

Finishing up Apache Spark

Susan B. Davidson and Zachary G. Ives

University of Pennsylvania

CIS 545 – Big Data Analytics



*Portions of this lecture have been contributed to the OpenDS4All project,
piloted by Penn, IBM, and the Linux Foundation*

<https://tinyurl.com/cis545-lecture-02-16-22>

Recall: Sharding and Tables

Given a cluster with n workers, running remotely, Spark creates a table with *at least* n partitions (here, 200, where 100 are stored on each machine)

Spark will partition “automatically” but it’s best to *repartition* on the key you want!

```
%%spark
```

```
# 10
```

```
link
```

```
my_1
```

```
link
```

```
re
```

_id	name	locality	skills
in-00000001	[given_name -> Dr...	United States	[Key Account Deve...
in-00001	[given_name -> An...	Antwerp Area, Bel...	[Molecular Biolog...
in-00006	[given_name -> Sh...	San Francisco, Ca...	[DNA, Nanotechnol...
in-000montgomery	[given_name -> Ed...	San Francisco Bay...	null
in-000vijaychauhan	[given_name -> Vi...	Chennai Area, India	[Program Manageme...

only showing top 5 rows

<https://tinyurl.com/cis545-lecture-02-16-22>

Computation in a Sharded System: Selection, Projection

_id	name	locality	skills
in-00000001	[given_name -> Dr...	United States	[Key Account Deve...
in-00001	[given_name -> An...	Antwerp Area, Bel...	[Molecular Biolog...
in-00006	[given_name -> Sh...	San Francisco, Ca...	[DNA, Nanotechnol...
in-000montgomery	[given_name -> Ed	San Francisco Bay	null
in-000vijaychauhan	[given_name -> Vi		

only showing top 5 rows

Selection + projection is “farme

```
linked_df.filter(linked_df.loc[
    'name', 'locality']).show(5)
```

_id	name	locality
in-00000001	[given_name -> Dr...	United States
in-100percenthair	[given_name -> Su...	United States
in-1solone	[given_name -> Ha...	United States
in-2raviagarwal	[given_name -> Ra...	United States
in-aarongatescarlton	[given_name -> Aa...	United States

only showing top 5 rows

<https://tinyurl.com/cis545-lecture-02-16-22>

Apply (with Python Functions) in Spark

<https://docs.databricks.com/spark/latest/spark-sql/udf-python.html>

```
%%spark
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType

acro = udf(lambda x: ''.join([n[0] for n in x]), StringType)

linked_df.select("id", acro("locality").alias("acronym"))
```

Note also that we used Spark's `select` arguments looks much like a list for

As with `select` / `project`, `apply` is run parallel!

_id	acronym
in-00000001	US
in-00001	AAB
in-00006	SFC
in-000montgomery	SFBA
in-000vijaychauhan	CAI

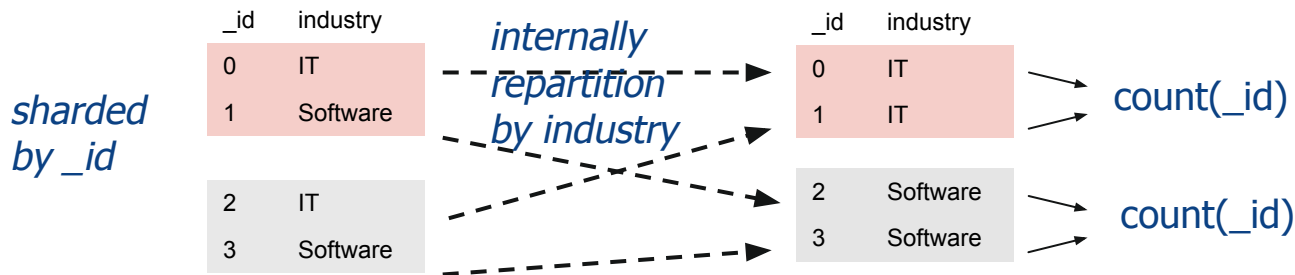
only showing top 5 rows

<https://tinyurl.com/cis545-lecture-02-16-22>

Grouping

Grouping needs *all* of the tuples in a group to be on the same machine, in order to do a computation over the group!

```
%%spark
# Which industries are most popular?
sqlContext.sql('select count(_id), industry '+\
'from linked_in group by industry '+\
'order by count(_id) desc').show(5)
```



<https://tinyurl.com/cis545-lecture-02-16-22>

Failures

- In a large cluster running for a long time – machines may die or software may crash
- Spark actually handles such failures transparently
 - It periodically “checkpoints” or snapshots what has happened
 - And if a node dies, it can restart the computation elsewhere!

<https://tinyurl.com/cis545-lecture-02-16-22>

Brief Review

<https://canvas.upenn.edu/courses/1636888/quizzes/2771539> (08F)

Spark is not written in Python, which means;

- a. Spark is slower than Pandas
- b. schemas are strongly typed
- c. we can't use Python strings
- d. Spark must be written in C

How do we handle (a small number of) worker failures when Spark tasks are executed?

- e. Spark handles this transparently
- f. We need to buy new servers
- g. We have to retry our jobs
- h. We have to write try/except blocks in Python

<https://tinyurl.com/cis545-lecture-02-16-22>

Recap

Summary of Big Data so far:

- We need to partition or *shard* data by keys, allowing machines to work in parallel across different shards
- Apache Spark is the most popular system for doing this right now
 - Supports Spark dataframes or Spark SQL
 - Some variations from Pandas: strong typing, syntax variations, special **udf** function
 - You can control sharding via **repartition**
 - Select, project, **apply** all work in parallel across shards
 - Grouping typically requires the machines to exchange or repartition data

<https://tinyurl.com/cis545-lecture-02-16-22>

Big Data and Cloud Services

Susan B. Davidson and Zachary G. Ives

University of Pennsylvania

CIS 545 – Big Data Analytics



*Portions of this lecture have been contributed to the OpenDS4All project,
piloted by Penn, IBM, and the Linux Foundation*

<https://tinyurl.com/cis545-lecture-02-16-22>

What Do People Mean by “Big Data”?

