New Possibilities with SparkR

Big Data without leaving R

Vincent D. Warmerdam @ GoDataDriven

Who is this guy

- Vincent D. Warmerdam
- data guy @ GoDataDriven
- from Amsterdam
- avid python, R and js user.
- give open sessions in R/Python
- minor user of scala, julia, clang.
- hobbyist gamer. Blizzard fanboy.
- in **no way** affiliated with Blizzard.

Today

1. Describe a cool, but big, data task
2. Explain downsides of 'normal' R
3. Explain idea behind SparkR
4. Explain how to get started with SparkR
5. Show some SparkR code
6. Quick Conclusion
7. if(time) Demo
8. if(time) Questions

TL;DR

Spark is a very worthwhile tool that is opening up for R.

If you just know R, it feels to be a preferable way to do big data in the cloud. It performs, scales and feels like writing code in normal R, although the api is limited.

This project has gained enormous traction, is being used in many impressive production systems and you can expect more features in the future.

1. The task and data

We're going to analyze a video game

World of Warcraft Auction House

Items of Warcraft

Items/gear are an important part of the game. You can collect raw materials and make gear from it. Another alternative is to sell it.

- you can collect virtual goods
- you trade with virtual gold
- to buy cooler virtual swag
- to get better, faster, stronger
- collect better virtual goods

WoW data is cool!

- now about 10 million of players
- 100+ identical wow instances (servers)
- real world economic assumptions still hold
- perfect measurement that you don't have in real life
- each server is an identical
- these worlds are independant of eachother

It is cool, it also has a problem.

The Blizzard API gave me snapshots every two hours of the current auction house status.

One such snapshot is a 2 GB blob op json data.

After a few days the dataset does not fit in memory. Even one snapshot is something a single threaded process doesn't like.

R seems to fall short here.

2. The technical problem

This problem occurs often

This is a BIG DATA problem

'When your data is too big to analyze on a single computer.'
- Ian Wrigley, Cloudera

What do you do when you want to blow up a building?

Use a bomb.

What do you do when you want to blow up a building?

Use a bomb.

What do you do when you want to blow up a bigger building?

Use a bigger, way more expensive, bomb

What do you do when you want to blow up a building?

Use a bomb.

What do you do when you want to blow up a bigger building?

~~Use a bigger, way more expensive, bomb~~

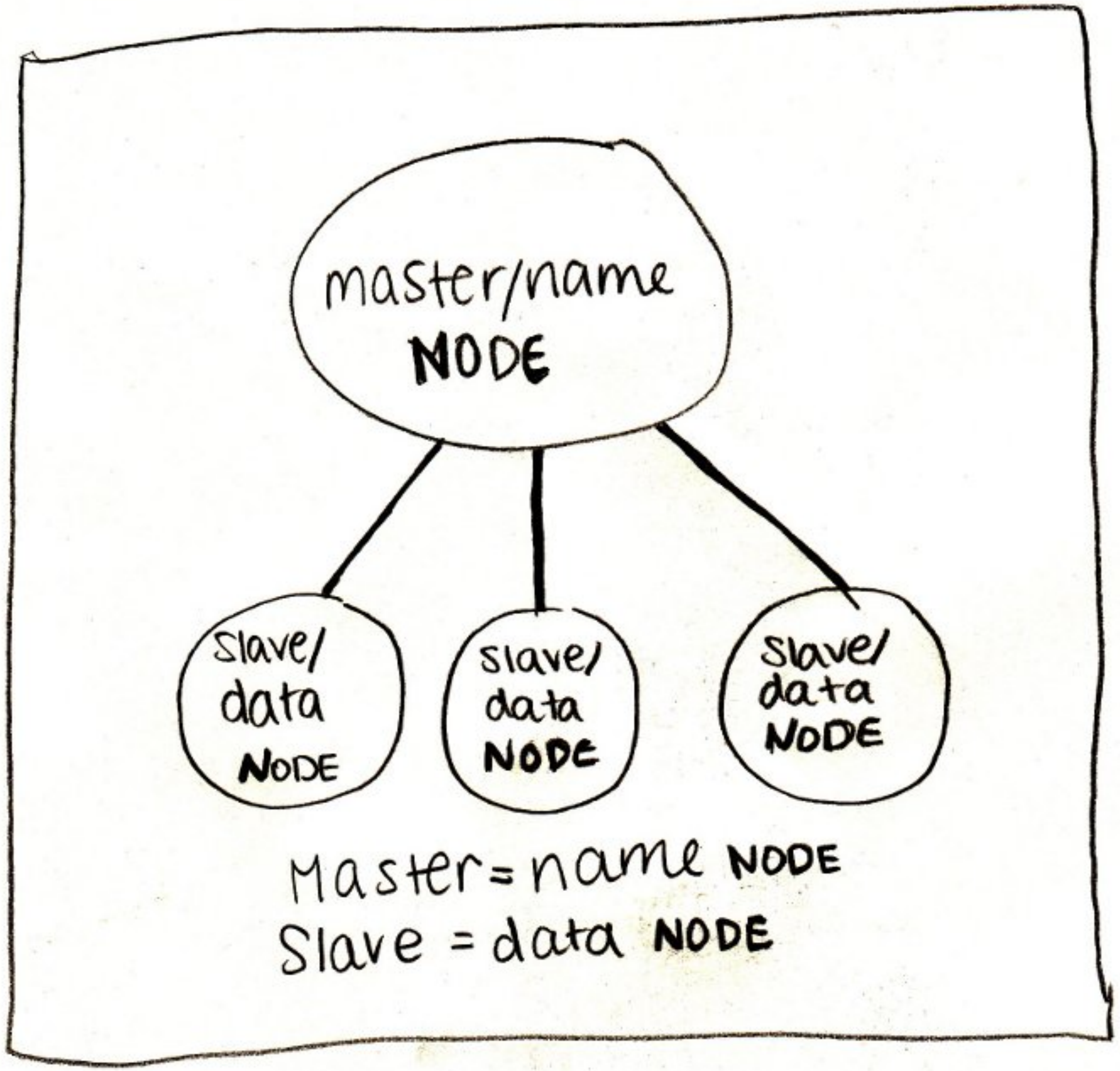
Use many small ones.

