

# Big Data Lab

April 24, 2017

# Start an EMR cluster

- Launch an EMR cluster from the AWS console
- Select the "Spark" radio button under Applications
- You can use just 1 EC2 instance (1 master only)
- You *\*must\** select the public/private key pair you created! (this is needed to SSH into the master node)

# Spark's MLlib

- MLlib is Spark's machine learning library
- Goal is to make practical machine learning scalable and easy
- Includes common learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimensionality reduction, lower-level optimization primitives and higher-level pipeline APIs

# FP-Growth

- “FP”=“frequent pattern”
- Like apriori-like algorithms, the first step of FP-growth is to calculate item frequencies and identify frequent items
- Unlike apriori-like algorithms, the second step of FP-growth uses a suffix tree (FP-tree) structure to encode transactions without generating candidate sets explicitly
- Based on the paper Han, Pei, and Yin, “Mining frequent patterns without candidate generation”, SIGMOD, 2000.

# FP-Growth

- After the second step, the frequent itemsets can be extracted from the FP-tree.
- spark.mllib implements a parallel version of FP-growth called PFP, as described in Li et al., PFP: Parallel FP-growth for query recommendation
- PFP distributes the work of growing FP-trees based on the suffices of transactions, and hence more scalable than a single-machine implementation
- Spark's FP-Growth implementation takes the following parameters:
  - minSupport: the minimum support for an itemset to be identified as frequent. For example, if an item appears 3 out of 5 transactions, it has a support of  $3/5=0.6$ .
  - numPartitions: the number of partitions used to distribute the work

# Example



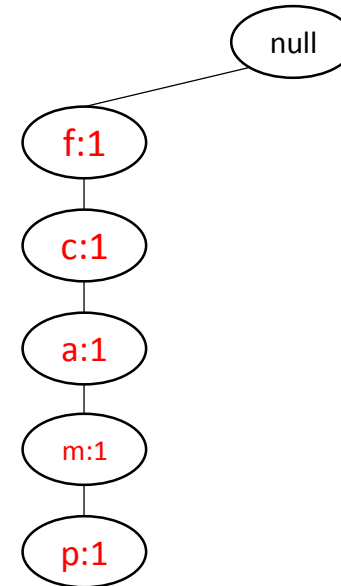
null

TID	Items Bought	(Ordered) Frequent Items
100	f, a, c, d, g, i, m, p	f, c, a, m, p
200	a, b, c, f, l, m, o	f, c, a, b, m
300	b, f, h, j, o	f, b
400	b, c, k, s, p	c, b, p
500	a, f, c, e, l, p, m, n	f, c, a, m, p

- Create root of the tree, labeled “null”

# Example

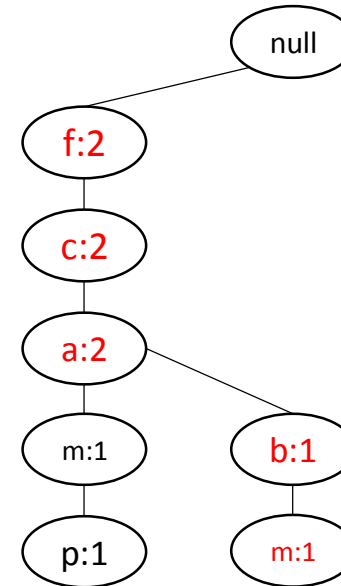
TID	Items Bought	(Ordered) Frequent Items
100	f, a, c, d, g, i, m, p	f, c, a, m, p
200	a, b, c, f, l, m, o	f, c, a, b, m
300	b, f, h, j, o	f, b
400	b, c, k, s, p	c, b, p
500	a, f, c, e, l, p, m, n	f, c, a, m, p



- Create root of the tree, labeled “null”
- Second scan of the DB:
- Scan of first transaction leads to first branch of the tree: <(f:1), (c:1), (a:1), (m:1), (p:1)>

# Example

TID	Items Bought	(Ordered) Frequent Items
100	f, a, c, d, g, i, m, p	f, c, a, m, p
200	a, b, c, f, l, m, o	<b>f, c, a, b, m</b>
300	b, f, h, j, o	f, b
400	b, c, k, s, p	c, b, p
500	a, f, c, e, l, p, m, n	f, c, a, m, p