“5 V’s of Big Data”, seeking *value*:

- **Veracity**: Data is of high *quality*
- **Variety**: Data is *heterogeneous*
- **Volume**:
 - Many rows
 - Large data objects
- **Velocity**: Data changes often



Data has many **dimensions**

<https://tinyurl.com/cis545-lecture-02-16-22>

Roadmap for this Module

- Cloud-hosted compute clusters for big data
- Distributed Spark execution and joins
- Storing big data on the cloud
- View materialization

... Preparing us for looking at complex *graphs* of relationships!

<https://tinyurl.com/cis545-lecture-02-16-22>

Elastic MapReduce and Clusters on the Cloud

Susan B. Davidson and Zachary G. Ives

University of Pennsylvania

CIS 545 – Big Data Analytics



*Portions of this lecture have been contributed to the OpenDS4All project,
piloted by Penn, IBM, and the Linux Foundation*

<https://tinyurl.com/cis545-lecture-02-16-22>

Clusters on the Cloud

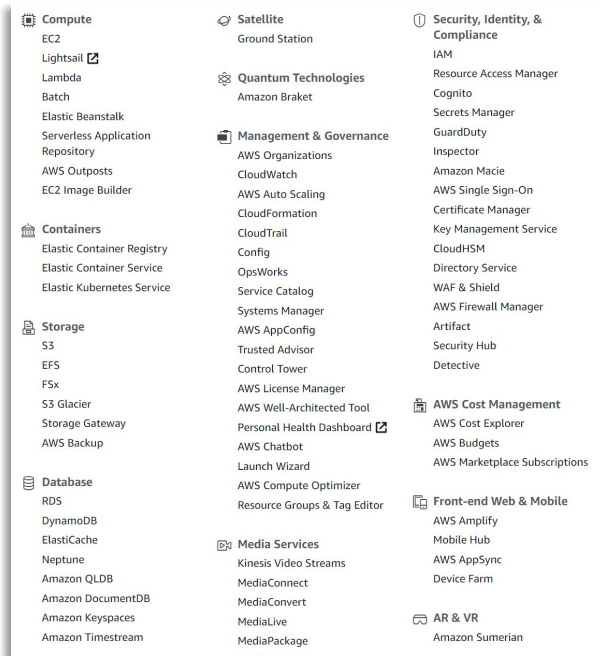


How do we run Spark in clusters on the cloud?

- Cloud service “layers”
- Platform-as-a-service for big data

<https://tinyurl.com/cis545-lecture-02-16-22>

An Overwhelming Array of Services



Amazon has over 50 cloud services!

(Azure, Google, Oracle have fewer, but still a large number)

How do we decide what we need? Understand our options

No cloud standard exists, but we've converged on a rough taxonomy...

<https://tinyurl.com/cis545-lecture-02-16-22>

A Taxonomy: Cloud *Service Layers*

Software as a Service (SaaS) – applications hosted on the cloud

Netflix, GMail, Facebook, Salesforce

Platform as a Service (PaaS) – libraries, specialized platforms

Colab, Google Compute Engine, Amazon Elastic MapReduce

Infrastructure as a Service (IaaS) – “raw” machines & storage

Amazon Elastic Compute Cloud, Simple Storage Service

<https://tinyurl.com/cis545-lecture-02-16-22>

Colab (and AWS SageMaker Notebook)

Cloud-hosted Platform-as-a-Service / Software-as-a-Service hybrid

- Google-customized Jupyter Notebook on Ubuntu 16.04 with Python 3.6
- (Possible to install single-node Spark)

But to get the most out of Spark, we need to connect Colab to a cluster!

<https://tinyurl.com/cis545-lecture-02-16-22>

Our Main Focus: AWS Elastic MapReduce

Preconfigured compute clusters!

Built over EC2, and you can always
go down to the EC2 level

Pick number of machines, configura
details, launch and use!

<https://tinyurl.com/cis545-lecture-02-16-22>

Software configuration

Release

- Applications
- ☒ Core Hadoop: Hadoop 2.10.0, Hive 2.3.7, Hue 4.7.1, Mahout 0.13.0, Pig 0.17.0, and Tez 0.9.2
 - ☐ HBase: HBase 1.4.13, Hadoop 2.10.0, Hive 2.3.7, Hue 4.7.1, Phoenix 4.14.3, and ZooKeeper 3.4.14
 - ☐ Presto: Presto 0.238.3 with Hadoop 2.10.0 HDFS and Hive 2.3.7 Metastore
 - ☐ Spark: Spark 2.4.6 on Hadoop 2.10.0 YARN and Zeppelin 0.8.2

☐ Use AWS Glue Data Catalog for table metadata

Hardware configuration


Instance type

Number of instances (1 master and 2 core nodes)

Cluster scaling ☐ scale cluster nodes based on workload

Software

Software Configuration

Release 

<input checked="" type="checkbox"/> Hadoop 3.2.1	<input type="checkbox"/> Zeppelin 0.9.0	<input checked="" type="checkbox"/> Livy 0.7.0
<input type="checkbox"/> JupyterHub 1.1.0	<input type="checkbox"/> Tez 0.9.2	<input type="checkbox"/> Flink 1.11.0
<input type="checkbox"/> Ganglia 3.7.2	<input type="checkbox"/> HBase 2.2.5	<input checked="" type="checkbox"/> Pig 0.17.0
<input checked="" type="checkbox"/> Hive 3.1.2	<input type="checkbox"/> Presto 0.232	<input type="checkbox"/> PrestoSQL 338
<input type="checkbox"/> ZooKeeper 3.4.14	<input type="checkbox"/> MXNet 1.6.0	<input type="checkbox"/> Sqoop 1.4.7
<input checked="" type="checkbox"/> Hue 4.7.1	<input type="checkbox"/> Phoenix 5.0.0	<input type="checkbox"/> Oozie 5.2.0
<input checked="" type="checkbox"/> Spark 3.0.0	<input type="checkbox"/> HCatalog 3.1.2	<input type="checkbox"/> TensorFlow 2.1.0