3. The technical problem

Take the many small bombs approach

Distributed disk (Hadoop/Hdfs)

- connect machines
- store the data on multiple disks
- compute map-reduce jobs in parallel
- bring code to data
- not the other way around
- old school: write map reduce jobs

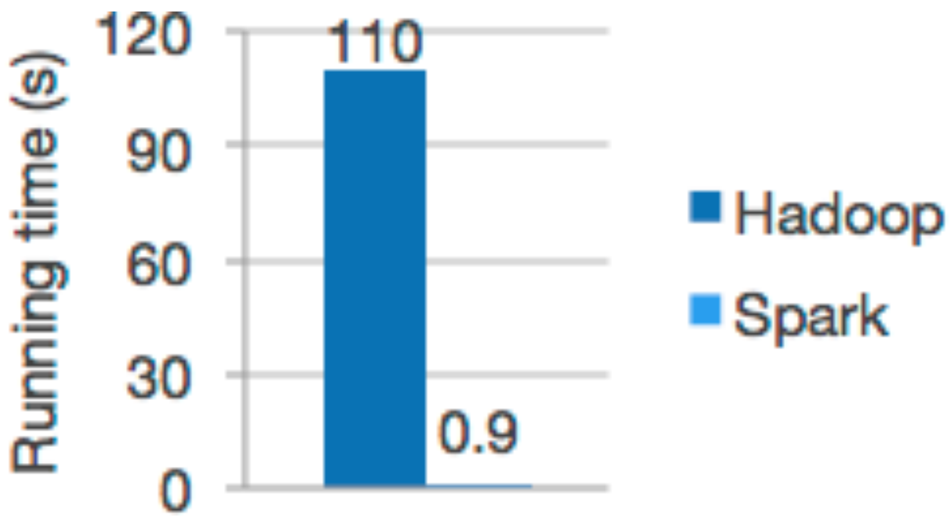


Why Spark?

"It's like MapReduce on Hadoop but preferable."

Why Spark?

"Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk."



Attemps to do all computation in memory.
Can cache to disk if needed.

Spark is parallel

Even locally

```
Processes: 228 total, 3 running, 3 stuck, 222 sleeping, 1345 threads          21:04:
Load Avg: 3.24, 2.29, 1.87  CPU usage: 96.94% user, 2.76% sys, 0.29% idle
SharedLibs: 90M resident, 0B data, 14M linkedit.
MemRegions: 83992 total, 7019M resident, 76M private, 13G shared.
PhysMem: 13G used (2546M wired), 632M unused.
VM: 608G vsize, 1312M framework vsize, 3013284(0) swapins, 3316559(0) swapouts.
Networks: packets: 29603472/34G in, 11073080/2276M out.
Disks: 3185216/85G read, 3042468/109G written.

PID    COMMAND      %CPU     TIME     #TH    #WQ  #PORT #MREGS MEM    RPRVT  PURG   CMPRS
48026  java         775.5    11:21.01 95/8   0    236-  2339   941M-  947M-  0B     138M
36104  top          18.9     42:47.01 1/1    0    45    56     7904K  7748K  0B     172K
118    WindowServer 2.4      02:45:02 4      0    732   6561-  581M-  120M-  29M    242M
```

Under the hood; why shift Hadoop -> Spark

- it doesn't persist full dataset to HDFS
- distributed in memory -> no disk io
- lazy eval and the DAG
- relatively easy simple to code in
- DataFrame/ML/Graph/Streaming support

4. How to set up Spark

It's not that hard

Spark Provisioning: Locally

Download Spark here. Unzip. Then:

```
$ /path/to/spark-1.5.1/bin/sparkR
```

You can set some flags if you want to have more power.

```
$ ./sparkR --driver-memory 5g
```

Spark Provisioning: Locally

Running it in Rstudio is only a little more work.

First configure syspaths.

```
spark_link <- "spark://codes-MacBook-Pro.local:7077"
spark_path <- "/Users/code/Downloads/spark-1.5.0-bin-hadoop2.6"
spark_lib_path <- paste0(spark_path, '/R/lib')
spark_bin_path <- paste0(spark_path, '/bin')

.libPaths(c(.libPaths(), spark_lib_path))
Sys.setenv(SPARK_HOME = spark_path)
Sys.setenv(PATH = paste(Sys.getenv(c('PATH')), spark_bin_path, sep=':'))
```

Spark Provisioning

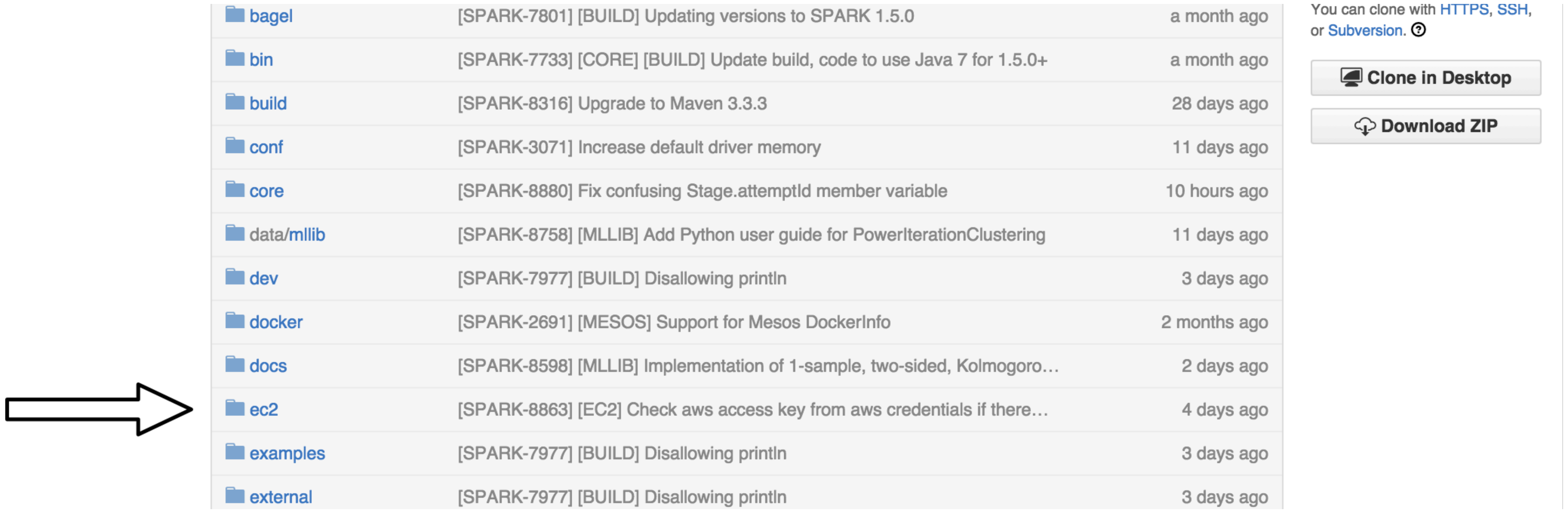
Running it in Rstudio is only a little more work.