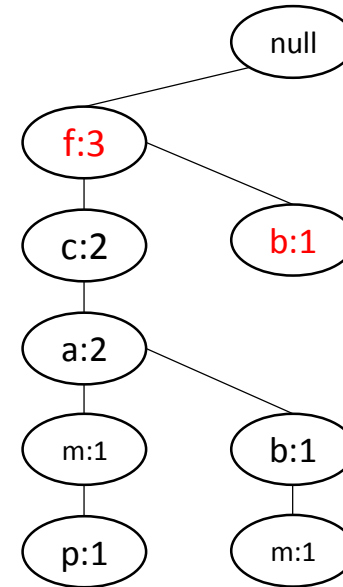


- Create root of the tree, labeled “null”
- Second scan of the DB:
- Scan of first transaction leads to first branch of the tree: <(f:1), (c:1), (a:1), (m:1), (p:1)>
- Second transaction: since its ordered frequent item list shares a common prefix with <f, c, a> with the existing path <f, c, a, m, p>, the count of each node along the prefix is incremented by 1 and one new node (b:1) is created and linked as a child of (a:2) and another new node (m:1) is create and linked as the child of (b:1)



# Example

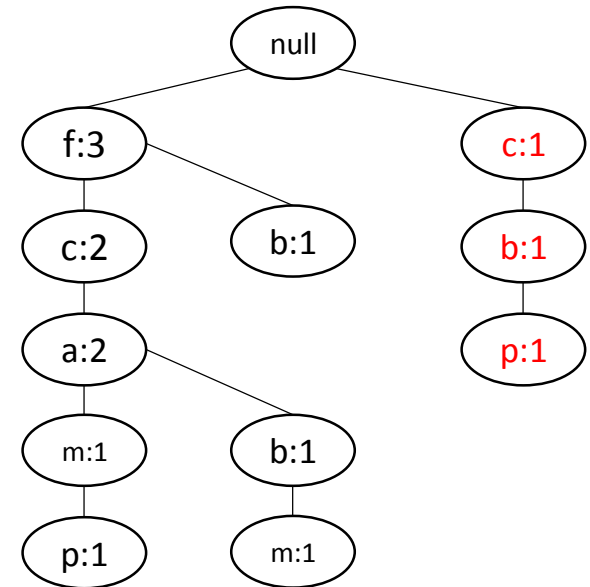
TID	Items Bought	(Ordered) Frequent Items
100	f, a, c, d, g, i, m, p	f, c, a, m, p
200	a, b, c, f, l, m, o	f, c, a, b, m
300	b, f, h, j, o	<b>f, b</b>
400	b, c, k, s, p	c, b, p
500	a, f, c, e, l, p, m, n	f, c, a, m, p



- Create root of the tree, labeled “null”
- Second scan of the DB:
- Scan of first transaction leads to first branch of the tree: <(f:1), (c:1), (a:1), (m:1), (p:1)>
- Second transaction: since its ordered frequent item list shares a common prefix with <f, c, a> with the existing path <f, c, a, m, p>, the count of each node along the prefix is incremented by 1 and one new node (b:1) is created and linked as a child of (a:2) and another new node (m:1) is create and linked as the child of (b:1)
- Third transaction: f’s count is incremented by 1, (b:1) is created as a child of (f:3)

# Example

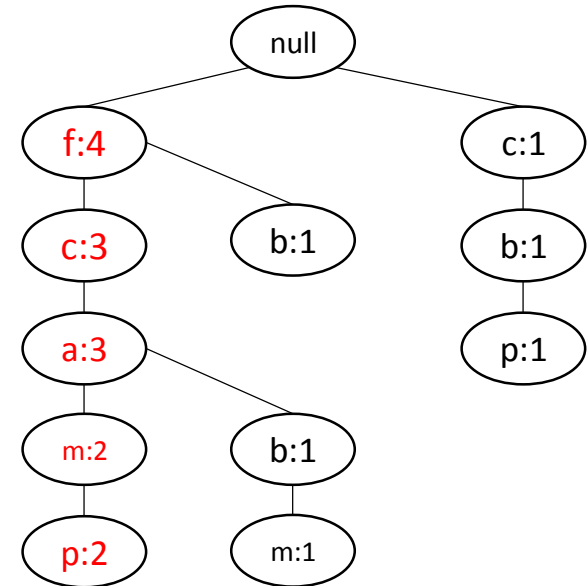
TID	Items Bought	(Ordered) Frequent Items
100	f, a, c, d, g, i, m, p	f, c, a, m, p
200	a, b, c, f, l, m, o	f, c, a, b, m
300	b, f, h, j, o	f, b
400	b, c, k, s, p	<b>c, b, p</b>
500	a, f, c, e, l, p, m, n	f, c, a, m, p



- Create root of the tree, labeled “null”
- Second scan of the DB:
- Scan of first transaction leads to first branch of the tree: <(f:1), (c:1), (a:1), (m:1), (p:1)>
- Second transaction: since its ordered frequent item list shares a common prefix with <f, c, a> with the existing path <f, c, a, m, p>, the count of each node along the prefix is incremented by 1 and one new node (b:1) is created and linked as a child of (a:2) and another new node (m:1) is create and linked as the child of (b:1)
- Third transaction: f’s count is incremented by 1, (b:1) is created as a child of (f:3)
- Fourth transaction: Since no prefix in common, create the second branch of the tree <(c:1), (b:1), (p:1)>

