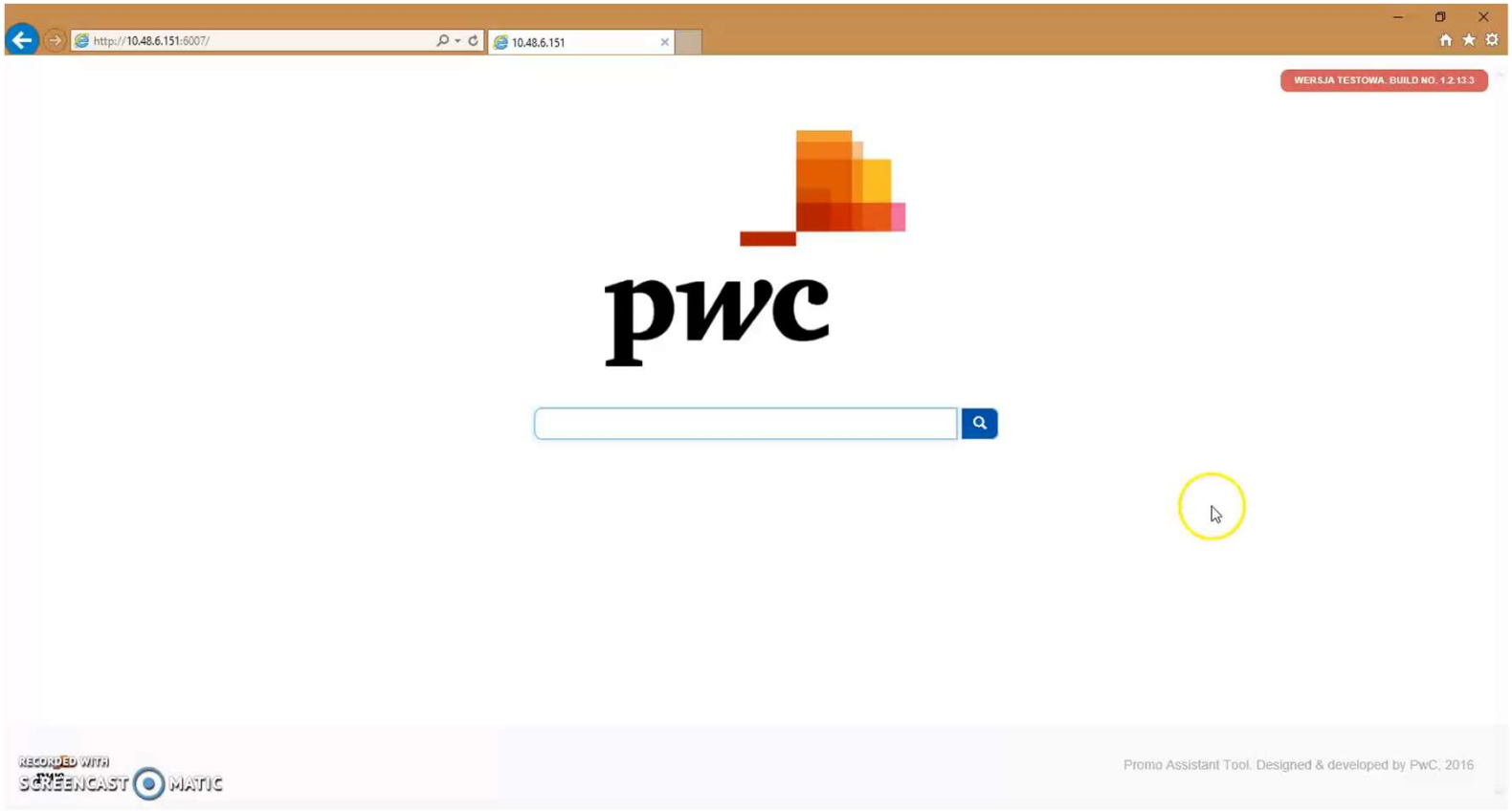
Rozpraszanie wielowątkowe na przykładzie analizy transakcji w Retailu