- For now: need at least Spark, Livy, Hive
- We'll have a largely-preconfigured template for you
- Later for deep learning: MXNet, PyTorch

<https://tinyurl.com/cis545-lecture-02-16-22>

Creating a Cluster

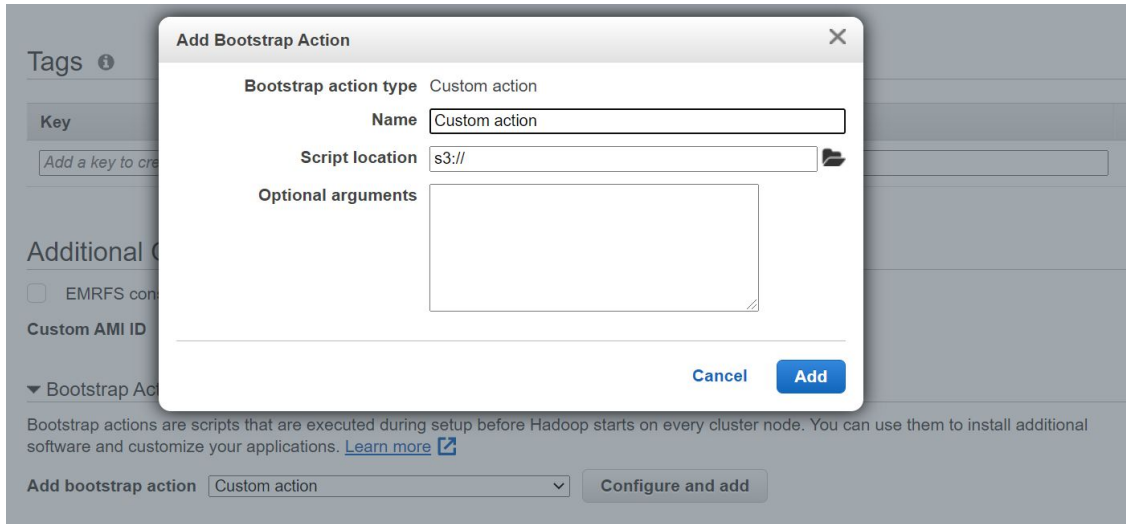
Node type	Instance type				
Master Master - 1	m5.xlarge 4 vCore, 16 GiB memory, EBS only storage EBS Storage: 64 GiB Add configuration settings	<input type="radio"/>	m5.8xlarge	32	128 EBS only
		<input type="radio"/>	m5.12xlarge	48	192 EBS only
		<input type="radio"/>	m5.16xlarge	64	256 EBS only
		<input type="radio"/>	m5.24xlarge	96	384 EBS only
		<input type="radio"/>	m5a.xlarge	4	16 EBS only
		<input type="radio"/>	m5a.2xlarge	8	32 EBS only
		<input type="radio"/>	m5a.4xlarge	16	64 EBS only
		<input type="radio"/>	m5a.8xlarge	32	128 EBS only
		<input type="radio"/>	m5a.12xlarge	48	192 EBS only
		<input type="radio"/>	m5a.16xlarge	64	256 EBS only
		<input type="radio"/>	m5a.24xlarge	96	384 EBS only
Core Core - 2	m5.xlarge 4 vCore, 16 GiB memory, EBS only storage EBS Storage: 64 GiB Add configuration settings				
Task Task - 3	m5.xlarge 4 vCore, 16 GiB memory, EBS only storage EBS Storage: 64 GiB Add configuration settings				

☐ Spot
Use on-demand as max price

<https://tinyurl.com/cis545-lecture-02-16-22>

Bootstrap Actions

To install Python (or Java/Scala) packages across the cluster, set up a shell script as a *bootstrap action* and place on AWS S3



The screenshot shows the 'Add Bootstrap Action' dialog box in the AWS EMR console. The dialog has a title bar with a close button (X). Inside, the 'Bootstrap action type' is set to 'Custom action'. The 'Name' field contains 'Custom action'. The 'Script location' field contains 's3://' and has a folder icon to its right. The 'Optional arguments' field is a large empty text area. At the bottom right of the dialog are 'Cancel' and 'Add' buttons. Below the dialog, the background shows the 'Add bootstrap action' section of the console, with a dropdown menu set to 'Custom action' and a 'Configure and add' button. A descriptive paragraph about bootstrap actions is also visible.

Tags ⓘ

Key

Add a key to create

Additional

☐ EMRFS con

Custom AMI ID

▼ Bootstrap Ac

Bootstrap actions are scripts that are executed during setup before Hadoop starts on every cluster node. You can use them to install additional software and customize your applications. [Learn more](#) ⓘ

Add bootstrap action Custom action ▼ Configure and add

<https://tinyurl.com/cis545-lecture-02-16-22>

A Spark Cluster

Summary

ID: j-1TI9TBPVGD6YY


Creation date: 2020-10-03 10:28 (UTC-4)

Elapsed time: 9 minutes

After last step completes: Cluster waits

Termination protection: Off [Change](#)

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

Master public DNS: ec2-54-159-35-214.compute-1.amazonaws.com 

[Connect to the Master Node Using SSH](#)

Configuration details

Release label: emr-6.1.0

Hadoop distribution: Amazon 3.2.1


Applications: Hive 3.1.2, Pig 0.17.0, Hue 4.7.1, Spark 3.0.0, Livy 0.7.0, MXNet 1.6.0