# Example

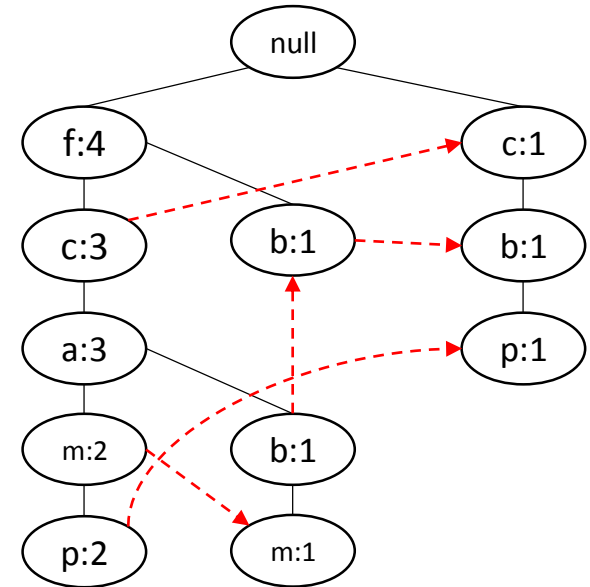
TID	Items Bought	(Ordered) Frequent Items
100	f, a, c, d, g, i, m, p	f, c, a, m, p
200	a, b, c, f, l, m, o	f, c, a, b, m
300	b, f, h, j, o	f, b
400	b, c, k, s, p	c, b, p
500	a, f, c, e, l, p, m, n	f, c, a, m, p



- Create root of the tree, labeled “null”
- Second scan of the DB:
- Scan of first transaction leads to first branch of the tree: <(f:1), (c:1), (a:1), (m:1), (p:1)>
- Second transaction: since its ordered frequent item list shares a common prefix with <f, c, a> with the existing path <f, c, a, m, p>, the count of each node along the prefix is incremented by 1 and one new node (b:1) is created and linked as a child of (a:2) and another new node (m:1) is create and linked as the child of (b:1)
- Third transaction: f’s count is incremented by 1, (b:1) is created as a child of (f:3)
- Fourth transaction: Since no prefix in common, create the second branch of the tree <(c:1), (b:1), (p:1)>
- Fifth transaction: Since same as the first, each count incremented by 1

# Example

TID	Items Bought	(Ordered) Frequent Items
100	f, a, c, d, g, i, m, p	f, c, a, m, p
200	a, b, c, f, l, m, o	f, c, a, b, m
300	b, f, h, j, o	f, b
400	b, c, k, s, p	c, b, p
500	a, f, c, e, l, p, m, n	f, c, a, m, p

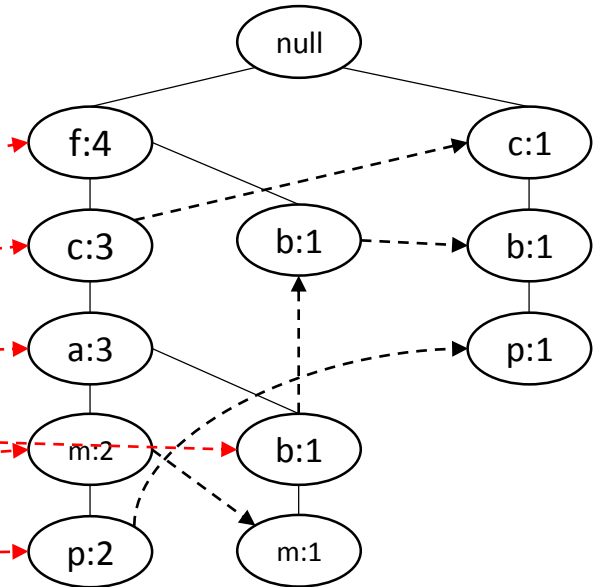


- Create root of the tree, labeled “null”
- Second scan of the DB:
- Scan of first transaction leads to first branch of the tree:  $\langle (f:1), (c:1), (a:1), (m:1), (p:1) \rangle$
- Second transaction: since its ordered frequent item list shares a common prefix with  $\langle f, c, a \rangle$  with the existing path  $\langle f, c, a, m, p \rangle$ , the count of each node along the prefix is incremented by 1 and one new node  $(b:1)$  is created and linked as a child of  $(a:2)$  and another new node  $(m:1)$  is create and linked as the child of  $(b:1)$
- Third transaction:  $f$ 's count is incremented by 1,  $(b:1)$  is created as a child of  $(f:3)$
- Fourth transaction: Since no prefix in common, create the second branch of the tree  $\langle (c:1), (b:1), (p:1) \rangle$
- Fifth transaction: Since same as the first, each count incremented by 1
- Nodes with the same item name are linked via “node-links”

