowanie odbywa się za pomocą funkcji biglm.
- Model jest aktualizowany o kolejne fragmenty za pomocą funkcji aktualizacji.

- Porównanie z funkcją `lm`.

```
b <- lm(ff, data=trees)
rbind(coef(a),coef(b))
```

```
##      (Intercept) log(Girth) log(Height)
## [1,]   -6.631617    1.98265    1.117123
## [2,]   -6.631617    1.98265    1.117123
```

- Jeśli celem nie jest dopasowywanie żadnego modelu, a tylko chęć wydajnego filtrowania, selekcji i podsumowania statystyk, to należy wykorzystać pakiet `data.table`.
- Składnia jest mniej przyjazna niż w pakiecie `dplyr`, ale za to pakiet jest **BARDZO SZYBKI** w porównaniu do konkurencji.

```
library(data.table)
n <- 1e6
k <- c(200,500)
p <- 3
L1 <- sapply(k, function(x)
  as.character(sample(1:x, n, replace = TRUE) ))
L2 <- sapply(1:p, function(x) rnorm(n) )
```

```
tbl <- data.table(L1,L2)
colnames(tbl) <- c(paste("v",1:length(k),sep=""),
                   paste("x",1:p,sep="") )
tbl_dt <- tbl
tbl_df <- as.data.frame(tbl)
```

Subsection 2

Przetwarzanie z wykorzystaniem bazy danych

- Wczesne rozwiązania dla dużych zbiorów danych polegały na przechowywaniu danych w niektórych DBMS, takich jak *MySQL*, *PostgresSQL*, *SQLite*, *H2*, *Oracle*, itd. - Kilka pakietów `R` zapewnia interfejsy do tych DBMS, na przykład `sqldf`, `RDBI`, `RSQite`.
- Przechowywanie danych w DBMS ma tę zaletę, że zazwyczaj można polegać na dostawcach DBMS, którzy dołączą bardzo wydajne algorytmy do obsługiwanych zapytań.
- Z drugiej strony zapytania SQL mogą zawierać wiele statystyk podsumowujących, ale rzadko obejmują narzędzie do modelowania.
- Oznacza to, że nawet w przypadku prostych rzeczy, takich jak modele liniowe, będziesz trzeba powrócić do funkcji `R` - zazwyczaj jest to jakiś rodzaj EMA z fragmentami DBMS.

Subsection 3

Przetwarzanie z wydajnych struktur plików

- Istnieje kilka narzędzi, które umożliwiają zapisywanie i wykonywanie obliczeń bezpośrednio z dysku:

1. **Memory Mapping**: gdzie adresy RAM są mapowane do pliku w pamięci, poprzez rozszerzenie pamięci RAM do pojemności dysku (HD, SSD.). Wydajność nieznacznie się pogarsza, ale dostęp jest zwykle bardzo szybki. To podejście jest zaimplementowane w pakiecie `bigmemory`.
2. **Efficient Binaries**: Dane są przechowywane jako plik na dysku. Plik jest binarny, z dobrze zaprojektowaną strukturą, co ułatwia dzielenie go na fragmenty. To podejście jest zaimplementowane w pakiecie `ff`.

```
library("bigmemory")
library("pryr")
```

```
## Registered S3 method overwritten by 'pryr':
##   method      from
##   print.bytes Rcpp

##
## Attaching package: 'pryr'

## The following object is masked from 'package:data.table
##
##     address
```

```
big_table <- matrix(runif(10^4*10^4),10^4,10^4)
object_size(big_table)
```

```
## 800 MB
```

```
small_table <- as.big.matrix(big_table)
object_size(small_table)
```

```
## 696 B
```

Section 4

Literatura

- *Multicore Data Science with 'R' and Python*
- *Beyond Single-Core R* by Jonoathan Dursi (also see *GitHub repo for slide source*)
- *The venerable Parallel 'R'* by McCallum and Weston (a bit dated on the tooling, but conceptually solid)
- *http://www.bytemining.com/2010/08/taking-r-to-the-limit-part-ii-large-datasets-in-r/*
- Adler, D., Nenadic, O., Zucchini, W., & Glaser, C. (2007). *The "ff" package: Handling Large Data Sets in 'R' with Memory Mapped Pages of Binary Flat Files*. Institute for Statistics and Econometrics.
- Emerson, J. W., & Kane, M. J. (2009). *The 'R' Package bigmemory: Supporting Efficient Computation and Concurrent Programming with Large Data Sets*. Journal of Statistical Software, 10.