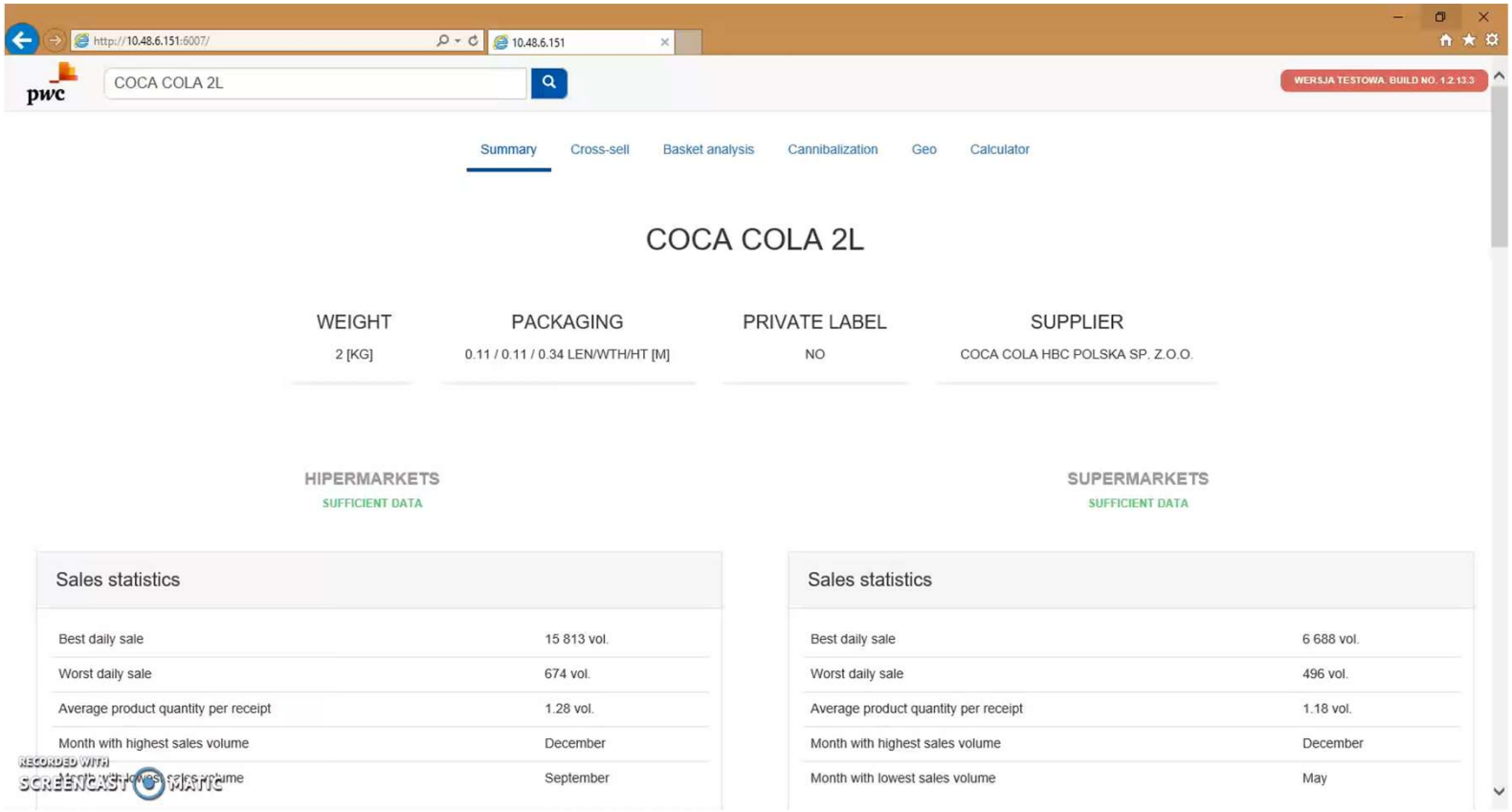
Rafał Kobiela i Michał Cisek

pwc

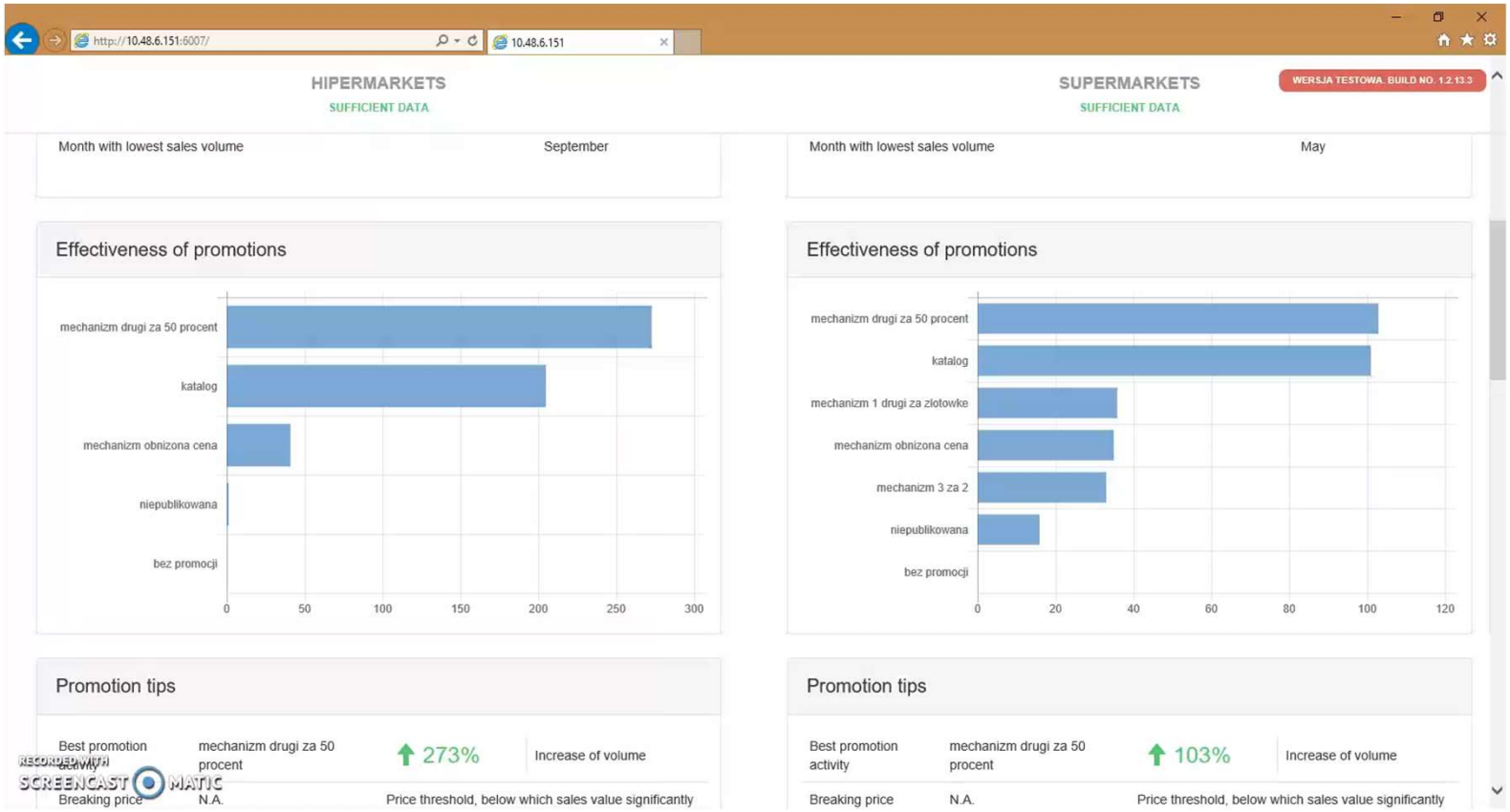
Dashboard – statystyki opisowe



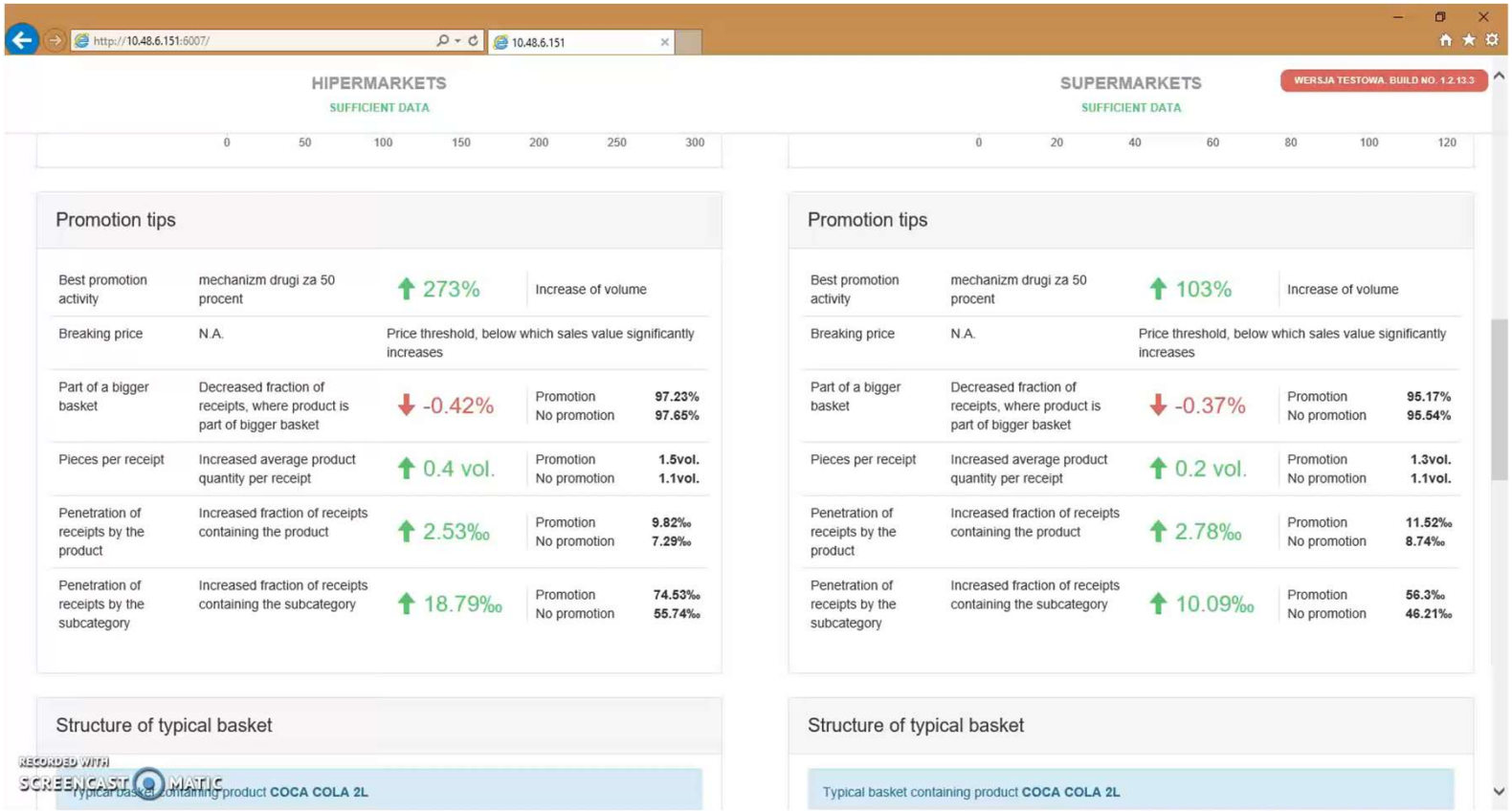