# Example

Header  
Table

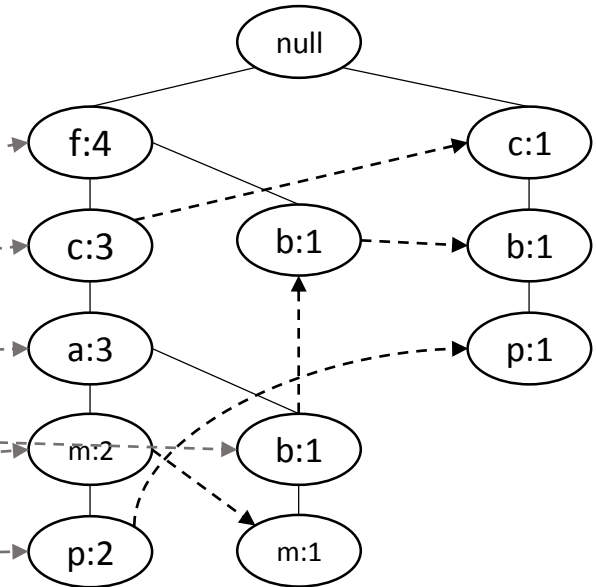
Item	Head of node-links
f	
c	
a	
b	
m	
p	



# Example

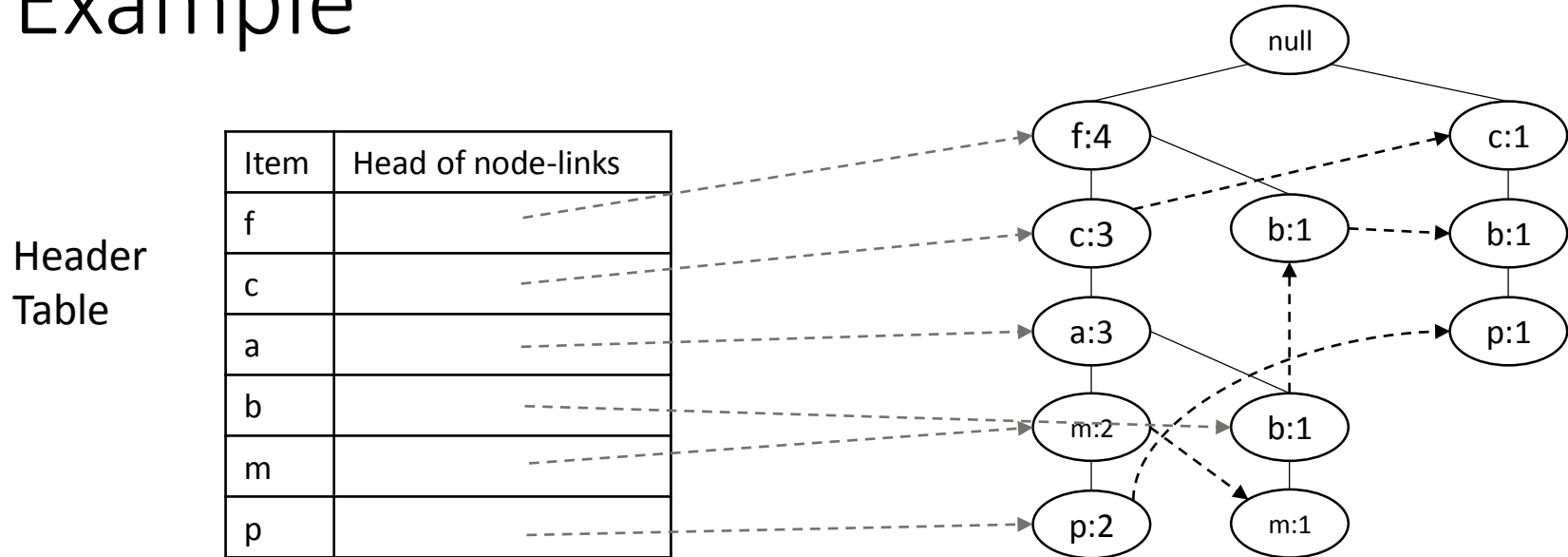
Header  
Table

Item	Head of node-links
f	
c	
a	
b	
m	
p	



- Mining frequent patterns
- Collect all patterns that a node  $x$  participates in by starting from  $x$ 's head (in the header table) and following  $x$ 's node-links