Log URI: s3://aws-logs-884743372678-us-east-1/elasticmapreduce/ 

EMRFS consistent view: Disabled

Custom AMI ID: --

⚠ Not secure | ec2-54-159-35-214.compute-1.amazonaws.com:8998/ui ☆

 Sessions

Interactive Sessions

Show entries

Search:

Session Id	Application Id	Name	Owner	Proxy User	Session Kind	State	Logs
0					pyspark	starting	session

Showing 1 to 1 of 1 entries

Previous **1** Next

<https://tinyurl.com/cis545-lecture-02-16-22>

Google DataProc

Name [?]
cluster-7463

Region [?] us-central1 Zone [?] us-central1-b

Cluster mode [?]
Standard (1 master, N workers)

Master node
Contains the YARN Resource Manager, HDFS NameNode, and all job drivers

Machine configuration

Machine family
General-purpose
Machine types for common workloads, optimized for cost and flexibility

Series
N1
Powered by Intel Skylake CPU platform or one of its predecessors

Machine type
n1-standard-4 (4 vCPU, 15 GB memory)

	vCPU	Memory	GPUs
	4	15 GB	-

[CPU platform and GPU](#)

Primary disk size (minimum 15 GB) [?] 500 GB Primary disk type [?] Standard persistent disk

Standard Cloud Dataproc image Custom image

Cloud Dataproc uses versioned images to bundle the operating system, big data components, and Google Cloud Platform connectors into one packages that is deployed on your cluster. [Learn more](#)

- ☐ 1.5 (Debian 10, Hadoop 2.10, Spark 2.4)
First released on 3/25/2020.
- ☐ 1.4 (Debian 10, Hadoop 2.9, Spark 2.4)
First released on 3/22/2019.
- ☒ 1.3 (Debian 10, Hadoop 2.9, Spark 2.3)
First released on 8/16/2018.
- ☐ 1.5 (Ubuntu 18.04 LTS, Hadoop 2.10, Spark 2.4)
First released on 3/25/2020.
- ☐ 1.4 (Ubuntu 18.04 LTS, Hadoop 2.9, Spark 2.4)
First released on 3/22/2019.
- ☐ 1.3 (Ubuntu 18.04 LTS, Hadoop 2.9, Spark 2.3)
First released on 3/22/2019.
- ☐ PREVIEW 2.0 (Debian 10, Hadoop 3.2, Spark 3.0)
Preview released on 6/10/2020.
- ☐ PREVIEW 2.0 (Ubuntu 18.04 LTS, Hadoop 3.2, Spark 3.0)
Preview released on 6/10/2020.

<https://tinyurl.com/cis545-lecture-02-16-22>

Microsoft Azure Databricks

Categories

All
General
Compute
Networking
Storage
Web
Mobile
Containers
Databases
Analytics
Blockchain
AI + machine learning
Internet of things
Mixed reality
Integration
Identity
Security
DevOps
Migrate
Monitor
Management + governance
Intune
Other

ANALYTICS (14)

 Azure Synapse Analytics (for

 HDInsight clusters

 Power BI Embedded

 Data Lake Analytics

 Event Hubs

 Log Analytics workspaces

 Azure Data Explorer Clusters

Microsoft Azure

Create Cluster

New Cluster Cancel Create Cluster 2-8 Workers: 28.0-112.0 GB Memory, 8-32 Cores, 1.5-6 DBU
1 Driver: 14.0 GB Memory, 4 Cores, 0.75 DBU

Cluster Name

Cluster Mode

Pool

Databricks Runtime Version [Learn more](#)

New This Runtime version supports only Python 3.

Autopilot Options
☒ Enable autoscaling
☒ Terminate after minutes of inactivity

Worker Type

	Min Workers	Max Workers
<input type="text" value="Standard_DS3_v2"/> 14.0 GB Memory, 4 Cores, 0.75 DBU	<input type="text" value="2"/>	<input type="text" value="8"/>

Driver Type
 14.0 GB Memory, 4 Cores, 0.75 DBU

Advanced Options

<https://tinyurl.com/cis545-lecture-02-16-22>

Brief Review

<https://canvas.upenn.edu/courses/1636888/quizzes/2771536> (09B)

Apache Spark is an instance of:

- a. Software-as-a-Service (SaaS)
- b. Cloud-as-a-Service (CaaS)
- c. Platform-as-a-Service (PaaS)
- d. Infrastructure-as-a-Service (IaaS)

To `pip install` Python packages so they are usable in Spark jobs, you need to

- e. Run `!pip install` from Colab
- f. Add an EMR bootstrap action
- g. Run `!pip install` from your `%%spark cell`
- h. Run `anaconda` from Colab

<https://tinyurl.com/cis545-lecture-02-16-22>

Recap of Cloud Cluster Management

One type of *platform-as-a-service* – pay-as-you-go clusters with preconfigured software

You'll generally:

- Install Apache Spark + Livy (and Hive for its SQL libraries)
 - *Bootstrap* script lets you install libraries on all nodes
- Configure at least 16GB RAM, 3 nodes – beware you are billed by how long the cluster is running!
- **We'll have a preconfigured CloudFront template for you**

<https://tinyurl.com/cis545-lecture-02-16-22>

How Spark Works on a Cluster

Susan B. Davidson and Zachary G. Ives

University of Pennsylvania

CIS 545 – Big Data Analytics



*Portions of this lecture have been contributed to the OpenDS4All project,
piloted by Penn, IBM, and the Linux Foundation*

<https://tinyurl.com/cis545-lecture-02-16-22>

From SQL to a Spark Query Plan

<https://tinyurl.com/cis545-007>

```
yelp_business_sdf = spark.read.format("csv").option("header",  
"true").load("yelp_business.csv")
```

```
avg_reviews_by_city_sdf = spark.sql(\  
    'select city, avg(stars) as avg_rating '\  
    'from yelp_\  
    'group by city\'  
avg_reviews_by_city_sdf
```

