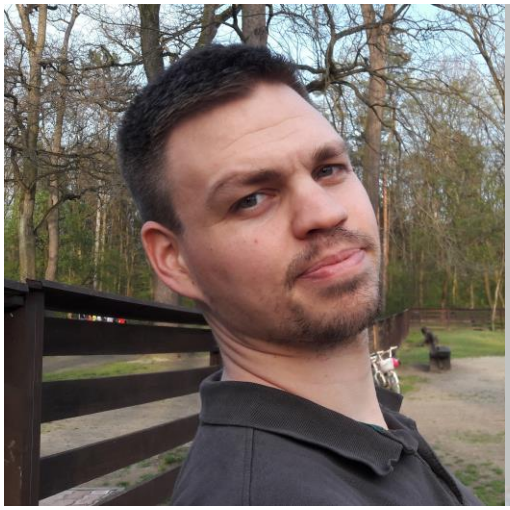



Dr. Hegedűs Péter, Dr. Ferenc Rudolf

Nagyméretű adatbázisok

Jelen tananyag a Szegedi Tudományegyetemen készült az Európai Unió támogatásával.

Projekt azonosító: EFOP-3.4.3-16-2016-00014

Szegedi Tudományegyetem
Cím: 6720 Szeged, Dugonics tér 13.
www.u-szeged.hu
www.szechenyi2020.hu

Hadoop Streaming

Összefoglalás

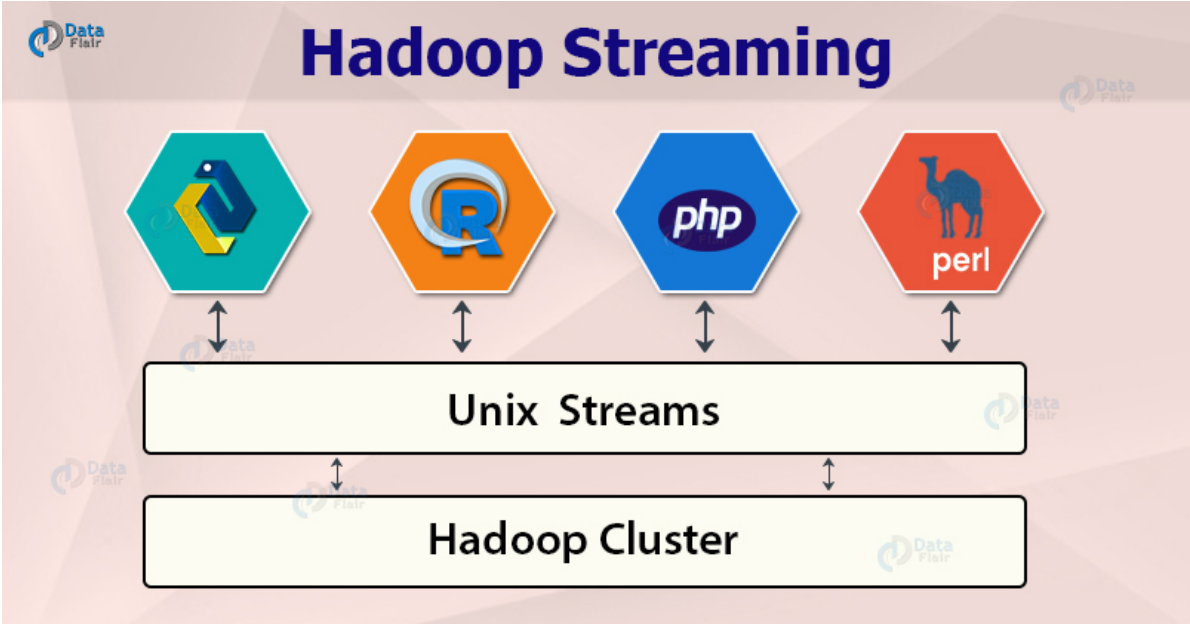
Az olvasó ebből az olvasóleckéből megismerkedhet az Apache Hadoop Streaming eszközzel. Egyrészt bemutatjuk a streaming alapvető működési elvét, valamint azt, hogy hogyan illeszkedik ez bele a teljes Hadoop ökoszisztémába. A streaming működési elve mellett gyakorlati példákon keresztül ismerkedhet meg az olvasó azzal, hogyan kell használni a streaming eszközt ahhoz, hogy a klasszikus Java MapReduce programok helyett egyszerű futtatható állományokkal vagy szkriptekkel valósítsa meg a map és reduce függvényeket, valamint futtassa le ezt job-ként a Hadoop klaszteren.
A lecke fejezetei:

- 1. fejezet: **Az Apache Hadoop Streaming bemutatása, alapvető működési elve, bevezető példa (olvasó)**
- 2. fejezet: **A klasszikus word count probléma megoldása Python szkript segítségével, streaming-en futtatva (olvasó)**
- 3. fejezet: **Árfolyam átlag számítási példa megoldása Python szkript segítségével, streaming-en futtatva (olvasó)**

Téma típusa: **gyakorlati**
Olvasási idő: **45 perc**

📖 1. fejezet
A Hadoop Streaming eszköz

A Hadoop streaming egy eszköz, amelyet az alap Hadoop disztribúció tartalmaz. Segítségével tetszőleges futtatható állomány vagy szkript használható mint map és reduce megvalósítás egy MapReduce job-hoz. Ezt nagyon egyszerűen a Unix Stream-ek használatával éri el, amely interfészt képez a Hadoop klaszter és a map/reduce megvalósítások futtatható állományai között. Mind a mapper mind a reducer a standard bemenetről olvassa be az adatokat, és a standard kimenetre írja azokat. A beolvasás soronként történik. Ezáltal a map/reduce megvalósítás tetszőleges futtatható állomány lehet, ami a standard bemenetet olvassa és az eredményét a standard kimenetre írja. Továbbá a Hadoop Streaming több szkript nyelvet is támogat map/reduce megvalósításhoz: Python, Perl, R, vagy PHP.



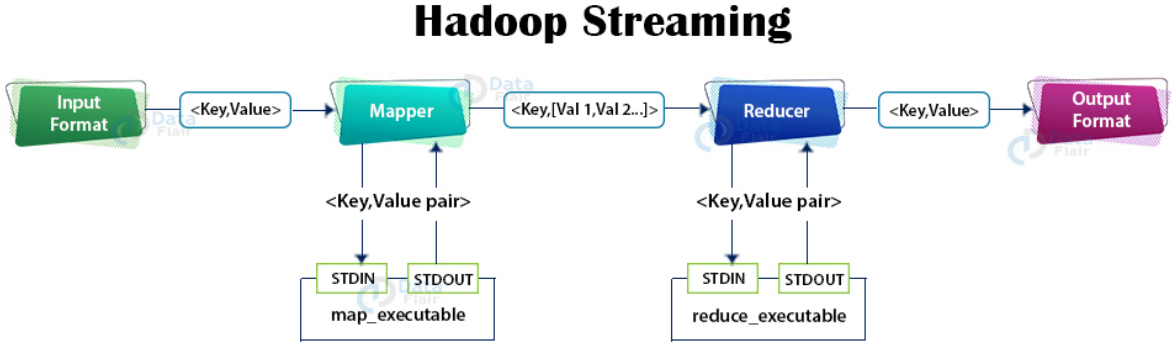
> https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/sites/2/2020/03/hadoop-streaming.jpg

Egy nagyon egyszerű MapReduce program példa a streaming használatával a következő lehet:

```
$HADOOP_HOME/bin/hadoop jar $HADOOP_HOME/hadoop-streaming.jar \
    -input myInputDirs \
    -output myOutputDir \
    -mapper /bin/cat \
    -reducer /bin/wc
```

Az egyes paraméterek jelentése a következő:

Paraméter	Leírás
`-input myInputDirs`	A mapper bemenetét tartalmazó könyvtár
`-output myOutputDir`	A reducer kimenetének helye
`-mapper /bin/cat`	A mapper funkciót megvalósító futtatható program (fájl tartalmának kiírása)
`-reducer /usr/bin/wc`	A reducer funkciót megvalósító futtatható program (unix word count parancs)