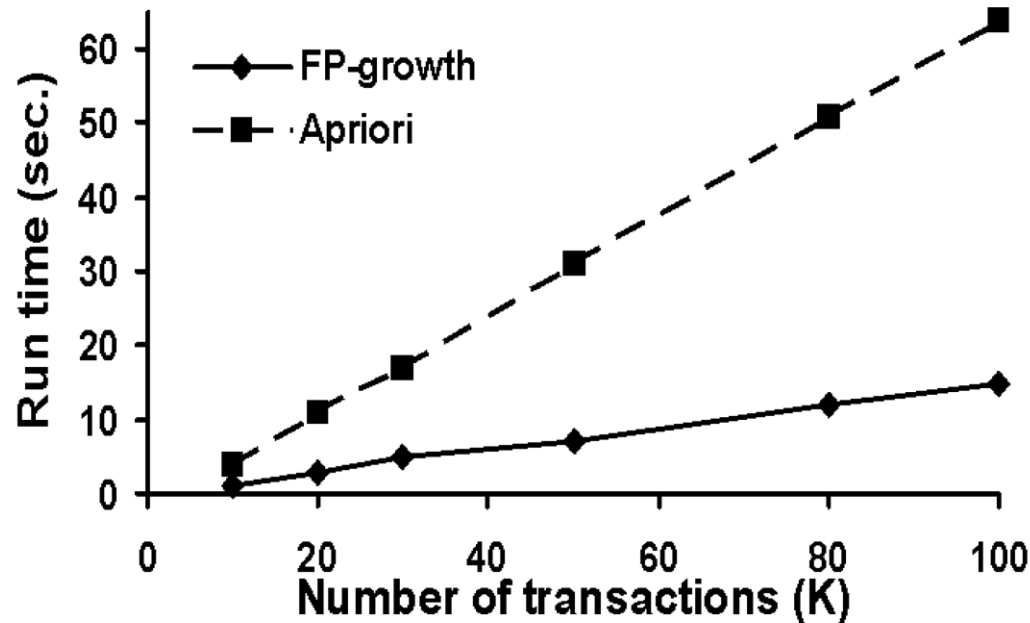
# Example



- Mining frequent patterns
- Collect all patterns that a node  $x$  participates in by starting from  $x$ 's head (in the header table) and following  $x$ 's node-links
- Example: item  $p$
- Node  $p$  derives a frequent pattern ( $p:3$ ) and two paths in the FP tree:  $\langle f:4, c:3, a:3, m:2, p:2 \rangle$  and  $\langle c:1, b:1, p:1 \rangle$ 
  - The first path indicates that the string  $(f,c,a,m,p)$  appears twice in the DB
  - Second path indicates that  $(c,b,p)$  appears once in the DB
- Since both paths contain  $(c,p)$ , this is a frequent pattern,  $(cp:3)$

# Benefits

- Apriori-like algorithms can generate an exponential number of candidates in the worst case, but size of an FP-tree is bounded by the size of its database
- Can lead to faster runtime





# In PySpark

```
from pyspark.mllib.fpm import FPGrowth
```

```
data = sc.textFile("data/mllib/sample_fpgrowth.txt")
```

```
transactions = data.map(lambda line: line.strip().split(' '))
```

```
model = FPGrowth.train(transactions, minSupport=0.2, numPartitions=10)
```

```
result = model.freqItemsets().collect()
```

```
for fi in result:
```

```
    print(fi)
```

# In PySpark

```
from pyspark.mllib.fpm import FPGrowth
```

```
data = sc.textFile("data/mllib/sample_fpgrowth.txt")
```

```
transactions = data.map(lambda line: line.strip().split(' '))
```

```
model = FPGrowth.train(transactions, minSupport=0.2, numPartitions=10)
```

```
result = model.freqItemsets().collect()
```

```
for fi in result:
```

```
    print(fi)
```


# In PySpark

```
from pyspark.mllib.fpm import FPGrowth
```

```
data = sc.textFile("data/mllib/sample_fpgrowth.txt")
```

```
transactions = data.map(lambda line: line.strip().split(' '))
```

takes an RDD of transactions, where each transaction is an List of items of a generic type



```
model = FPGrowth.train(transactions, minSupport=0.2, numPartitions=10)
```

```
result = model.freqItemsets().collect()
```

```
for fi in result:
```

```
    print(fi)
```

# In PySpark

```
from pyspark.mllib.fpm import FPGrowth
```

```
data = sc.textFile("data/mllib/sample_fpgrowth.txt")
```

```
transactions = data.map(lambda line: line.strip().split(' '))
```

takes an RDD of transactions, where each transaction is an List of items of a generic type

```
model = FPGrowth.train(transactions, minSupport=0.2, numPartitions=10)
```

Returns an FPGrowthModel object

```
result = model.freqItemsets().collect()
```

```
for fi in result:
```

```
    print(fi)
```

# In PySpark

```
from pyspark.mllib.fpm import FPGrowth
```

```
data = sc.textFile("data/mllib/sample_fpgrowth.txt")
```

```
transactions = data.map(lambda line: line.strip().split(' '))
```

takes an RDD of transactions, where each transaction is an List of items of a generic type

```
model = FPGrowth.train(transactions, minSupport=0.2, numPartitions=10)
```