Dashboard – efektywność promocji



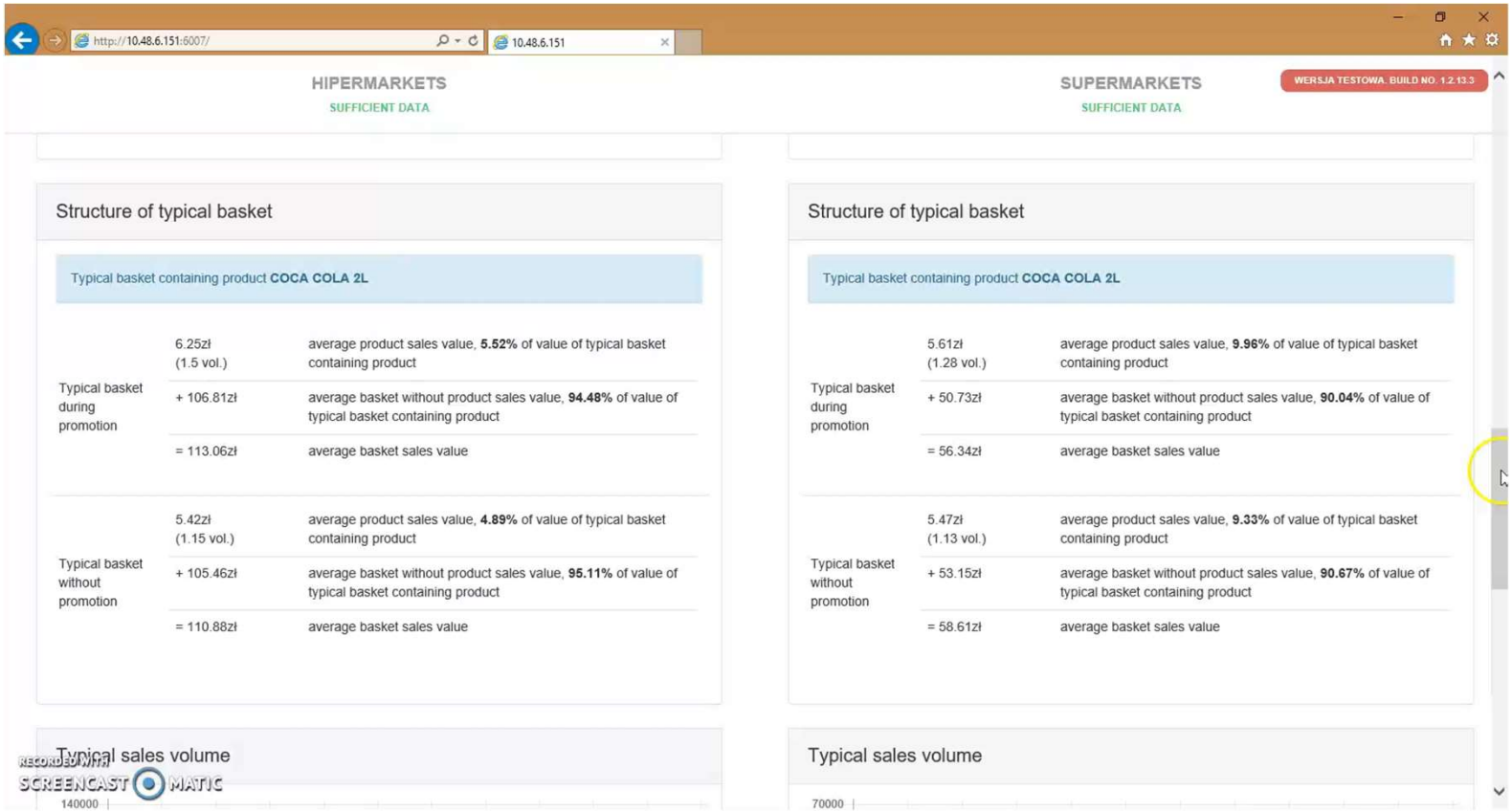
Dashboard – wskazówki marketingowe



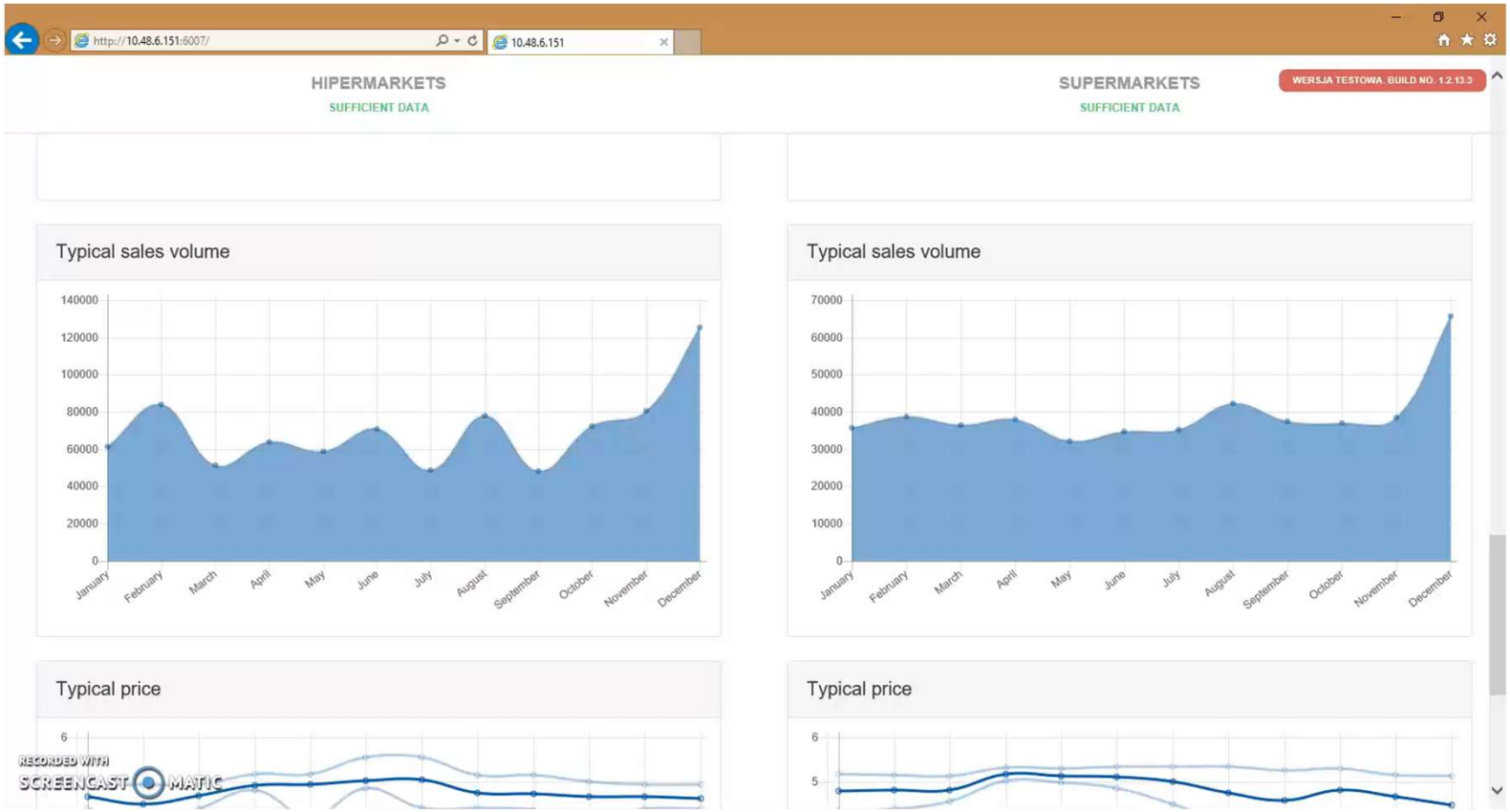
Dashboard – struktura koszyka