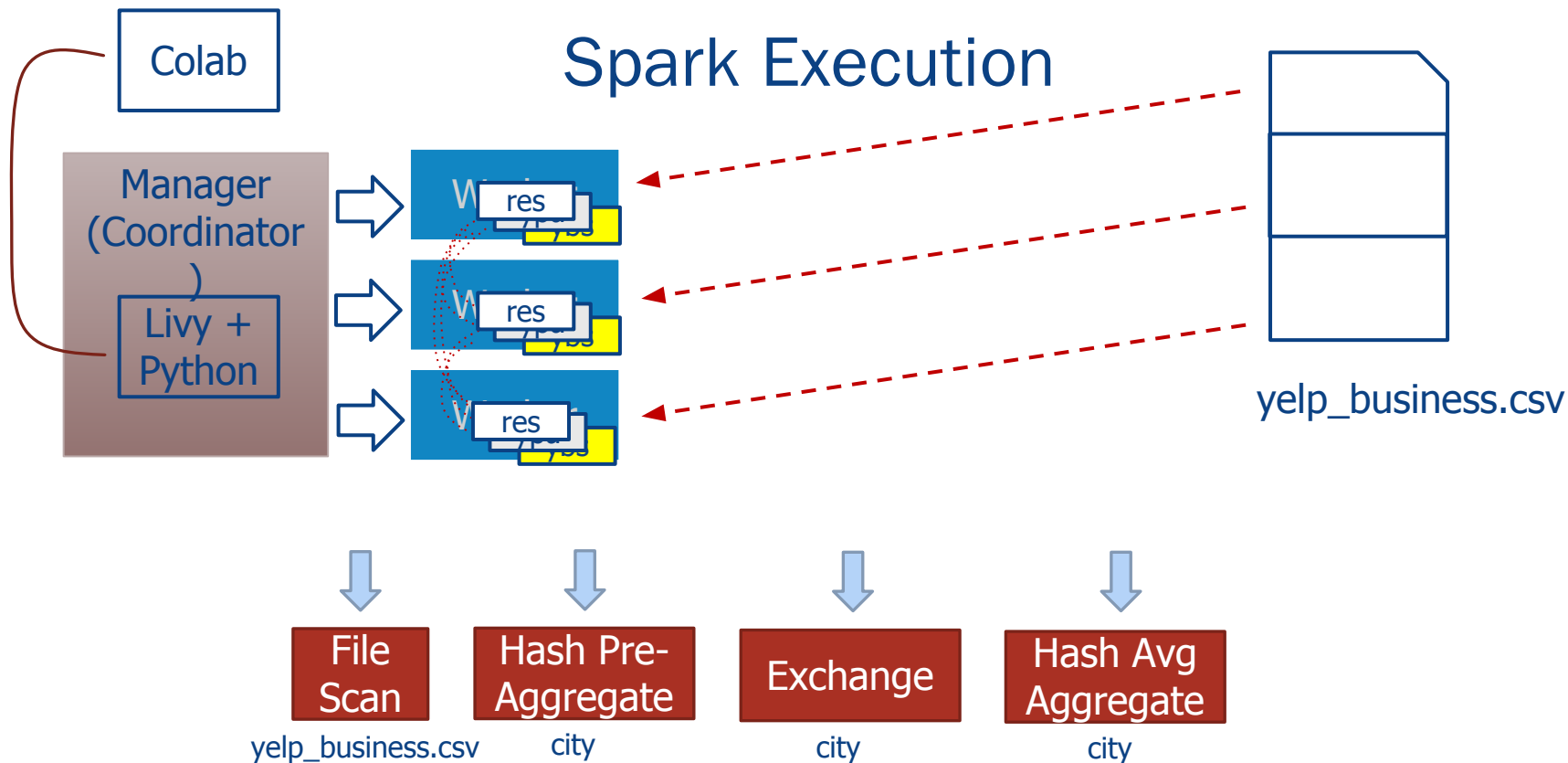
name	neighborhood	address	city	state
Dental by Design	null	4855 E Warner Rd, ...	Ahwatukee	AZ
Stephen Szabo Salon	null	3101 Washington Rd	McMurray	PA
Western Motor Veh...	null	6025 N 27th Ave, ...	Phoenix	AZ
Sports Authority	null	5000 Arizona Mill...	Tempe	AZ
		Ave	Cuyahoga Falls	OH



```
* (2) HashAggregate
+- Exchange hash
  +- * (1) HashAggregate
    [part
  +- FileScan
    Location: InMemoryFileIndex[File:/content/yelp_business.csv],
    PartitionFilters: [], PushedFilters: [], ReadSchema: struct<city:string,stars:string>
```

<https://tinyurl.com/cis545-lecture-02-16-22>

Spark Execution



<https://tinyurl.com/cis545-lecture-02-16-22>

Distributed Joins

```
same_city_sdf = spark.sql(  
    'select b1.name, b2.name from yelp_business b1 join yelp_business b2 '\n'  
    ' on b1.city = b2.city and b1.name <> b2.name')  
'
```

yelp_business

id	name	city	
FYNWN1	Dental by Design	Ahwatukee	Server 0
BADF	My Wine Cellar	Ahwatukee	Server 1
KQPW8	Western Motor Vehicles	Phoenix	Server 0
8DSHNS	Sports Authority	Tempe	Server 1



Sharded by ID

<https://tinyurl.com/cis545-lecture-02-16-22>

Distributed Joins

```
same_city_sdf = spark.sql(  
    'select b1.name, b2.name from yelp_business b1 join yelp_business b2 '\n'  
    ' on b1.city = b2.city and b1.name <> b2.name')  
'
```

yelp_business

id	name	city	
FYNWN1	Dental by Design	Ahwatukee	Server 0
BADF	My Wine Cellar	Ahwatukee	Server 1
KQPW8	Western Motor Vehicles	Phoenix	Server 0
8DSHNS	Sports Authority	Tempe	Server 1