Returns an FPGrowthModel object

```
result = model.freqItemsets().collect()
```

Returns (items, freq) tuples

```
for fi in result:
```

```
    print(fi)
```

# In PySpark

```
from pyspark.mllib.fpm import FPGrowth
```

```
data = sc.textFile("data/mllib/sample_fpgrowth.txt")
```

```
transactions = data.map(lambda line: line.strip().split(' '))
```

takes an RDD of transactions, where each transaction is an List of items of a generic type

```
model = FPGrowth.train(transactions, minSupport=0.2, numPartitions=10)
```

Returns an FPGrowthModel object

```
result = model.freqItemsets().collect()
```

Returns (items, freq) tuples

```
for fi in result:
```

```
    print(fi)
```

Could access item list by fi.items, frequency by fi.freq

# Resources

- <https://spark.apache.org/docs/latest/mllib-frequent-pattern-mining.html>
- <https://spark.apache.org/docs/latest/api/python/pyspark.mllib.html#pyspark.mllib.fpm.FPGrowth>

# Set Security Group Settings

- In order to connect to your master node via SSH, you will need to first modify your security group.
- To do this, go to the cluster you just created, and click on the blue link following “Security groups for Master”.

[Add step](#) [Resize](#) [Clone](#) [Terminate](#) [AWS CLI export](#)

Cluster: My cluster Waiting Cluster ready after last step completed. C

**Connections:** [Enable Web Connection](#) – Hue, Spark History Server, Ganglia, Resource Manager ... (View All)

**Master public DNS:** ec2-52-37-72-119.us-west-2.compute.amazonaws.com [SSH](#)

**Tags:** -- [View All](#) / [Edit](#)

**Summary**

ID: j-L94BLUSDP52J

Creation date: 2016-02-28 22:57 (UTC-5)

Elapsed time: 10 minutes

Auto-terminate: No

Termination protection: Off [Change](#)

**Configuration Details**

Release label: emr-4.3.0

Hadoop Amazon 2.7.1 distribution:

Applications: Ganglia 3.7.2, Hive 1.0.0, Hue 3.7.1, Mahout 0.11.0, Pig 0.14.0, Spark 1.6.0

Log URI: s3://aws-logs-265425801888-us-west-2/elasticmapreduce/ 

EMRFS consistent view: Disabled

**Network and Hardware**

Availability zone: us-west-2a

Subnet ID: [subnet-bbffcfc](#)

Master: Running 1 m3.xlarge

Core: Running 2 m3.xlarge

Task: --

**Security and Access**

Key name: ecc

EC2 instance profile: EMR\_EC2\_DefaultRole

EMR role: EMR\_DefaultRole

Visible to all [Change](#)

Users:

Security groups [sg-5d2f9d3a](#) (ElasticMapReduce-for Master: master)

Security groups [sg-9c2f9d3b](#) (ElasticMapReduce-for Core & Task: slave)

► Monitoring



- In the bottom pane, select the Inbound tab and click the Edit button.

The screenshot shows the AWS IAM console interface for a Security Group. At the top, there's a search bar with 'sg-5d2f9d3a' and a table listing security groups. Below this, the 'Security Group: sg-5d2f9d3a' section is visible. It has tabs for 'Description', 'Inbound', 'Outbound', and 'Tags'. The 'Inbound' tab is selected. In the top left of the 'Inbound' tab, the 'Edit' button is circled in red. Below the tabs is a table of inbound rules.

Type	Protocol	Port Range	Source
All TCP	TCP	0 - 65535	sg-5c2f9d3b (ElasticMapReduce-slave)
All TCP	TCP	0 - 65535	sg-5d2f9d3a (ElasticMapReduce-master)
Custom TCP Rule	TCP	8443	54.240.230.184/29
Custom TCP Rule	TCP	8443	54.240.230.240/29
Custom TCP Rule	TCP	8443	205.251.233.32/28
Custom TCP Rule	TCP	8443	205.251.234.32/28

