Tactical data engineering using Hadoop streaming and Python

Rohan Kekatpure

Intuit, Inc

rohan_kekatpure@intuit.com

October 13, 2015

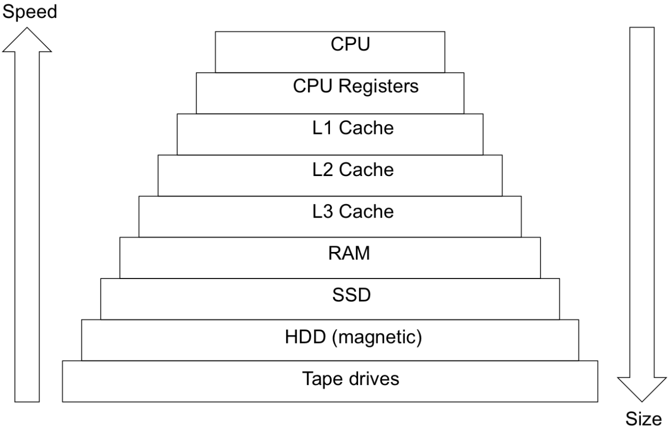
Data engineering patterns

- Tactical
  - Initial exploration, gaining familiarity
  - Short term, less time for planning
- Strategic
  - Longer planning
  - Established tools
  - Known architectural patterns

- Many data ingestion projects begin in tactical mode
- Strategic projects are a series of tactical steps

Streaming is important in tactical (and strategic) data engineering

The invariant in a changing landscape



Avoid writing to disk, if possible

What problems can(not) be expressed as streaming

- OK
  - Mutually independent tasks
  - Line-oriented input
  - Text processing, parsing, enrichment. . .
- Difficult
  - Interacting parts
  - Matrix-like input
  - $Ax = b$ (linear equation systems), $Ax = \lambda x$ (eigenvalues), $A = U \Sigma V^T$ (SVD)

Hadoop and HDFS

Hadoop = HDFS + Mapreduce

- HDFS + streaming
- Streaming mapreduce
- Hive + streaming UDFs

Streaming pattern

```
<input rows> | exec1 | exec2 ... | execn  <output>
```

Generate data

```python
# generate the data
import json
from datetime import datetime
import string
import random

states = ["Alabama", "Arkansas", ...]

with open("random_data.json", "w") as rd:
for _ in range(10000):
payload = json.dumps({
    "userid": "".join(random.sample(string.digits, 7)),
    "state": random.sample(states, 1)[0],
    "message": "".join(random.sample(string.ascii_letters, 16)),
    "source_timestamp": datetime.now().strftime("%Y-%m-%d %H:%M:%S")
})
rd.write("%s\n" % payload)
```

```bash
# Upload to Hadoop (done in Bash shell)
$ hadoop fs -put random_data.json ibd2015
```

State count: HDFS + local shell

```
# Get frequency count of states
$ hadoop fs -cat  ibd2015/random_data.json | \
grep -oiE ’"state": "([a-zA-Z]*)"’ | \
sort | \
uniq -c | \
sort -k1nr
```

State count: HDFS + local Python

```
import sys
import json
from collections import defaultdict

counter = defaultdict(int)
# Count the number of states in a dictionary (hashmap)
for line in sys.stdin:
obj = json.loads(line)
counter[obj["state"]] += 1

print counter
```

```
# Pipe HDFS output to Python (done in shell)
$ hadoop fs -cat ibd2015/random_data.json | \
   python ex1.py
```

HDFS + local shell + streaming put (upload)

```
$ hadoop fs -cat ibd2015/random_data.json | \
   grep -oiE ’"message": "([a-zA-Z]*)"’| \
   hadoop fs -put - ibd2015/output.txt
```

HDFS + local Python + streaming put (upload)

```
# Get day of the week from timestamp
from datetime import datetime
import json
import sys

FMT = "%Y-%m-%d %H:%M:%S"

for line in sys.stdin:
    obj = json.loads(line)
    dt_str = obj["source_timestamp"]
    dt_obj = datetime.strptime(dt_str, FMT)
    dt_dow = dt_obj.strftime("%A")
    obj.update({"dow": dt_dow})
    print json.dumps(obj)
```

```
# Process + streaming upload to HDFS
$ hadoop fs -cat ibd2015/random_data.json | \
   python ex3.py | \
   hadoop fs -put - ibd2015/output2.txt
```

What did we just do?