Sharded by ID



Create two
copies, sharded
by city

<https://tinyurl.com/cis545-lecture-02-16-22>

Distributed Joins

```
same_city_sdf = spark.sql(  
    'select b1.name, b2.name from yelp_business b1 join yelp_business b2 '\n'  
    ' on b1.city = b2.city and b1.name <> b2.name')  
'
```

yelp_business

id	name	city	
FYNWN1	Dental by Design	Ahwatukee	Server 0
BADF	My Wine Cellar	Ahwatukee	Server 1
KQPW8	Western Motor Vehicles	Phoenix	Server 0
8DShNS	Sports Authority	Tempe	Server 1

yelp_business

id	name	city	
FYNWN1	Dental by Design	Ahwatukee	Server 0
BADF	My Wine Cellar	Ahwatukee	Server 0
KQPW8	Western Motor Vehicles	Phoenix	Server 1
8DShNS	Sports Authority	Tempe	Server 1



Sharded by ID

Exchange / repartition / shuffle



Sharded by city

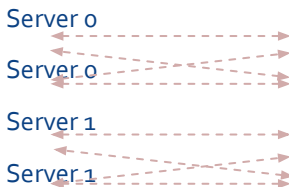
<https://tinyurl.com/cis545-lecture-02-16-22>

Distributed Joins

```
same_city_sdf = spark.sql(
    'select b1.name, b2.name from yelp_business b1 join yelp_business b2 '\
    ' on b1.city = b2.city and b1.name <> b2.name')
```

yelp_business (b1)

id	name	city
FYNWN1	Dental by Design	Ahwatukee
BADF	My Wine Cellar	Ahwatukee
KQPW8	Western Motor Vehicles	Phoenix
8DSHNS	Sports Authority	Tempe



yelp_business (b2)

id	name	city
FYNWN1	Dental by Design	Ahwatukee
BADF	My Wine Cellar	Ahwatukee
KQPW8	Western Motor Vehicles	Phoenix
8DSHNS	Sports Authority	Tempe

Server 0

Server 0

Server 1

Server 1

name	name
My Wine Cellar	Dental by Design
Dental by Design	My Wine Cellar

<https://tinyurl.com/cis545-lecture-02-16-22>

Variation: (Left) Outerjoin

```
same_city_sdf = spark.sql(
  'select b1.name, b2.name from yelp_business b1 left join yelp_business b2 '\
  ' on b1.city = b2.city and b1.name <> b2.name')
```

yelp_business (b1)

id	name	city
FYNWN1	Dental by Design	Ahwatukee
BADF	My Wine Cellar	Ahwatukee
KQPW8	Western Motor Vehicles	Phoenix
8DSHNS	Sports Authority	Tempe

Server 0
Server 0
Server 1

yelp_business (b2)

id	name	city
FYNWN1	Dental by Design	Ahwatukee
BADF	My Wine Cellar	Ahwatukee
KQPW8	Western Motor Vehicles	Phoenix
	Sports Authority	Tempe

Server 0
Server 0
Server 1
Server 1

name	name
My Wine Cellar	Dental by Design
Dental By Design	My Wine Cellar
Western Motor...	NULL
Sports Authority	NULL

<https://tinyurl.com/cis545-lecture-02-16-22>

Minimizing *Shuffle/Exchange* Steps

- Every time we do a join or a group-by, we need the data to be sharded on the key
 - If it isn't, we need to do an *exchange* or *repartition*!
- A good strategy: *amortize* the repartitions across multiple operations if possible!

<https://tinyurl.com/cis545-lecture-02-16-22>

Catalyst: Spark's Query Optimizer Generates the Plans

- Spark's query optimizer seeks to:
 - Estimate how big the input sources are
 - Estimate how many results will be produced in each filter, join, groupby – compare different orderings of operations
- Find the strategy that minimizes the overall cost, including repartitions and join costs

<https://tinyurl.com/cis545-lecture-02-16-22>

Spark Handles Failures!

What happens if one of our worker nodes dies?

Spark re-reads its input data using the other nodes, and re-executes the missing part of the query!

<https://tinyurl.com/cis545-lecture-02-16-22>

Brief Review

<https://canvas.upenn.edu/courses/1636888/quizzes/2771512> (09C)

When Spark runs on a cluster, it creates and executes a Spark query plan when

- a. we execute a cell with a Pandas operation
- b. we execute a cell that invokes an action like `show()` or `save()`
- c. we execute a cell with a dataframe operation like a join
- d. we execute a cell with an SQL query

Given two dataframes **students(id,name)** and **enrolled(course_id,student_id)**, if we execute a query to join on the student IDs, Spark must:

- e. ensure **students** is sharded by **ID** and **enrolled** is sharded by **student_id**, or add exchange operators as needed
- f. perform a hash join within each of the worker nodes, without adding any exchange operators
- g. ensure **students** is sharded by ID and **enrolled** is sharded by **course_id**, or add exchange operators as needed
- h. sort the **enrolled** dataframe by **course_id**

<https://tinyurl.com/cis545-lecture-02-16-22>

Recap

Apache Spark queries are *lazy* to maximize what can be optimized

Upon an *action* like `show()`, the queries are combined and a *plan* is generated – which minimizes cost

Group-by and join require *the data to be sharded on the key* – may need to *exchange or reshuffle or repartition* data

If a worker fails in execution, its work is re-executed

Spark's Catalyst query optimizer seeks to find the minimum-cost plan, but occasionally you may need to manually override it

<https://tinyurl.com/cis545-lecture-02-16-22>

Storing Data on the Cloud

Susan B. Davidson and Zachary G. Ives

University of Pennsylvania

CIS 545 – Big Data Analytics



*Portions of this lecture have been contributed to the OpenDS4All project,
piloted by Penn, IBM, and the Linux Foundation*

<https://tinyurl.com/cis545-lecture-02-16-22>

Where Do We *Put* Our Big Data?