Next, just import libraries.

```
library(SparkR)
library(ggplot2)
library(magrittr)

sc <- sparkR.init("local[2]", "SparkR", spark_path,
                  list(spark.executor.memory="8g"))
sqlContext <- sparkRSQL.init(sc)
```

What about if I have a huge dataset?

You could go for Databricks, or you could set up your own on AWS. Other platforms also have offerings but AWS support comes with Spark.



Spark Provisioning

On AWS it's just is a one-liner.

```
./spark-ec2 \
--key-pair=pems \
--identity-file=/path/pems.pem \
--region=eu-west-1 \
-s 8 \
--instance-type c3.xlarge \
--copy-aws-credentials
launch my-spark-cluster
```

This starts up the whole cluster, takes 10-20 mins.

Spark Provisioning

If you want to turn it off.

```
./spark-ec2 \
--key-pair=pems \
--identity-file=/path/pems.pem \
--region=eu-west-1 \
 destroy my-spark-cluster
```

This brings it all back down, warning: potentially deletes data.

Spark Provisioning

If you want to log into your machine.

```
./spark-ec2 \
--key-pair=pems \
--identity-file=/path/pems.pem \
--region=eu-west-1 \
 login my-spark-cluster
```

It does the ssh for you.

Reading from S3

Reading in `.json` file from amazon.

```
# no need for credentials with --copy-aws-credentials
filepath <- "s3n://<aws_key>:<aws_secret>@wow-dump/total.json"

ddf <- sqlContext %>%
  textFile(filepath, 'json') %>%
  cache()
```

These credentials can be automatically retreived if boot was via `--copy-aws-credentials`.

5. Writing SparkR

Feels like R code

The `ddf` is designed to feel like normal R.

```
ddf$date <- ddf$timestamp %>% substr(1, 10)
```

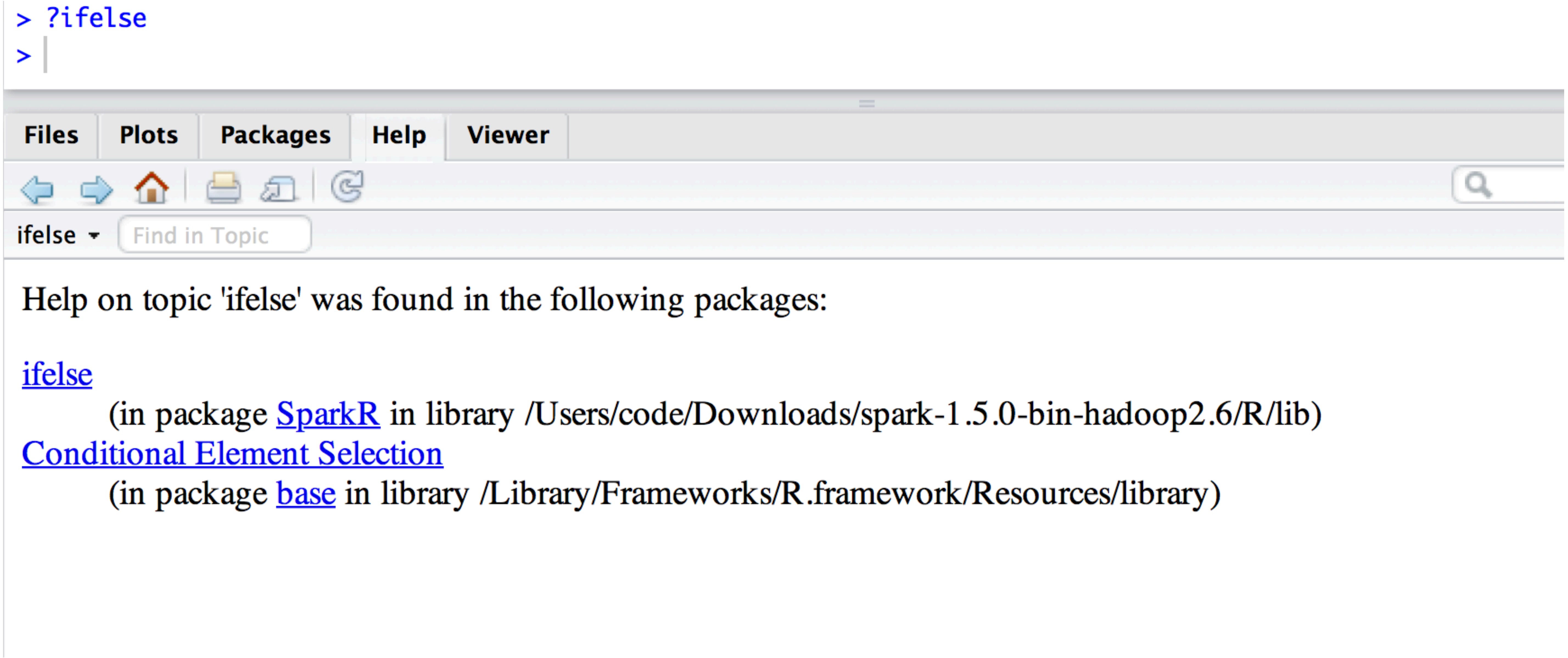
If you use Rstudio, you'll notice that autocomplete works for distributed dataframes as well.

Lost of R functions

Many SparkR functions work like normal R functions but on distributed DataFrames. Not everything is supported but currently there is support for:

```
%in%
ifelse
regex
datetimes
levenshtein
glm
```

Different functions?