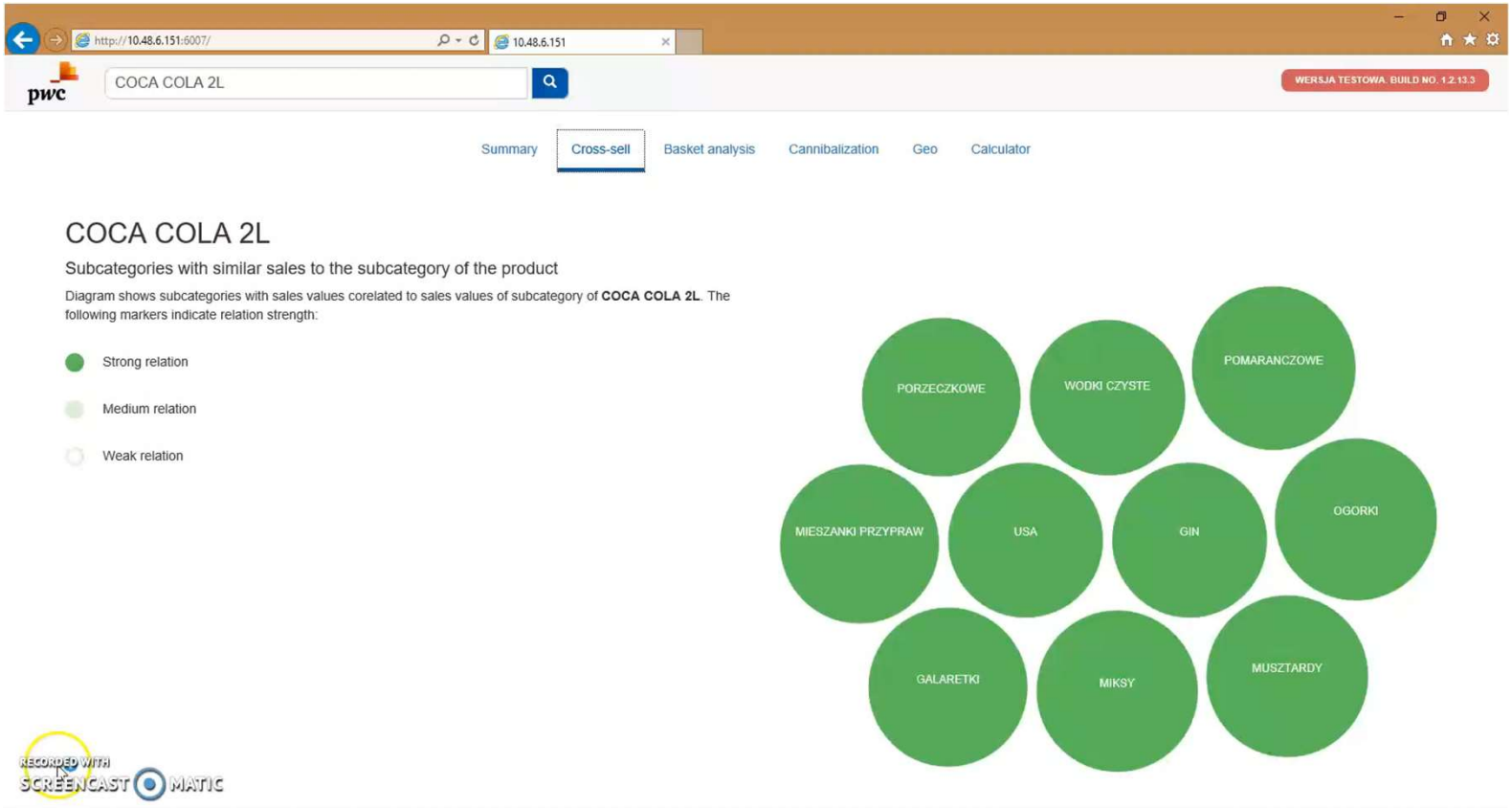
Dashboard – sezonowość sprzedaży



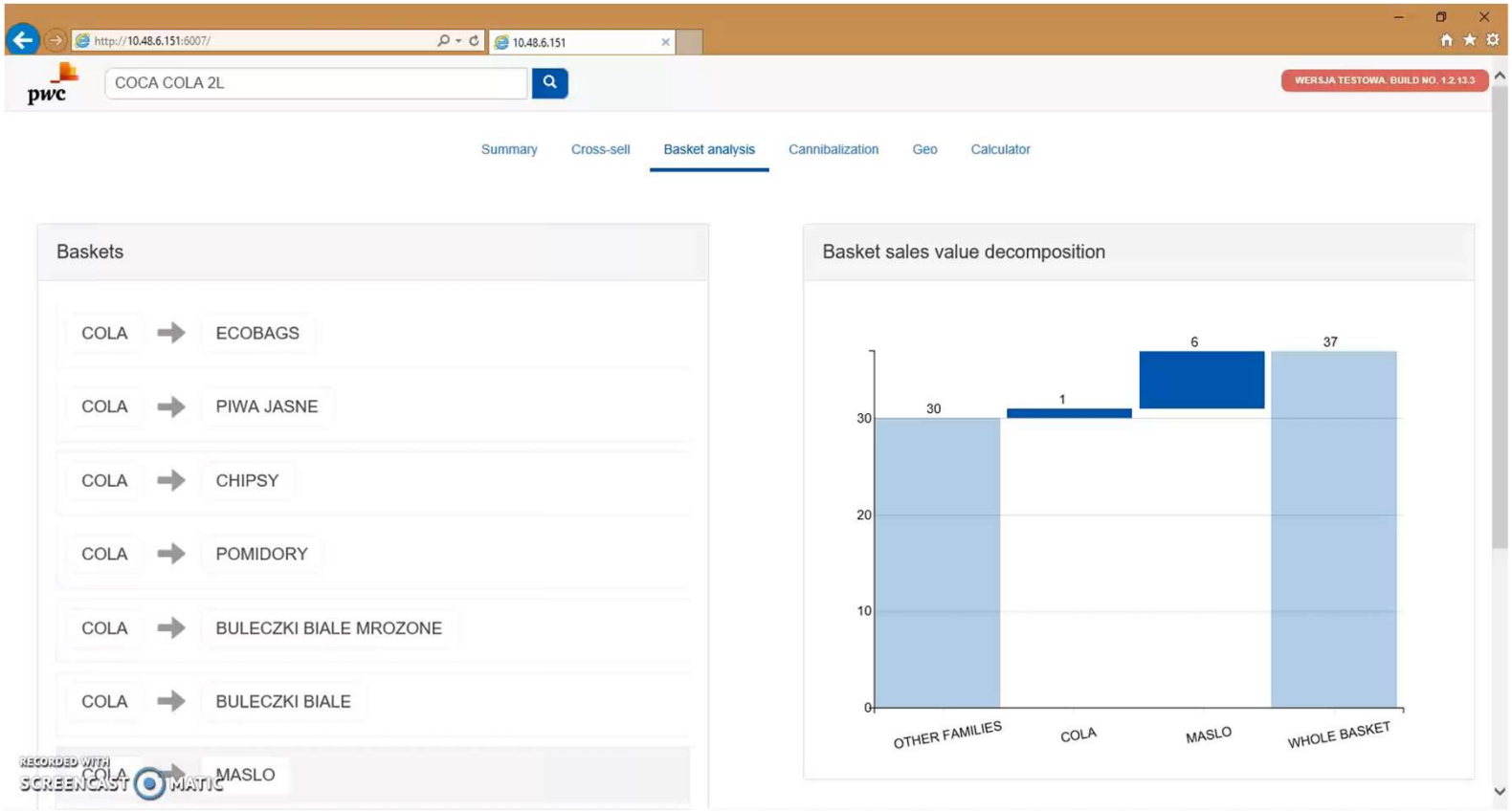
Dashboard – Cross-sell



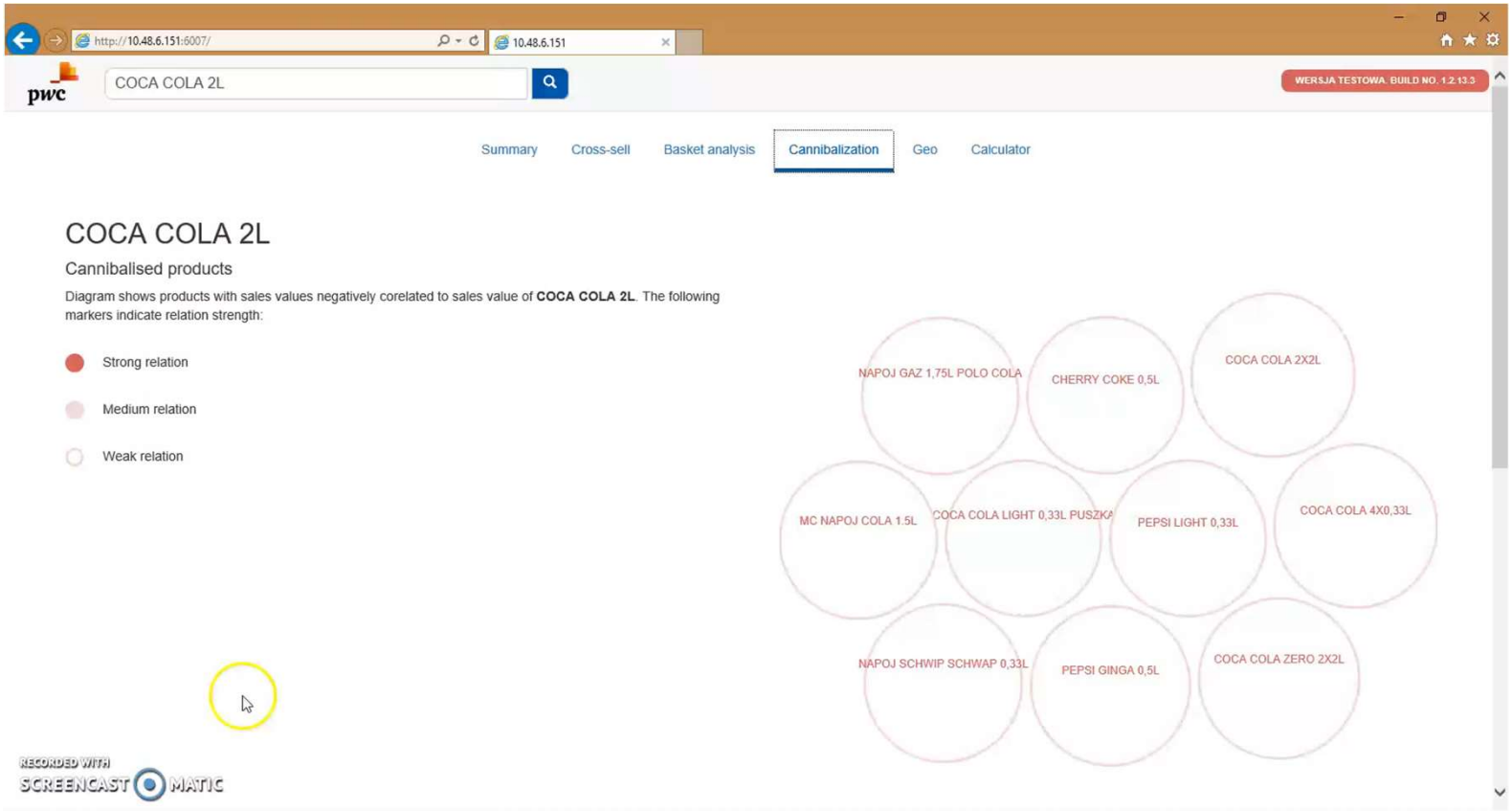
Dashboard – Analiza koszykowa