> https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/sites/2/2020/03/Hadoop-Streaming-1.jpg

A streaming működésének részletei a következők:

- A fenti példában a mapper és reducer funkciókat a paraméterben megadott szkriptek valósítják meg, amelyek a standard bemenetről olvasnak, és a standard kimenetre küldik az eredményt (emit).
- A streaming eszköz a háttérben létrehoz egy klasszikus MapReduce job-ot, amit elküld végrehajtásra a Hadoop klaszterre.
- Ha szkriptet adunk meg mapper megvalósításként, job végrehajtásakor minden egyes map művelet meghívja ezt a szkriptet egy külön folyamatban.
- A mapper konvertálja a bemenetét (kulcs-érték párok) sorokká, és a folyamat standard bemenetére küldi azt. Majd a szkript által előállított eredmény sorokat begyűjti a folyamat standard kimenetéről és visszakonvertálja azokat kulcs-érték párokká, amely a mapper fázis eredményét képezi.
- Ha szkriptet adunk meg reducer megvalósításként, job végrehajtásakor minden egyes reduce művelet meghívja ezt a szkriptet egy külön folyamatban.
- A reducer konvertálja a bemenetét (kulcs-érték párok) sorokká, és a folyamat standard bemenetére küldi azt. Majd a szkript által előállított eredmény sorokat begyűjti a folyamat standard kimenetéről és visszakonvertálja azokat kulcs-érték párokká, amely a reduce fázis eredményét képezi.
- Mind a mapper mind pedig a reducer számára a sor első tab karakteréig tartó rész a kulcs, az utána lévő pedig az érték. Tab karakter hiányában a teljes sort kulcsnak tekintik. Ez megfelelő paraméterekkel módosítható.

📖 2. fejezet
Word count megoldása Python szkript és streaming segítségével

Tekintsük a klasszikus szó összeszámlálós feladatot. Ehhez indítsuk el a Docker alapú Hadoop klasztert, amit a `3g_BigData-hadoop-SPOC` olvasólecke 1. fejezetében ismertettünk. A map és reduce szkripteket Python nyelven valósítjuk meg, amiket a `namenode` container-re fel kell másolnunk, hogy a streaming segítségével futtatni tudjuk őket.

Először lássuk a mapper funkcionalitás Python kódját (`mapper.py`):

```
#!/usr/bin/python
"""mapper.py"""

import sys

# input comes from STDIN (standard input)
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()
    # increase counters
    for word in words:
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        #
```

```
        # tab-delimited; the trivial word count is 1
        print('%s\t%s' % (word, 1))
```

A standard inputról egyszerűen beolvassuk a sorokat, majd whitespace-k mentén széttördeljük (azaz szavakra bontjuk). Minden egyes szóhoz standard kimenetre írunk egy sort, ahol maga a szó a kulcs, majd egy tabulátor (kulcs és érték elválasztásához), utána pedig egy 1-es érték.

A reducer program a következőképp fest (`reducer.py`):

```
#!/usr/bin/python
"""reducer.py"""

from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

# input comes from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()

    # parse the input we got from mapper.py
    word, count = line.split('\t', 1)

    # convert count (currently a string) to int
    try:
        count = int(count)
    except ValueError:
        # count was not a number, so silently
        # ignore/discard this line
        continue

    # this IF-switch only works because Hadoop sorts map output
    # by key (here: word) before it is passed to the reducer
    if current_word == word:
        current_count += count
    else:
        if current_word:
            # write result to STDOUT
            print('%s\t%s' % (current_word, current_count))
        current_count = count
        current_word = word

# do not forget to output the last word if needed!
if current_word == word:
    print('%s\t%s' % (current_word, current_count))
```

Miután a sorokat kulcs-érték párokká alakítottuk, kihasználjuk, hogy a kulcsok szerint rendezve érkeznek az adatok. Így tulajdonképpen az egyes szavak sorozatának hosszát kell összeszámolnunk, és amikor változik a szó (tudjuk, hogy a másikból már biztosan nem lesz több), a sorozat hosszát, azaz az előző szó összes előfordulásának a számát kiírjuk standard kimenetre a szóval, mint kulccsal együtt.