Find most frequent wow items

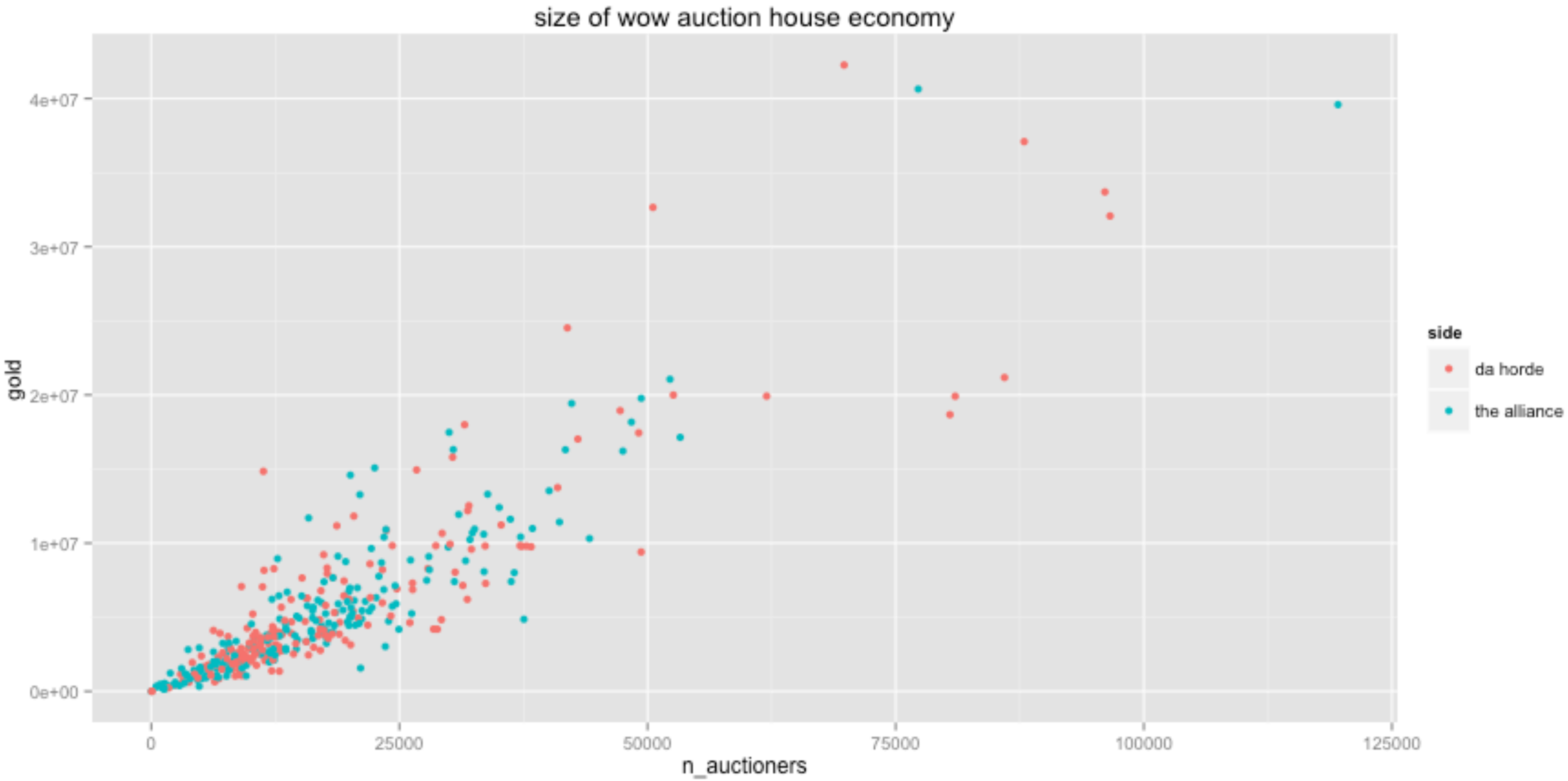
SparkR comes with `dplyr`-like functions.

```
agg <- ddf %>%
  groupBy(ddf$item) %>%
  summarize(count = n(ddf$item)) %>%
  collect

freq_df <- agg[order(-agg$count),] %>% head(30)
freq_items <- freq_df$item
```

Note that agg is a normal (nondist) dataframe.

Auctioneers vs. economy

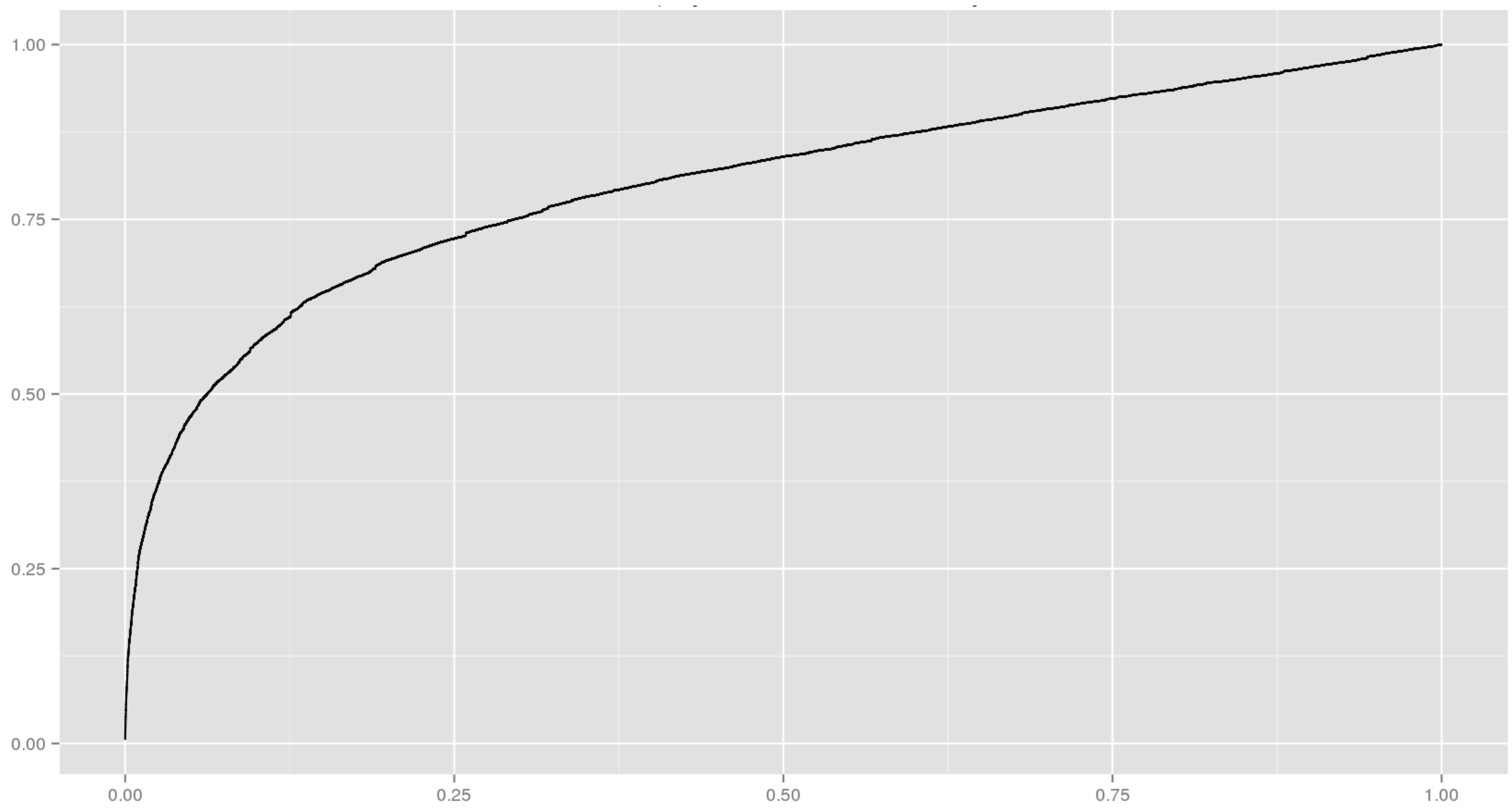
```
agg <- ddf %>%
  groupBy(ddf$ownerRealm, ddf$side) %>%
  summarize(n_auctioners = n(ddf$ownerRealm),
            gold = sum(ddf$buyout)/10000) %>%
  collect

agg$side <- ifelse(agg$side == 'alliance',
                   'the alliance', 'da horde')

ggplot(data=agg) +
  geom_point(aes(n_auctioners, gold, colour=side)) +
  ggtitle('size of wow auction house economy')
```