Dashboard – kanibalizacja

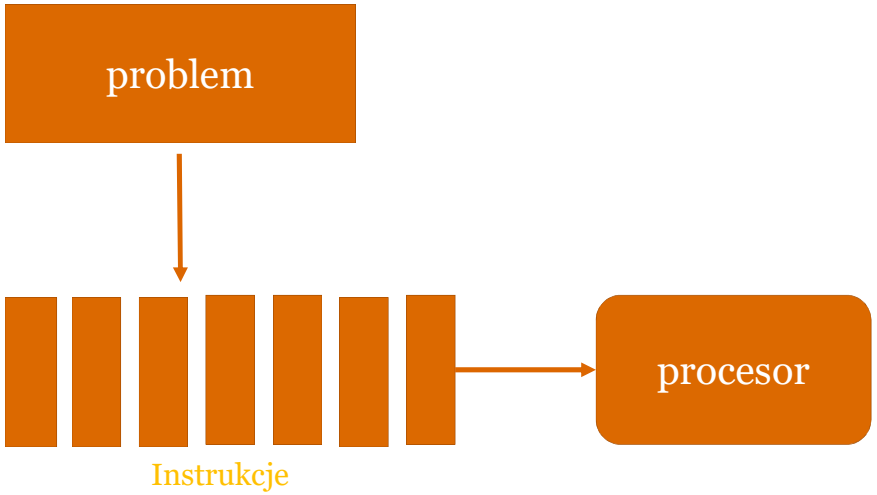


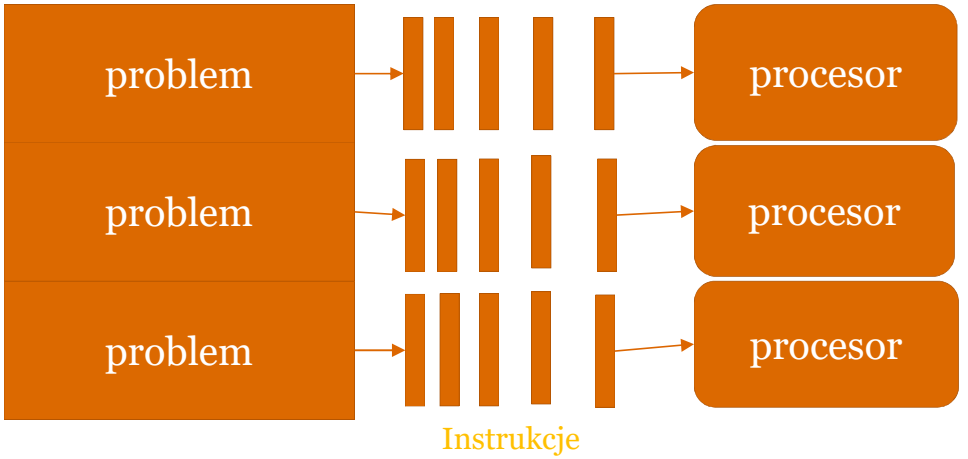
Dashboard – predykcja sprzedaży



Czym są obliczenia równoległe?

- wykonywanie więcej niż jednego zadania w tym samym czasie
- używanie wielu rdzeni tego samego procesora do różnych zadań
- używanie różnych komputerów połączonych w klaster obliczeniowych do różnych zadań