Másoljuk át a szkripteket a HDFS `namenode`-ra a következő docker parancssal:

```
$ docker cp mapper.py namenode:/
$ docker cp reducer.py namenode:/
```

Mielőtt kipróbálnánk őket a MapReduce job során, telepítsük fel a `python` programcsomagot és teszteljük le a szkripteket a következő módon a `namenode` container-en belül:

```
root@b055549cdaca:/# echo "foo foo quux labs foo bar quux" | python3 mapper.py |
sort -k1,1 | python3 reducer.py
bar     1
foo     3
labs    1
quux    2
```

Ha a telepítés sikeres, és a fenti kimenetet kapjuk, akkor telepítenünk kell a fentiek szerint a `python` programcsomagot a `nodemanager` gépre is. Ezután indítsunk egy streaming job-ot a `namenode` container-en belül az alábbi módon (az `input` könyvtár létrehozását és feltöltését fájlokkal a `3g_BigData-hadoop-SPOC` 1. fejezetében mutattuk be):

```
root@37f7633f3a17:/# hadoop jar /opt/hadoop-3.2.1/share/hadoop/tools/lib/hadoop-
streaming-3.2.1.jar -input /input -output /output -file ./mapper.py -mapper
mapper.py -file ./reducer.py -reducer reducer.py
```

Ahhoz, hogy a Python szkripteket meg tudjuk hívni a job végrehajtása során, fel is kell azokat tölteni a megfelelő node-ra, a `-file` paraméterek pontosan ezt csinálják. Amennyiben a job sikeresen lefutott, az output könyvtárba előálló fájl tartalma a következőképpen néz ki:

```
root@37f7633f3a17:/# hadoop fs -cat /output/part-00000

2020-08-17 12:57:14,787 INFO sasl.SaslDataTransferClient: SASL encryption trust
check: localHostTrusted = false, remoteHostTrusted = false
Bye     1
Goodbye 1
Hadoop  2
Hello   2
World   2
```

📖 3. fejezet
Árfolyam adatok átlagának kiszámítása Python map/reduce segítségével

Valósítsuk meg a xy fejezetben bemutatott Java MapReduce job-ot Python szkriptek és streaming használatával. A feladat a `code\5g_BigData-mapred-SPOC\data\daily_csv.csv` táblázatban szereplő valuta árfolyam értékek átlagát kiszámítani országonként.

A mapper funkció (`currency_mapper.py`) megvalósítása a következő:

```
#!/usr/bin/python
"""currency_mapper.py`"""

import sys
```

```
# input comes from STDIN (standard input)
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # We skip the header line
    if line.startswith('Date'):
        continue
    # split the line on commas
    row = line.split(',')
    # tab-delimited; emit one entry for each line with the current currency rate
if values are present
    if row[1] and row[2]:
        print('%s\t%s' % (row[1], row[2]))
```

A megvalósítás igen egyértelmű. A standard bemeneten fogadjuk a bemenő csv fájl sorait. A fejléc sort eldobjuk, minden más sort pedig a vesszők mentén feldarabolunk. Egy-egy sorhoz egy emit szükséges, azaz egy kiírást végzünk a standard kimenetre. A kulcs mindig az adott sorban megjelenő ország, az érték pedig az aktuális árfolyam adat.

A reducer megvalósítás (`currency_reducer.py`) a következő:

```
#!/usr/bin/python
"""reducer.py"""

import sys

current_country = None
current_sum = 0
count = 0

# input comes from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()

    # parse the input we got from mapper.py
    country, rate = line.split('\t')

    # convert rate to float
    try:
        rate = float(rate)
    except ValueError:
        # rate was not a number, so silently
        # ignore/discard this line
        continue

    if current_country == country:
        current_sum += rate
        count += 1
    else:
        if current_country:
            # write result to STDOUT, which is the sum divided by the number of
values
            print('%s\t%s' % (current_country, current_sum/count))
        current_sum = rate
        current_country = country
```

```
        count = 1

# do not forget to output the last country if needed!
if current_country == country:
    print('%s\t%s' % (current_country, current_sum/count))
```

A word count példához nagyon hasonló módon implementálható. Országonként sorba rendezve kapjuk meg az értékeket, így pusztán annyi a feladatunk, hogy az árfolyam értékeket összeadjuk amíg a következő országhoz tartozó értéket nem kapunk. Ha ez megtörtént, akkor a kiíratásnál arra kell csak ügyelni, hogy ne az értékek összegét, hanem a darabszámmal leosztott változatát, azaz az átlagot írjuk ki.