size of wow auction house economy
gold
4e+07
3e+07
2e+07
1e+07
0e+00
0
25000
50000
75000
100000
125000
n_auctioners
side
da horde
the alliance

The 1% of WOW

```
pltr <- ddf %>%
  filter(ddf$owner != '???') %>%
  group_by(ddf$owner) %>%
  summarize(n = countDistinct(ddf$auc), s = sum(ddf$buyout)) %>%
  arrange(desc(.$n)) %>%
  collect

pltr$cum <- cumsum(pltr$s)/sum(pltr$s)
pltr$per <- 1:nrow(pltr)/nrow(pltr)
```

The 1% of WOW



Local Benchmarks

I have an 8-core mac; spark notices this.

```
> start_time <- Sys.time()
> ddf <- sqlContext %>%
    loadDF('/Users/code/Development/wow-data/complete-day.json', 'json') %>%
    cache
> ddf %>% count
[1] 7250322
> Sys.time() - start_time
Time difference of 1.103298 mins
```

This is a 2 GB file. Pretty fast for local development.

Local Benchmarks

Spark also has a caching system.

```
> start_time <- Sys.time()
> ddf %>% count
[1] 7250322
> Sys.time() - start_time
Time difference of 0.44373053 secs
```

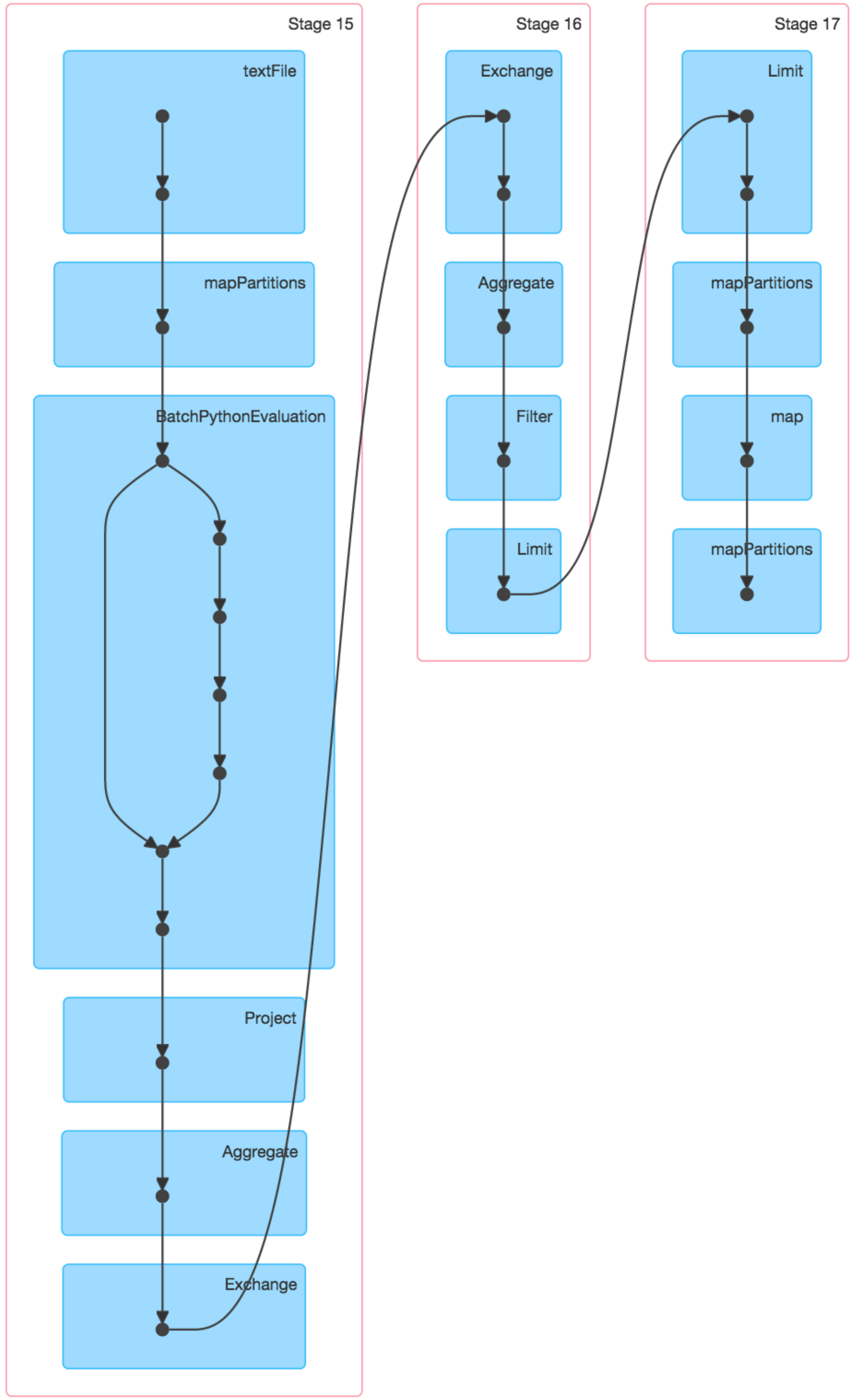
The second time the operation runs faster because of it.

Visualisation of the DAG

You can view the DAG in Spark UI.

The job on the right describes an aggregation task.