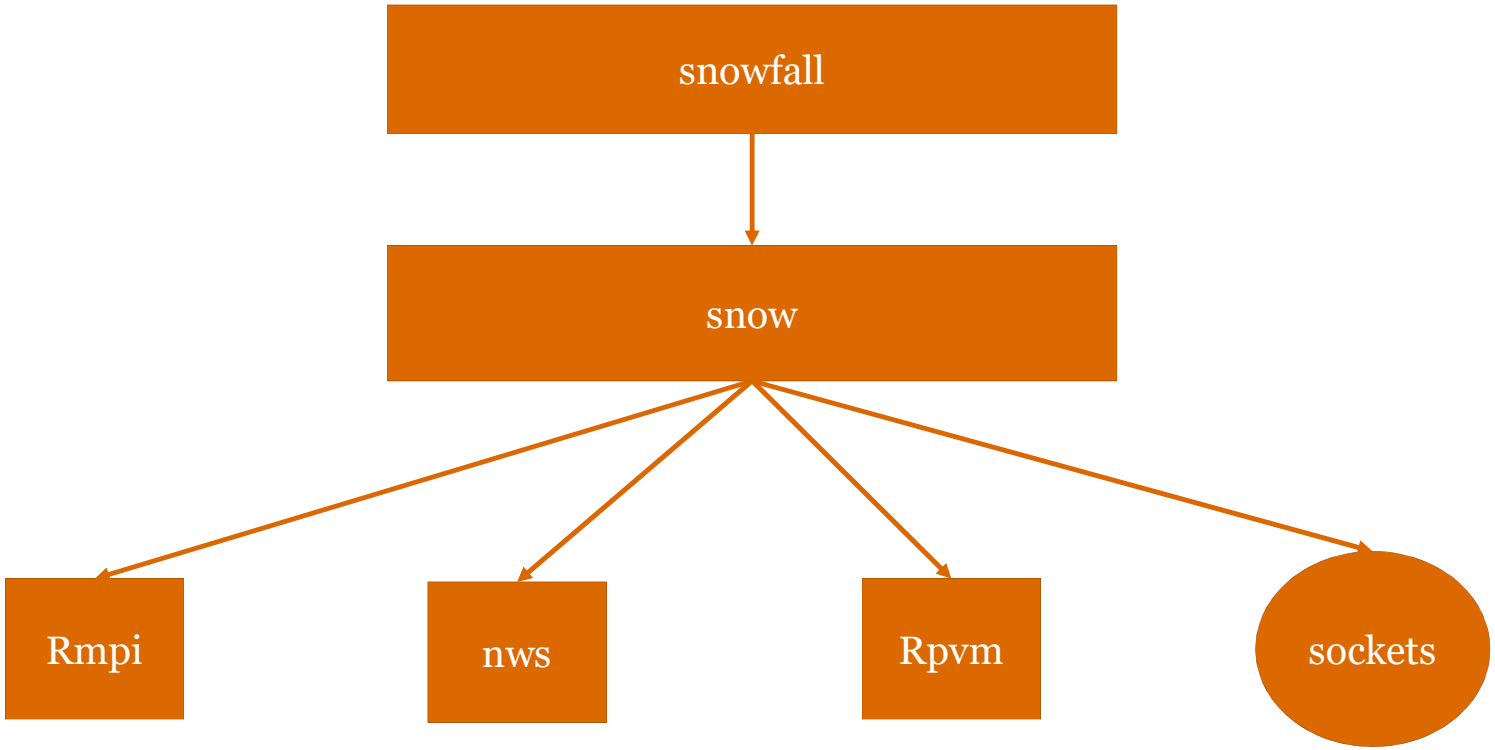




Dlaczego przydatne są równoległe obliczenia w R?

- Niezależnie od liczby rdzeni na procesorze, R będzie korzystał domyślnie tylko z jednego
- R czyta dane do pamięci domyślnie (łatwo wyczerpać pamięć RAM)
- Często pewien proces możemy powtórzyć wielokrotnie dla różnych danych wsadowych
- Przyspieszenie czasochłonnych fragmentów kodu

Explicit parallelism



Biblioteka Rmpi

MPI (message passing interface) definiuje środowisko gdzie programy mogą działać równolegle i komunikować się ze sobą poprzez przekazywanie wiadomości między sobą. MPI jest preferowane przez użytkowników bo jest proste w obsłudze. (internet działa w podobny sposób, poprzez przesyłanie wiadomości).

Biblioteka **Rmpi** jest interfejsem do MPI, pozwalającym na pisanie programów do obliczeń równoległych w R, bez konieczności używania C, C++ czy Fortrana. Jest elastyczna i wszechstronna dla ludzi zaznajomionych z MPI i klastrami, dla pozostałych może być trochę przerażająca. Wynika to m.in. z koniecznością radzenia sobie z kilkoma problemami bezpośrednio/wprost, takimi jak:

- wysyłanie danych i funkcji do procesów podrzędnych (**slaves**)
- odpytywanie procesu nadrzędnego (**master**) o więcej zadań
- sposób komunikacji procesowi nadrzędnemu o zakończeniu pracy przez dany proces podrzędny

Przykład - Rmpi

```
library(Rmpi)

ns <- mpi.universe.size() - 1
mpi.spawn.Rslaves(nslaves = ns)

.Last <- function(){
  if (is.loaded("mpi_initialize")){
    if (mpi.comm.size(1) > 0){
      print("Please use mpi.close.Rslaves() to close slaves.")
      mpi.close.Rslaves()
    }
    print("Please use mpi.quit() to quit R")
    .Call("mpi_finalize")
  }
}
mpi.bcast.cmd( id <- mpi.comm.rank() )
mpi.bcast.cmd( ns <- mpi.comm.size() )
mpi.bcast.cmd( host <- mpi.get.processor.name() )
mpi.remote.exec(paste("I am", mpi.comm.rank(),"of", mpi.comm.size()))

x <- 5
x <- mpi.remote.exec(rnorm, x)
length(x)

mpi.close.Rslaves(dellog = FALSE)
mpi.quit()
```

Biblioteka snow

Łatwiejszym sposobem na poradzenie są bliblioteki z rodziny **snow** (snow, snowfall), które prowadzą interfejsy do kilku bibliotek umożliwiających obliczenia równoległe:

- MPI poprzez **Rmpi**
- NWS – NetWork Spaces poprzez **nws**
- PVM (Parallel Virtual Machine)
- sockets poprzez system operacyjny

Przykład - snow

```
results <- lapply(rep(250, 4), function(x) randomForest(Class ~ .,
                                                        data = GermanCredit,
                                                        ntree = x))
do.call(combine, results)


library(snow)

cl <- makeCluster(4, type = "SOCK")

clusterEvalQ(cl, {library(randomForest); NULL})
clusterExport(cl, "GermanCredit")
results <- clusterApply(cl, rep(250, 4),
                        function(x) randomForest(Class ~ .,
                                                 data = GermanCredit,
                                                 ntree = x))
do.call(combine, results)
```

Przykład - snow c.d.

```
set.seed(56283)
sleeptime <- abs(rnorm(10, 10, 10))
tm <- snow.time(clusterApply(cl, sleeptime, Sys.sleep))
plot(tm)
```

Przykład – snow c.d.

```
set.seed(56283)
sleeptime <- abs(rnorm(10, 10, 10))
tm <- snow.time(clusterApplyLB(cl, sleeptime, Sys.sleep))
plot(tm)
```

Przykład – snow c.d.