Töltsük fel a két Python szkriptet, valamint a `daily_csv.csv` fájlt a `namenode` container-be. Mielőtt bármit csinálnánk, teszteljük le a parancssorból az elkészített Python fájlokat:

```
root@37f7633f3a17:/# printf "%s\n%s\n%s\n%s" -,a,12 -,a,20 -,b,12.5 -,b,11.5 |
python currency_mapper.py | python currency_reducer.py
a       16.0

b       12.0
```

Másoljuk fel a szükséges fájlokat a `namenode`-ra:

```
$ docker cp currency_mapper.py namenode:/
$ docker cp currency_reducer.py namenode:/
$ docker cp daily_csv.csv namenode:/
```

Másoljuk fel a `daily_csv.csv` fájlt HDFS-re:

```
root@37f7633f3a17:/# hadoop fs -mkdir /curr_input
root@37f7633f3a17:/# hadoop fs -put daily_csv.csv /curr_input/daily_csv.csv
```

Most pedig futtassuk le a job-ot streaming segítségével:

```
root@37f7633f3a17:/# hadoop jar /opt/hadoop-3.2.1/share/hadoop/tools/lib/hadoop-
streaming-3.2.1.jar -input /curr_input -output /curr_output -file
./currency_mapper.py -mapper currency_mapper.py -file ./currency_reducer.py -
reducer currency_reducer.py
```

Sikeres futtatás után a kimeneti könyvtárban előáll az eredmény:

```
root@37f7633f3a17:/# hadoop fs -cat /curr_output/part-00000

2020-08-17 14:22:15,586 INFO sasl.SaslDataTransferClient: SASL encryption trust
check: localHostTrusted = false, remoteHostTrusted = false
Australia       1.21374897179

Brazil  2.14264140327

Canada  1.21786241406

China   6.15819406403
```

```
Denmark 6.62108855104

Euro    0.846233340345
Hong Kong       7.65389236658

India   31.294657238

Japan   161.394100603

Malaysia        2.99091501744

Mexico  11.1633652987

New Zealand     1.46022580151

Norway  6.65583709869

Singapore       1.67174713177

South Africa    4.79191671203

South Korea     981.608407379

Sweden  6.75538317479

Switzerland     1.67871153291

Taiwan  31.2081810328

Thailand        31.2990204807

United Kingdom  0.591069834395

Venezuela       3.00335605007
```

✅ További feladatok

1. Módosítsuk a word count példát úgy, hogy akkor is működjön, ha a reducer nem sorba rendezve kapja meg az inputját, azaz gyűjtsük egy belső dictionary-be az előfordulások gyakoriságát, és a végén azt írassuk ki! ⭐
2. Módosítsuk az árfolyam átlag számító példát úgy, hogy ne számítsuk ki az árfolyamok összegét, hanem az átlagot inkrementálisan [4], minden újabb input sor után frissítsük! ⭐
3. Módosítsuk az árfolyam átlag számító példát úgy, hogy az átlag helyett a medián értéket számítsa ki! ⭐
4. Írjunk egy Java MapReduce job-ot, amely egy lebutított streaming megoldást implementál, azaz a mapper és reducer funkció is egy külső végrehajtható állományt hívjon meg (`Runtime.exec()`), és standard bemenetükre továbbítsa az aktuális adatot, valamint a standard kimenetükről gyűjtse be azt és konvertálja megfelelően kulcs-érték párokra! ⭐ ⭐⭐

Referenciák

[1] http://hadoop.apache.org/docs/r1.2.1/streaming.html

[2] https://data-flair.training/blogs/hadoop-streaming/

[3] https://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/

[4] https://ubuntuincident.wordpress.com/2012/04/25/calculating-the-average-incrementally/