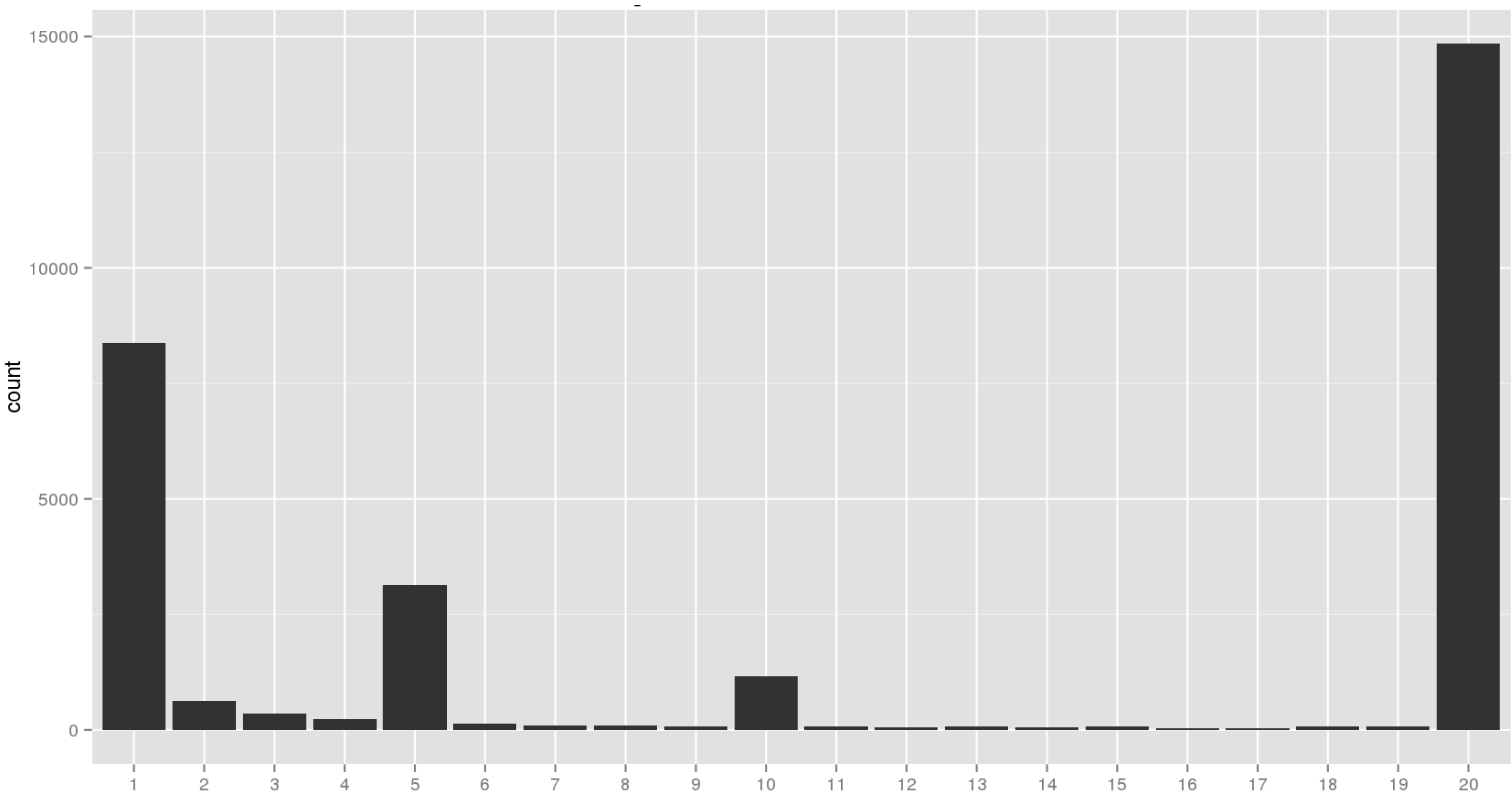
You can find this at `master-ip:4040`.



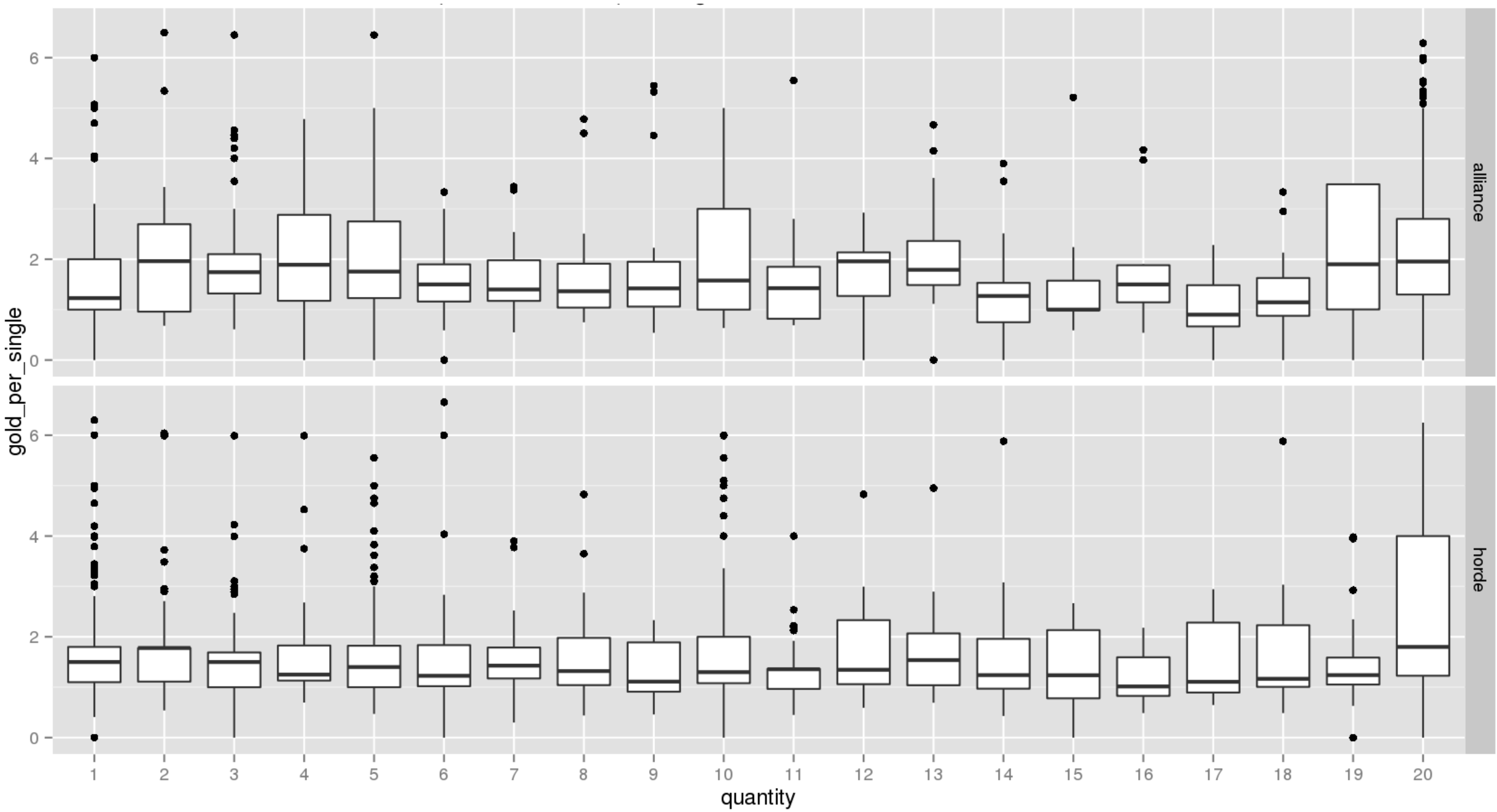
Crunch in Spark, analyse in R

```
ddf$gold_per_single <- ddf$buyout/ddf$quantity/10000

pltr <- ddf %>%
  filter(ddf$side != 'neutral') %>%
  filter(ddf$item == freq_items[5]) %>%
  collect

pltr$quantity <- pltr$quantity %>% as.factor

pltr <- subset(pltr,
  pltr$gold_per_single < quantile(pltr$gold_per_single, probs = 0.95)
)
```