- Click Add Rule

Edit inbound rules

Type <small>i</small>	Protocol <small>i</small>	Port Range <small>i</small>	Source <small>i</small>	
All TCP ▾	TCP	0 - 65535	Custom IP ▾ sg-5c2f9d3b	✕
All TCP ▾	TCP	0 - 65535	Custom IP ▾ sg-5d2f9d3a	✕
Custom TCP Rule ▾	TCP	8443	Custom IP ▾ 54.240.230.184	✕
Custom TCP Rule ▾	TCP	8443	Custom IP ▾ 54.240.230.240	✕
Custom TCP Rule ▾	TCP	8443	Custom IP ▾ 205.251.233.32	✕
Custom TCP Rule ▾	TCP	8443	Custom IP ▾ 205.251.234.32	✕
Custom TCP Rule ▾	TCP	8443	Custom IP ▾ 205.251.233.16	✕
Custom TCP Rule ▾	TCP	8443	Custom IP ▾ 205.251.233.16	✕
Custom TCP Rule ▾	TCP	8443	Custom IP ▾ 205.251.233.48	✕
Custom TCP Rule ▾	TCP	8443	Custom IP ▾ 54.240.230.176	✕
All UDP ▾	UDP	0 - 65535	Custom IP ▾ sg-5c2f9d3b	✕
All UDP ▾	UDP	0 - 65535	Custom IP ▾ sg-5d2f9d3a	✕
All ICMP ▾	ICMP	0 - 65535	Custom IP ▾ sg-5c2f9d3b	✕
All ICMP ▾	ICMP	0 - 65535	Custom IP ▾ sg-5d2f9d3a	✕

Add Rule

Cancel Save

- select SSH for Type and Anywhere for the Source. Click Save.

Edit inbound rules

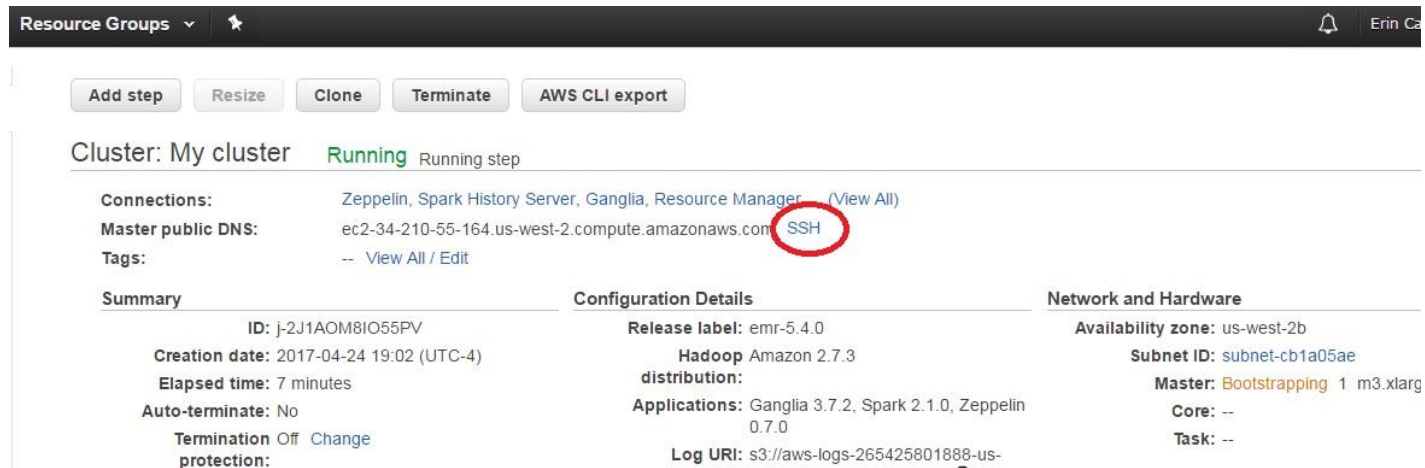
Type <small>i</small>	Protocol <small>i</small>	Port Range <small>i</small>	Source <small>i</small>	
All TCP ▾	TCP	0 - 65535	Custom IP ▾ sg-5c2f9d3b	✕
All TCP ▾	TCP	0 - 65535	Custom IP ▾ sg-5d2f9d3a	✕
Custom TCP Rule ▾	TCP	8443	Custom IP ▾ 54.240.230.184	✕
Custom TCP Rule ▾	TCP	8443	Custom IP ▾ 54.240.230.240	✕
Custom TCP Rule ▾	TCP	8443	Custom IP ▾ 205.251.233.32	✕
Custom TCP Rule ▾	TCP	8443	Custom IP ▾ 205.251.234.32	✕
Custom TCP Rule ▾	TCP	8443	Custom IP ▾ 205.251.233.16	✕
Custom TCP Rule ▾	TCP	8443	Custom IP ▾ 205.251.233.16	✕
Custom TCP Rule ▾	TCP	8443	Custom IP ▾ 205.251.233.16	✕
Custom TCP Rule ▾	TCP	8443	Custom IP ▾ 205.251.233.48	✕
Custom TCP Rule ▾	TCP	8443	Custom IP ▾ 54.240.230.176	✕
All UDP ▾	UDP	0 - 65535	Custom IP ▾ sg-5c2f9d3b	✕
All UDP ▾	UDP	0 - 65535	Custom IP ▾ sg-5d2f9d3a	✕
All ICMP ▾	ICMP	0 - 65535	Custom IP ▾ sg-5c2f9d3b	✕
All ICMP ▾	ICMP	0 - 65535	Custom IP ▾ sg-5d2f9d3a	✕
SSH ▾	TCP	22	Anywhere ▾ 0.0.0.0/0	✕

