PySpark of Warcraft
understanding video games better through data

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.
- hobbyist gamer. Blizzard fanboy.
- in no way affiliated with Blizzard.
Today

1. Description of the task and data
2. Description of the big technical problem
3. Explain why Spark is good solution
4. Explain how to set up a Spark cluster
5. Show some PySpark code
6. Share some conclusions of Warcraft
7. Conclusion + Questions
8. If time: demo!
Spark is a very worthwhile, open tool.

If you just know python, it's a preferable way to do big data in the cloud. It performs, scales and plays well with the current python data science stack, although the api is a bit limited.

This project has gained enormous traction, so you can expect more in the future.
1. The task and data

For those that haven't heard about it yet
The Game of Warcraft

- you keep getting stronger
- fight stronger monsters
- get stronger equipment
- fight stronger monsters
- you keep getting stronger
- repeat ...
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
World of Warcraft Auction House
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
Wow Auction House Data

For every auction we have:

- the product id (which is traceable to actual product)
- the current bid/buyout price
- the amount of the product
- the owner of the product
- the server of the product

See api description.
Sort of questions you can answer?

• Do basic economic laws make sense?
• Is there such a thing as an equilibrium price?
• Is there a relationship between production and price?

This is very interesting because...

• It is very hard to do something like this in real life.
How much data is it?

The Blizzard API gives you 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.
What to do?

It is not trivial to explore this dataset.

This dataset is too big to just throw in excel.

Even pandas will have trouble with it.
Possible approach

Often you can solve a problem by avoiding it.

- use a better fileformat (csv instead of json)
- hdf5 where applicable

This might help, but this approach does not scale.
The scale of this problem seems too big.
2. The technical problem

This problem occurs more often
This is a BIG DATA problem

What is a big data problem?
'Whenever 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 Hadoop but it tries to do computation in memory."
Why Spark?

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

It does performance optimization for you.
Spark is parallel

Even locally

Processes: 228 total, 3 running, 3 stuck, 222 sleeping, 1345 threads
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.

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

The api just makes functional sense.

Word count:

text_file = spark.textFile("hdfs://...")

text_file.flatMap(lambda line: line.split())
  .map(lambda word: (word, 1))
  .reduceByKey(lambda a, b: a+b)
Nice Spark features

• super fast because distributed memory (not disk)
• it scales linearly, like hadoop
• good python bindings
• support for SQL/Dataframes
• plays well with others (mesos, hadoop, s3, cassandra)
More Spark features!

- has parallel machine learning libs
- has micro batching for streaming purposes
- can work on top of Hadoop
- optimizes workflow through DAG operations
- provisioning on aws is pretty automatic
- multilanguage support (R, scala, python)
4. How to set up a Spark cluster

Don't fear the one-liner
Spark Provisioning

You could go for Databricks, or you could set up your own.
**Spark Provisioning**

Starting is a one-liner.

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

This starts up the whole cluster, takes about 10 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: 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.
from pyspark import SparkContext
from pyspark.sql import SQLContext, Row

CLUSTER_URL = "spark://<master_ip>:7077"
sc = SparkContext(CLUSTER_URL, 'ipython-notebook')
sqlContext = SQLContext(sc)
Reading from S3

Reading in .json file from amazon.

```python
filepath = "s3n://<aws_key>:<aws_secret>@wow-dump/total.json"

data = sc\n    .textFile(filepath, 30)\n    .cache()
```
Reading from S3

filepath = "s3n://<aws_key>:<aws_secret>@wow-dump/total.json"

data = sc\n    .textFile(filepath, 30)\n    .cache()

data.count() # 4.0 mins
data.count() # 1.5 mins

The persist method causes caching. Note the speed increase.
Reading from S3

data = sc\
    .textFile("s3n://<aws_key>:<aws_secret>@wow-dump/total.json", 200)\
    .cache()

data.count() # 4.0 mins
data.count() # 1.5 mins

Note that code doesn't get run until the .count() command is run.
More better: textfile to DataFrame!

df_rdd = data\
  .map(lambda x : dict(eval(x))))\
  .map(lambda x : Row(realms=x['realm'], side=x['side'],
                      buyout=x['buyout'], item=x['item']))

df = sqlContext.inferSchema(df_rdd).cache()

This dataframe is distributed!
5. Simple PySpark queries

It's similar to Pandas
Basic queries

The next few slides contain questions, queries, output, loading times to give an impression of performance.

All these commands are run on a simple AWS cluster with 8 slave nodes with 7.5 RAM each.