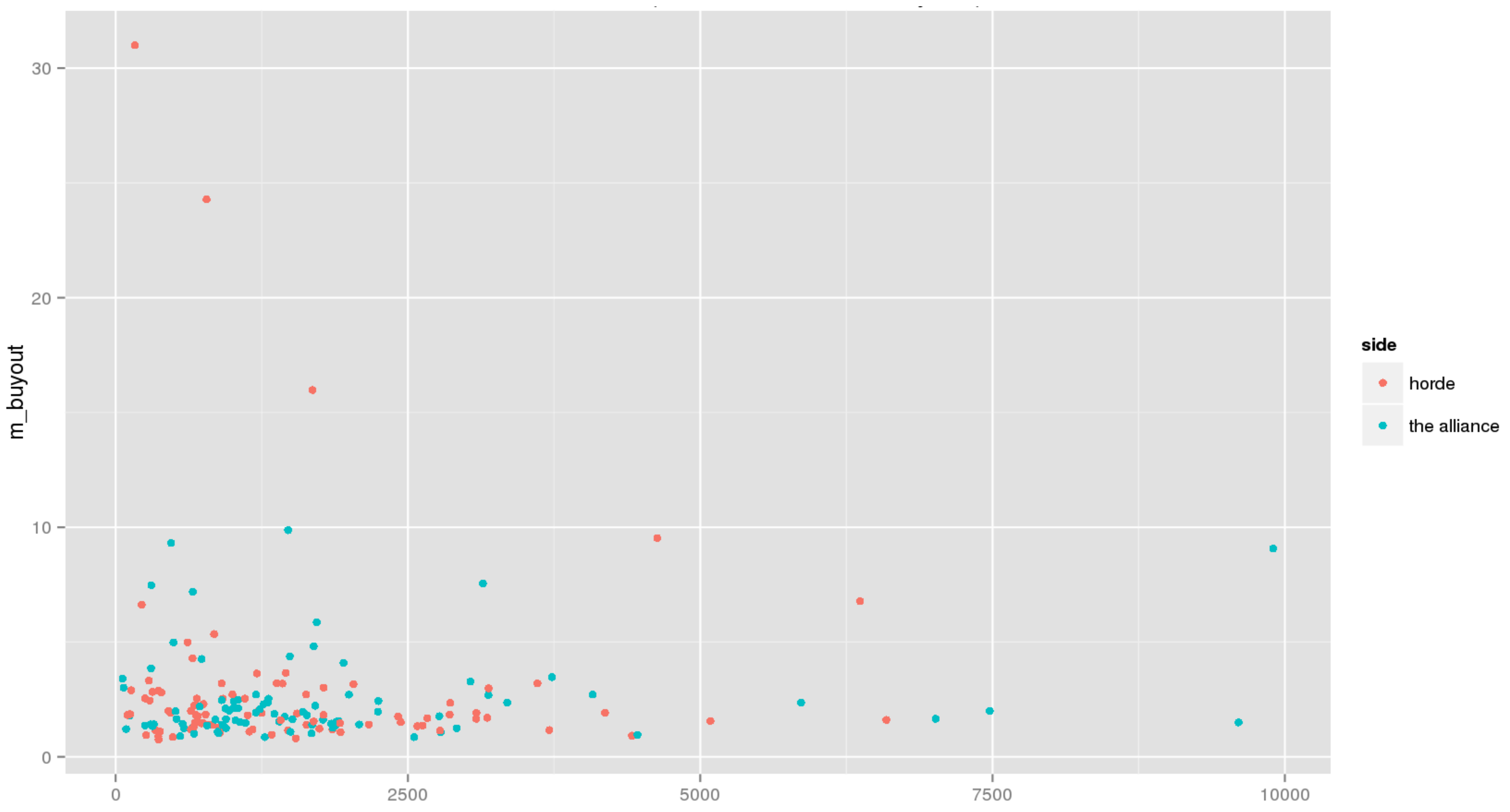
effect of stack size, spirit dust



effect of stack size, spirit dust



Market size vs price[1]



[1] for spirit dust we check for every server what the market quantity is and the mean buyout

Market size vs. price

We repeat for every product by calculating it's β_1 regression coefficient:

$$\beta_1 = \frac{Cov(x, y)}{Var(x)}$$

where x is market size and y is price. If $\beta_1 < 0$ then we may have found a product that is sensitive to market production.