Add Rule

Cancel Save

# SSH into the EMR master node

- From the EMR console, click "SSH" for instructions on how to SSH into the master node



The screenshot shows the AWS EMR console interface. At the top, there's a header with 'Resource Groups' and a user profile 'Erin Ca'. Below the header, there are buttons for 'Add step', 'Resize', 'Clone', 'Terminate', and 'AWS CLI export'. The main section displays 'Cluster: My cluster' with a status of 'Running' and a 'Running step' indicator. Under 'Connections', the 'Master public DNS' is listed as 'ec2-34-210-55-164.us-west-2.compute.amazonaws.com', and an 'SSH' button is circled in red. Below this, there are three tabs: 'Summary', 'Configuration Details', and 'Network and Hardware'. The 'Summary' tab shows the cluster ID 'j-2J1AOM8IO55PV', creation date '2017-04-24 19:02 (UTC-4)', elapsed time '7 minutes', and 'Auto-terminate: No'. The 'Configuration Details' tab shows 'Release label: emr-5.4.0', 'Hadoop Amazon 2.7.3 distribution', 'Applications: Ganglia 3.7.2, Spark 2.1.0, Zeppelin 0.7.0', and 'Log URI: s3://aws-logs-265425801888-us-'. The 'Network and Hardware' tab shows 'Availability zone: us-west-2b', 'Subnet ID: subnet-cb1a05ae', 'Master: Bootstrapping 1 m3.xlarg', 'Core: --', and 'Task: --'.

Resource Groups ▾ ☆

Erin Ca

Add step Resize Clone Terminate AWS CLI export

Cluster: My cluster **Running** Running step

**Connections:** Zeppelin, Spark History Server, Ganglia, Resource Manager (View All)

**Master public DNS:** ec2-34-210-55-164.us-west-2.compute.amazonaws.com **SSH**

**Tags:** -- View All / Edit

Summary	Configuration Details	Network and Hardware
<b>ID:</b> j-2J1AOM8IO55PV	<b>Release label:</b> emr-5.4.0	<b>Availability zone:</b> us-west-2b
<b>Creation date:</b> 2017-04-24 19:02 (UTC-4)	<b>Hadoop Amazon 2.7.3 distribution:</b>	<b>Subnet ID:</b> subnet-cb1a05ae
<b>Elapsed time:</b> 7 minutes	<b>Applications:</b> Ganglia 3.7.2, Spark 2.1.0, Zeppelin 0.7.0	<b>Master:</b> Bootstrapping 1 m3.xlarg
<b>Auto-terminate:</b> No	<b>Log URI:</b> s3://aws-logs-265425801888-us-	<b>Core:</b> --
<b>Termination protection:</b> Off Change		<b>Task:</b> --

# Get the files

Once SSHed into the master node, get the files for the lab by typing:

```
hadoop fs -get s3://bigdataclassecc/Lab11/freqitems.py
```

```
hadoop fs -get s3://bigdataclassecc/Lab11/groceries.csv
```

```
hadoop fs -copyFromLocal groceries.csv
```

# Run the Sample Program

1. Type

```
cat freqitems.py
```

to view the program

2. To run the job, use the command

```
spark-submit freqitems.py groceries.csv > freqitemsoutput.txt
```

3. Type

```
cat freqitemsoutput.txt
```

to view the output file

# Getting the output from EMR

- To get the output from the program onto your local machine, you can use scp. From a terminal on your local machine, type, for example:

```
scp -i ~/private-key.pem hadoop@ec2-35-161-42-105.us-west-2.compute.amazonaws.com:/home/hadoop/modifiedoutput.txt .
```

on Windows, from command line:

```
pscp -scp -i C:\Users\ecc\Desktop\private-key.ppk hadoop@ec2-35-161-42-105.us-west-2.compute.amazonaws.com:/home/hadoop/modifiedoutput.txt .
```

(red text should be replaced with your path to private key, EMR master address, and file location)

# Deliverable

Due Wednesday, April 26, 2017, 6pm

Suppose you are deciding which items to place next to each other at the grocery store, so you only care about frequent itemsets of size 2 or greater which appear in at least 2% of the transactions.

1. Modify the freqitems.py file to meet these constraints. Your code should print itemsets of size 2 or larger that appear in at least 2% of transactions, **sorted by decreasing frequency**. (This involves both setting an appropriate minSupport and modifying the code to prune itemsets of size 1 and sort).
2. Run the job with your modified freqitems.py file using spark-submit, saving the output to the file modifiedoutput.txt
3. Submit the modifiedoutput.txt file to NYU Classes.