- Read HDFS data
- Processed using Shell and Python
- Wrote it back to HDFS
- Without a single disk copy

Pros and cons?

What did we just do?

- Read HDFS data
- Processed using Shell and Python
- Wrote it back to HDFS
- Without a single disk copy

Pros and cons

- Computation was happening on single node
- Limited by resources of single node
- Single threaded
- Limited by IO ?

Check out the docs

https://hadoop.apache.org/docs/r1.2.1/streaming.html

DOW using Hadoop streaming MapReduce

```python
#! /usr/bin/python
from datetime import datetime
import json
import sys

FMT = "%Y-%m-%d %H:%M:%S"

for line in sys.stdin:
    obj = json.loads(line)
    dt_str = obj["source_timestamp"]
    dt_obj = datetime.strptime(dt_str, FMT)
    dt_dow = dt_obj.strftime("%A")
    obj.update({"dow": dt_dow})
    print json.dumps(obj)
```

```bash
#!/bin/bash

HADOOP_STREAMING_JAR_PATH="</path/to/streaming/jar>"
PROJECT_HOME="/home/rkekatpure-admin/sandbox/tutorials"
HDFS_INPUT_PATH="/user/rkekatpure-admin/ibd2015/random_data.json"
HDFS_OUTPUT_PATH="/user/rkekatpure-admin/ibd2015/mroutput_ex4"

hadoop fs -rm -r "$HDFS_OUTPUT_PATH"

hadoop jar "$HADOOP_STREAMING_JAR_PATH/hadoop-streaming.jar" \
-Dmapreduce.job.queuename=exp_dsa \
-file "$PROJECT_HOME/ex4.py" \
-mapper ex4.py \
-input "$HDFS_INPUT_PATH" \
-output "$HDFS_OUTPUT_PATH"
```

NoSQL → SQL using streaming MR

```python
#! /usr/bin/python
import sys
import json

keys = ["userid", "state", "message"]
delim = ’\x01’

for line in sys.stdin:
    obj = json.loads(line)

    try:
        row = [obj[k] for k in keys]
        print delim.join(row)
    except Exception:
        pass
```

```bash
#!/bin/bash

HADOOP_STREAMING_JAR_PATH="</path/to/streaming/jar>"
PROJECT_HOME="/home/rkekatpure-admin/sandbox/tutorials"
HDFS_INPUT_PATH="/user/rkekatpure-admin/ibd2015/random_data.json"
HDFS_OUTPUT_PATH="/user/rkekatpure-admin/ibd2015/mroutput_ex5"

hadoop fs -rm -r "$HDFS_OUTPUT_PATH"

hadoop jar "$HADOOP_STREAMING_JAR_PATH/hadoop-streaming.jar" \
-Dmapreduce.job.queuename=exp_dsa \
-file "$PROJECT_HOME/ex5.py" \
-mapper ex5.py \
-input "$HDFS_INPUT_PATH" \
-output "$HDFS_OUTPUT_PATH"
```

Create Hive external table (view)

createhivetable.sql

```
-- Drop database if it exists
drop database if exists ibd2015 cascade;

-- Create database
create database ibd2015;
use ibd2015;

-- Create table
create table ibd2015random (
    userid varchar(10),
    state varchar(50),
    message varchar(25)
)
row format delimited
fields terminated by '\u0001'
location '/user/rkekatpure-admin/ibd2015/mroutput_ex5'
```

Text

Hive UDFs in Python

```python
#!/usr/bin/python
import sys
import hashlib

def gethash(s):
    """
    Returns SHA224 hash of input string s
    """
    hasher = hashlib.sha224()
    hasher.update(s)
    return hasher.hexdigest()


DELIM = "\t"
for line in sys.stdin:
    sline = line.strip()
    userid, state, message = sline.split(DELIM)
    print DELIM.join([userid, state, message, gethash(userid)])
```

```sql
-- execute as a Hive script from cmd line
add file /home/rkekatpure-admin/sandbox/tutorials/hiveudf.py;
set mapred.job.ququq.name=exp_dsa;

use ibd2015;
select transform(ibd.userid, ibd.state, ibd.message)
using 'python hiveudf.py'
from ibd2015random ibd
limit 100;
```