GLM in Spark

```
freq_items <- ddf %>%
  groupBy(ddf$item) %>%
  summarize(count = n(ddf$item)) %>%
  orderBy(-.$count) %>%
  select(ddf$item) %>%
  head(100)

ml_ddf <- ddf %>%
  filter(ddf$item %in% freq_items$item) %>%
  group_by(ddf$item, ddf$side, ddf$ownerRealm) %>%
  summarize(n = sum(ddf$quantity), p = mean(ddf$buyout/ddf$quantity/10000))

d_mod <- glm(p ~ n, data = ml_ddf)
```

GLM in Spark

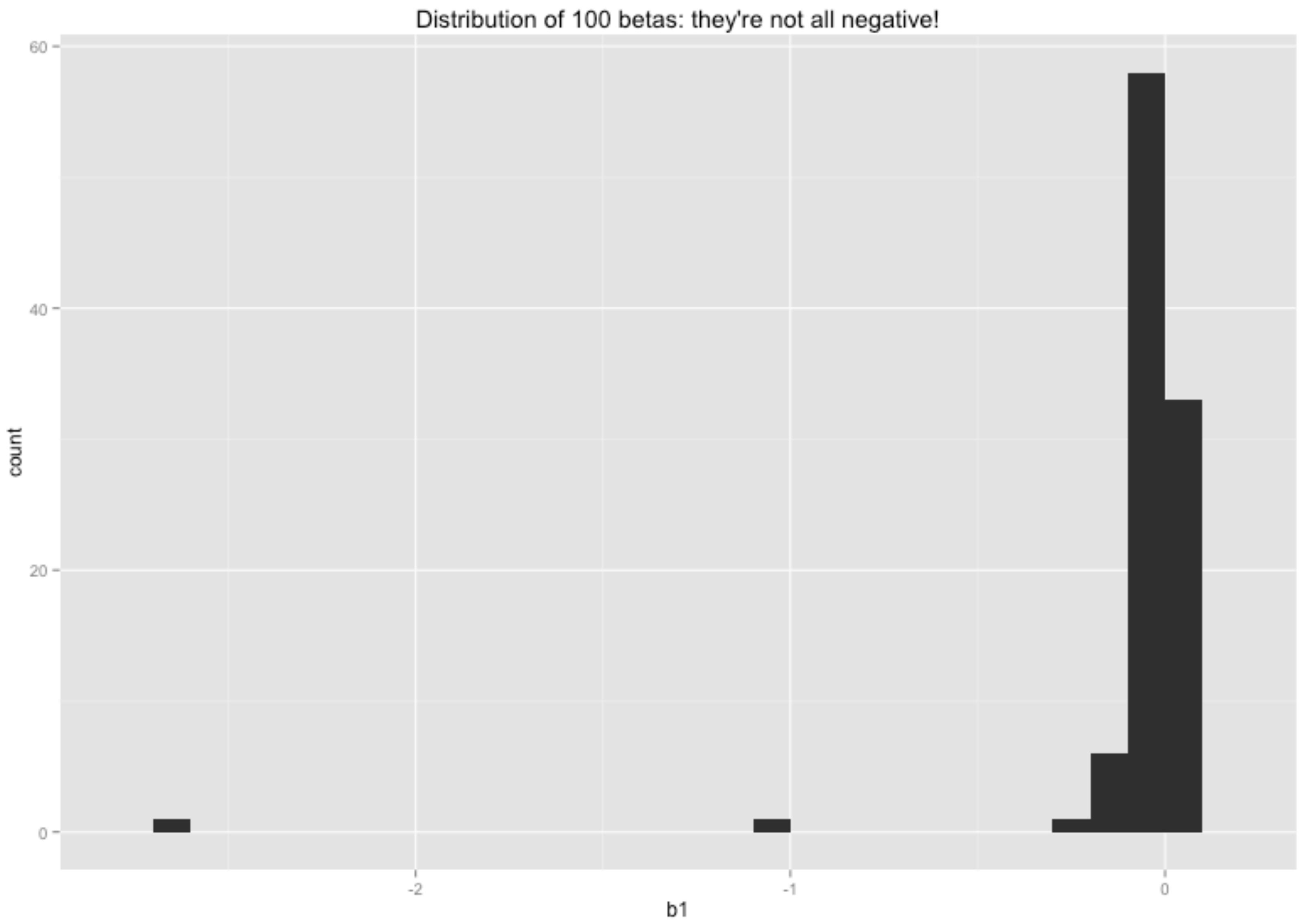
```
> d_mod %>% summary
$coefficients
                 Estimate
(Intercept) 78.08618816
n           -0.01784264
```

This result makes sense; but is not that interesting. I miss `dplyr::do` here.

A most common pattern

```
ml_df <- ml_ddf %>% collect

SparkR.stop()
detach("package:SparkR", unload=TRUE)

library(dplyr)
res <- ml_df %>%
  group_by(item) %>%
  do(mod = lm(p ~ n, data = .) %>% coefficients %>% .[2]) %>%
  mutate(b1 = mod %>% as.numeric)
```

Most interesting result



Conclusion

OK

But clusters cost more, correct?

Cheap = Profit

Isn't Big Data super expensive?

Cheap = Profit

Isn't Big Data super expensive?

Actually, no

Cheap = Profit

Isn't Big Data super expensive?

Actually, no

S3 transfers within same region = free.
40 GB x $0.03 per month = $1.2

```
$0.239 x hours x num_machines
```

If I use this cluster for a day.

```
$0.239 x 6 x 9 = $12.90
```

Conclustion

Spark is worthwhile tool.

If datasets become bigger this tool helps to keep the exploration feel interactive, which has always felt is the most powerful part of R/Rstudio.

Final Remarks

- don't forget to turn machines off
- please beware the inevitable hype
- only bother if your dataset is too big
- dplyr/tidyr/baseR has more flexible (better) api
- more features to come
- more features are needed

Demo

Questions?

The images

Some images from my presentation are from the nounproject.

Credit where credit is due;

- video game controller by Ryan Beck
- inspection by Creative Stall
- Shirt Size XL by José Manuel de Laá

Other content online:

- epic orc/human fight image

Demo Code

```
./spark-ec2
--key-pair=spark-df
--identity-file=//Users/code/Downloads/spark-df.pem
--region=eu-west-1 -s 2
--instance-type c3.xlarge
--copy-aws-credentials launch my-spark-cluster

./spark-ec2
--key-pair=spark-df
--identity-file=//Users/code/Downloads/spark-df.pem
--region=eu-west-1 -s 2
--copy-aws-credentials login my-spark-cluster

curl icanhazip.com
passwd rstudio
```

```
vars <- tail(read.csv('/root/spark-ec2/ec2-variables.sh'), 2)
colnames(vars) <- 'a'
vars$a <- as.character(vars$a)
for(i in gsub("export ", "", vars$a)){
  eval(parse( text = paste0(gsub("=", "='", i), "'") ))
}

filepath <- paste0("s3n://",
                   AWS_ACCESS_KEY_ID, ":",
                   AWS_SECRET_ACCESS_KEY,
                   "@wow-dump/chickweight.json")
ddf <- loadDF(sqlContext, filepath, 'json')

ddf
head(ddf)
collect(summarize(m = mean(ddf$weight), group_by(ddf ,ddf$Diet)))
```

```
./spark-ec2
--key-pair=spark-df
--identity-file=//Users/code/Downloads/spark-df.pem
--region=eu-west-1 -s 2
--copy-aws-credentials destroy my-spark-cluster
```