- A cloud file system?
- A cloud NoSQL system?
- A cloud relational DBMS?

<https://tinyurl.com/cis545-lecture-02-16-22>

Key questions

How complex and large is the data and its content?

videos, images; JSON; large CSVs

How will I query my data?

e.g., by pathname, by properties, by features

Do I need transactions?

<https://tinyurl.com/cis545-lecture-02-16-22>

S3 (or GCS) for Storing Large Objects

Buckets (5) [Refresh](#) [Copy ARN](#) [Empty](#) [Delete](#) [Create bucket](#)

Buckets are containers for data stored in S3. [Learn more](#)

[Upload](#) [+ Create folder](#) [Download](#) [Actions](#)

US East (N. Virginia) [Refresh](#)

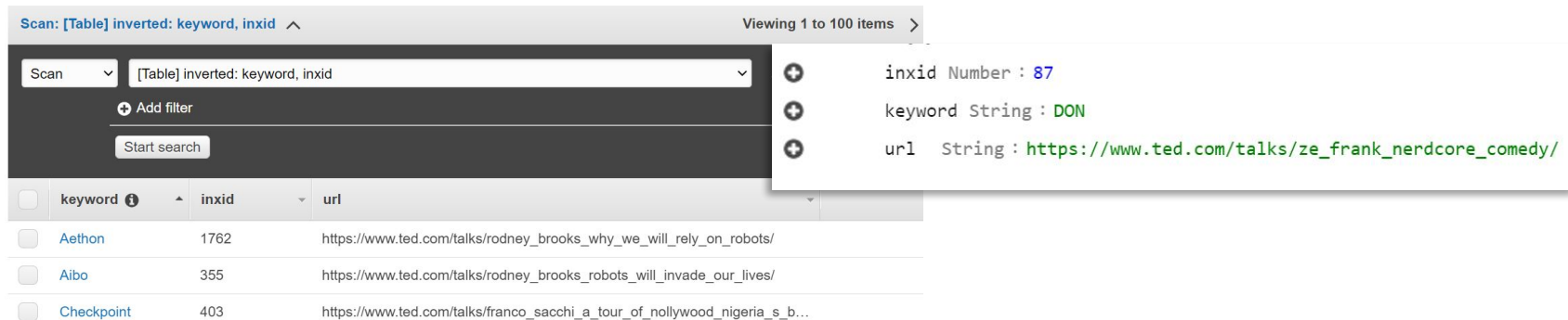
Name	Region
<input type="radio"/> aws-logs-884743372678-us-east-1	US East (N. Virginia) us-east-1
<input type="radio"/> penn-cis545-files	US East (N. Virginia) us-east-1

Name	Last modified	Size	Storage class
<input type="checkbox"/> GrammarandProductReviews.csv	Sep 17, 2020 12:50:33 PM GMT-0400	94.8 MB	Standard
<input type="checkbox"/> Melbourne_housing_extra_data.csv	Mar 30, 2020 8:27:22 AM GMT-0400	2.4 MB	Standard
<input type="checkbox"/> NY_Hospital_Acquired_Infections__Beginning_2008.csv	Sep 12, 2020 7:17:21 PM GMT-0400	4.1 MB	Standard
<input type="checkbox"/> UPDATED_2_airbnb_df.csv	Sep 12, 2020 10:47:11 PM GMT-0400	3.8 MB	Standard

- Amazon S3 supports “buckets” – virtual disk volumes
- Can use “s3a://bucketname/filename” to specify an S3 file
 - For dataframes: `df.write.parquet()`, `sqlContext.read.parquet()`

<https://tinyurl.com/cis545-lecture-02-16-22>

DynamoDB (or BigTable) for Small Object Lookup



The screenshot shows the AWS DynamoDB console interface. At the top, it says "Scan: [Table] inverted: keyword, inxid" and "Viewing 1 to 100 items". Below this is a search bar with a dropdown menu set to "Scan" and a text input field containing "[Table] inverted: keyword, inxid". There is an "Add filter" button and a "Start search" button. Below the search bar is a table with three columns: "keyword", "inxid", and "url". The table contains three rows of data:

keyword	inxid	url
Aethon	1762	https://www.ted.com/talks/rodney_brooks_why_we_will_rely_on_robots/
Aibo	355	https://www.ted.com/talks/rodney_brooks_robots_will_invade_our_lives/
Checkpoint	403	https://www.ted.com/talks/franco_sacchi_a_tour_of_nollywood_nigeria_s_b...

On the right side of the table, there is a tooltip showing the details of the selected item (inxid 87):

- inxid Number : 87
- keyword String : DON
- url String : https://www.ted.com/talks/ze_frank_nerdcore_comedy/

- Given objects in a *map* from keys to hierarchical values – DynamoDB is a good choice
 - Values may be JSON data, dictionaries (max 4KB / field)
- Queries largely limited to lookups by key

<https://tinyurl.com/cis545-lecture-02-16-22>

RDBMSs for Queriable Objects

- Relational DBMSs are best if we want:
 - Complex queries that return *subsets* of data to Spark
 - Atomic updates across tables, in transactions
- Interoperability with the most tools
- Amazon RDS lets us launch PostgreSQL, Oracle, MariaDB, ...