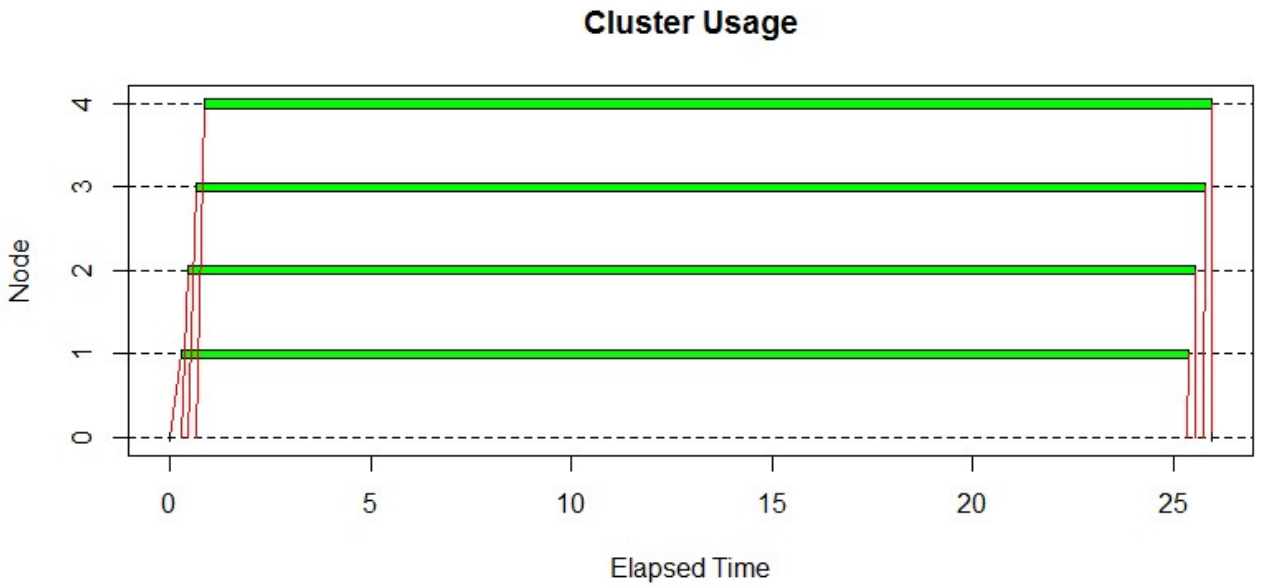
```
bigsleep <- function(sleeptime, mat)
Sys.sleep(sleeptime)
bigmatrix <- matrix(0, 2000, 2000)
sleeptime <- rep(1, 100)
```

Przykład – snow c.d.

```
tm <- snow.time(clusterApply(cl, sleeptime, bigsleep, bigmatrix))
plot(tm)
```

Przykład – snow c.d.

```
tm <- snow.time(parLapply(cl, sleeptime, bigsleep, bigmatrix))
plot(tm)
```



Biblioteka snowfall

snowfall – jest wrapperem na bibliotece **snow**, dodając na niej pewien rodzaj warstwy abstrakcyjnej, umożliwiającej łatwiejsze korzystanie.

Zalety korzystania z biblioteki **snowfall**:

- wszystkie zwrapowane funkcje posiadają możliwość obsługi błędów
- dostępność funkcji do ładowania bibliotek w klastrze
- dostępność funkcji do wymiany zmiennych między węzłami w klastrze
- zmiana ustawień klastra nie wymaga zmian w kodzie R, tylko np. z poziomu konsoli
- wszystkie funkcje działają także sekwencyjnie (a nie równolegle)
- load balancing – nie czekanie na najwolniejszą maszynie aż skończy swoje zadanie
- zapisywanie i przywracanie rezultatów osiągniętych w trakcie działania klastra

Implicit parallelism

Wyabstrahowanie problemu paralelizacji przez system.

Biblioteki:

- multicore
- foreach

Biblioteka multicore

Biblioteka **multicore** udostępnia funkcję pozwalające na obliczenia równoległe z wykorzystaniem R na systemach z wieloma rdzeniami bądź z wieloma procesorami (CPU).

Najważniejsze funkcje:

- mclapply()
- pvec()

Przykład - multicore

```
results <- lapply(rep(250, 4), function(x) randomForest(Class ~ .,
                                                        data = GermanCredit,
                                                        ntree = x))
do.call(combine, results)
```

Przykład – multicore c.d.

```
results <- mclapply(rep(250, 4), function(x) randomForest(Class ~ .,
                                                          data = GermanCredit,
                                                          ntree = x))
do.call(combine, results)
```

```
Unit: seconds
                                                                                                    expr
  results <- lapply(rep(250, 4), function(x) randomForest(Class ~      ., data = GermanCredit, ntree = x))
results <- mclapply(rep(250, 4), function(x) randomForest(Class ~      ., data = GermanCredit, ntree = x))
     min       lq     mean   median       uq      max neval
2.381816 2.393816 2.430070 2.406460 2.425078 2.713975   100
1.238276 1.256200 1.317212 1.268591 1.304051 1.790431   100
```

Przykład – multicore c.d.

```
results <- mclapply(rep(250, 4), function(x) randomForest(Class ~ .,
                                                          data = GermanCredit,
                                                          ntree = x),
                                 mc.cores = 8)
do.call(combine, results)
```

Przykład – multicore c.d.

```
results <- mclapply(rep(250, 4), function(x) randomForest(Class ~ .,
                                                          data = GermanCredit,
                                                          ntree = x),
                                 mc.cores = 8,
                                 mc.preschedule = FALSE)
do.call(combine, results)
```

Biblioteka foreach

foreach pozwala użytkownikowi na iteracje przez zbiór wartości równolegle. Domyślnie działa sekwencyjnie, dopóki nie użyjemy biblioteki jak **doMC** która jest backendem do zrównoleglenia. **doMC** jest jakby interfejsem pomiędzy **foreach** a **multicore**.

foreach może być używane na klastrze z użyciem różnych backendów do zrównoleglania. **doMC** używa **multicore**, a **doNWS** i **doSNOW** używają kolejno **nws** i **snow**.

Przykład - foreach

```
library(foreach)
foreach(ntree = rep(250, 4), .combine = combine) %do%
        randomForest(Class ~ ., data = GermanCredit, ntree = ntree)
```

```
library(doSnow)

cl <- makeCluster(4, type = "SOCK")
registerDoSNOW(cl)

foreach(ntree = rep(250, 4),
          .combine = combine,
          .packages = "randomForest") %dopar%
  randomForest(Class ~ ., data = GermanCredit, ntree = ntree)
```

Biblioteka parallel

Biblioteka łącząca funkcjonalności i funkcje dostępne w **multicore** oraz **snow**

Przykład – parallel + caret

```
library(parallel)
library(doParallel)
library(caret)

data(GermanCredit)
nodes <- c(rep('moonshot27', 4),
           rep('moonshot28', 4))
cl <- makePSOCKcluster(nodes,
                       rshcmd = 'plink',
                       user = 'r',
                       rscript = '/usr/bin/Rscript',
                       outfile = '')
registerDoParallel(cl)
rf_model <- train(Class ~ ., data = GermanCredit,
                          method = 'rf',
                          trControl = trainControl(method = 'cv',
                                                   number = 50,
                                                   allowParallel = TRUE))
stopCluster(cl)
```