Total .json file that we query is 20 GB. All queries ran in a time that is acceptable for exploratory purposes. It feels like pandas, but has a different api.
DF queries

economy size per server

df\n    .groupBy("realm")\n    .agg({"buyout":"sum"})\n    .toPandas()

You can cast to pandas for plotting
DF queries

offset price vs. market production

def.filter("item = 21877")
    .groupBy("realm")
    .agg({"buyout":"mean", "*":"count"})
    .show(10)
import pyspark.sql.functions as func

items_ddf = ddf.groupBy('ownerRealm', 'item')
    .agg(func.sum('quantity').alias('market'),
         func.mean('buyout').alias('m_buyout'),
         func.count('auc').alias('n'))
    .filter('n > 1')

# now to cause data crunching
items_ddf.head(5)
DF queries

visualisation of the DAG

You can view the DAG in Spark UI.
The job on the right describes the previous task.
You can find this at master-ip:4040.
DF queries

new column via user defined functions

# add new column with UDF
to_gold = UserDefinedFunction(lambda x: x/10000, DoubleType())
ddf = ddf.withColumn('buyout_gold', to_gold()('buyout'))
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

$0.239 \times \text{hours} \times \text{num\_machines}$

If I use this cluster for a day.

$0.239 \times 6 \times 9 = \$12.90$
6. Results of Warcraft Data, for the horde!
## Most popular items

<table>
<thead>
<tr>
<th>item</th>
<th>count</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>82800</td>
<td>2428044</td>
<td>pet-cage</td>
</tr>
<tr>
<td>21877</td>
<td>950374</td>
<td>netherweave-cloth</td>
</tr>
<tr>
<td>72092</td>
<td>871572</td>
<td>ghost-iron-ore</td>
</tr>
<tr>
<td>72988</td>
<td>830234</td>
<td>windwool-cloth</td>
</tr>
<tr>
<td>72238</td>
<td>648028</td>
<td>golden-lotus</td>
</tr>
<tr>
<td>4338</td>
<td>642963</td>
<td>mageweave-cloth</td>
</tr>
<tr>
<td>21841</td>
<td>638943</td>
<td>netherweave-bag</td>
</tr>
<tr>
<td>74249</td>
<td>631318</td>
<td>spirit-dust</td>
</tr>
<tr>
<td>72120</td>
<td>583234</td>
<td>exotic-leather</td>
</tr>
<tr>
<td>72096</td>
<td>578362</td>
<td>ghost-iron-bar</td>
</tr>
<tr>
<td>33470</td>
<td>563214</td>
<td>frostweave-cloth</td>
</tr>
<tr>
<td>14047</td>
<td>534130</td>
<td>runecloth</td>
</tr>
<tr>
<td>72095</td>
<td>462012</td>
<td>trillium-bar</td>
</tr>
<tr>
<td>72234</td>
<td>447406</td>
<td>green-tea-leaf</td>
</tr>
<tr>
<td>53010</td>
<td>443120</td>
<td>embersilk-cloth</td>
</tr>
</tbody>
</table>
what profession?

based on level 10-20 items

<table>
<thead>
<tr>
<th>type</th>
<th>m_gold</th>
</tr>
</thead>
<tbody>
<tr>
<td>skinning</td>
<td>2.640968</td>
</tr>
<tr>
<td>herbalism</td>
<td>2.316380</td>
</tr>
<tr>
<td>mining</td>
<td>1.586510</td>
</tr>
</tbody>
</table>

Seems like in the beginning skinning makes the most money. Note these values are aggregates, this number can also be calculated per server for end game items for relevance.
the one percent
effect of stack size, spirit dust
effect of stack size, spirit dust
effect of stack size, spirit dust
market size vs price

for spirit dust we check for every server that the market quantity is and the mean buyout
market size vs price

We repeat for every product by calculating its $\beta_1$ 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.
slightly shocking find

Turns out that most of these products have $\beta_1 \approx 0$.

What does this mean? Are our economical laws flawed?
Conclusion

Spark is worthwhile tool.

There's way more things supported:

• machine learning
• graph analysis tools
• real time tools
Conclusion

Spark is worthwhile tool.

Final hints:

- don't forget to turn machines off
- this setup is not meant for multi users
- only bother if your dataset is too big, scikit/pandas has more flexible api
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
/r/pokemon/
Feedback:

• pokemon fans did **not** agree that my model was correct
• pokemon fans did agree that my models output made sense

Why this matters:

• pokemon is relatively complicated