<https://tinyurl.com/cis545-lecture-02-16-22>

Brief Review

<https://canvas.upenn.edu/courses/1636888/quizzes/2771525> (09D)

If we have tabular data that we are retrieving solely by an ID, our best choice(s) for storage are likely to be:

- a. DynamoDB or RDS
- b. neither DynamoDB nor RDS
- c. DynamoDB only
- d. RDS only

If we have satellite photos, we are likely to want to store these on:

- e. RDS
- f. our laptop
- g. DynamoDB
- h. S3

<https://tinyurl.com/cis545-lecture-02-16-22>

Recap

- Our focus in this class: processing big data
- But there are multiple places we can save it:
 - “Large object stores” like S3 – videos, images, large CSVs, large parquet files
 - NoSQL stores like DynamoDB – JSON, simple objects
 - RDBMSs like RDS – tabular data that we’ll query

<https://tinyurl.com/cis545-lecture-02-16-22>

Materialization of Query Results

Susan B. Davidson and Zachary G. Ives

University of Pennsylvania

CIS 545 – Big Data Analytics



*Portions of this lecture have been contributed to the OpenDS4All project,
piloted by Penn, IBM, and the Linux Foundation*

<https://tinyurl.com/cis545-lecture-02-16-22>

When We Have Big Data, We May Need to Make Storage Decisions

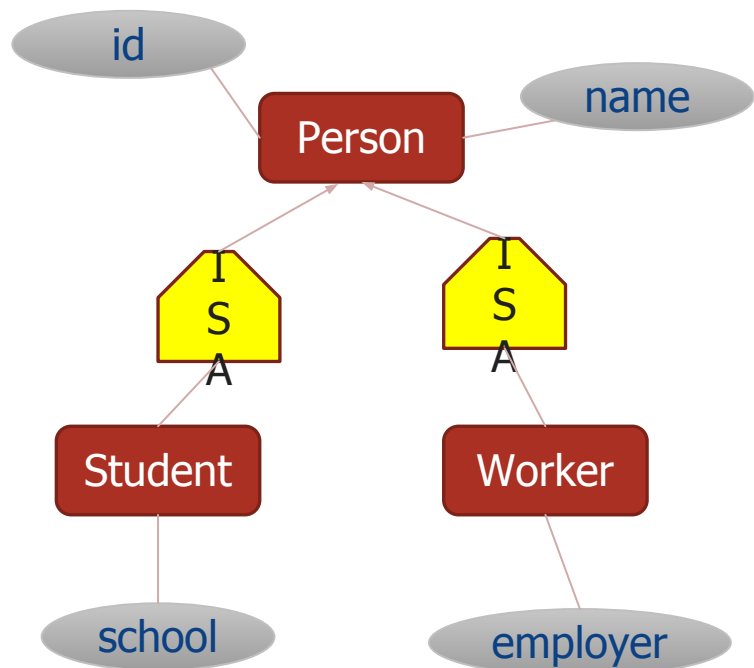
We've seen data with embedded *hierarchy*

- LinkedIn people included lists of education or job experiences
- Key idea: *split these into subtables, explode the lists*
- There's a goal of storing data without redundancy

But: Sometimes portions of data *overlap*, e.g., both parent and subclasses have some common instances

<https://tinyurl.com/cis545-lecture-02-16-22>

An Example of Instances and Subclasses



Person

id	name
123	Ai
456	Jay
789	Kaye

Student

id	school
456	Penn
789	MIT

Worker

id	employer
789	Lutron

<https://tinyurl.com/cis545-lecture-02-16-22>

Materialization

Ideally, our original data is stored without redundancy – this makes it easier to maintain

But as we generate analysis results, we may want to strategically store redundant info! “View materialization”!

Let’s apply to people, students, and workers...

<https://tinyurl.com/cis545-lecture-02-16-22>

Student and Worker are Naturally *Views*

```
CREATE VIEW WorkerPerson(id, name, employer) AS  
  SELECT *  
  FROM Person NATURAL JOIN Worker
```

WorkerPerson

id	name	employer
789	Kaye	Lutron

```
CREATE VIEW StudentPerson(id, name, employer) AS  
  SELECT *  
  FROM Person NATURAL JOIN Student
```

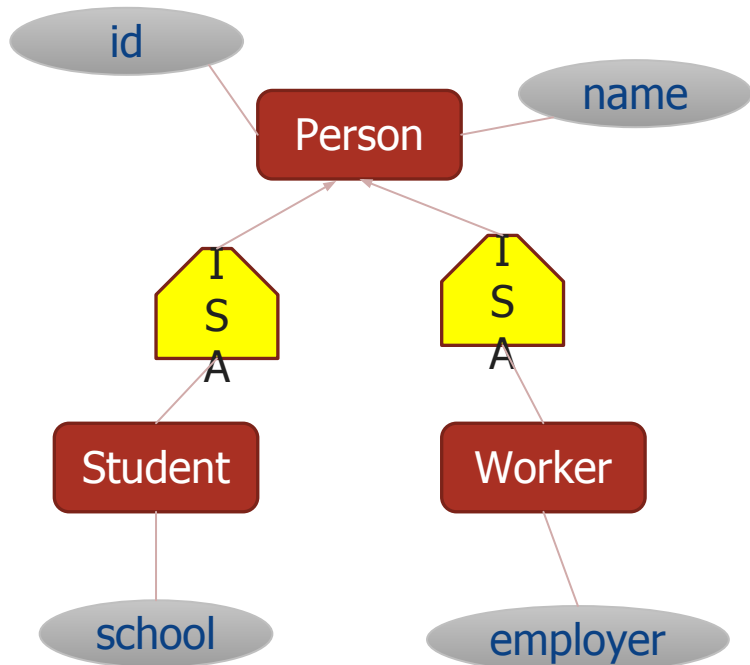
StudentPerson

id	name	school
456	Jay	Penn
789	Kaye	MIT

but views are simply named queries treated as tables...

<https://tinyurl.com/cis545-lecture-02-16-22>

An Example of Instances and Subclasses with Redundancy!



Person

id	name
123	Ai
456	Jay
789	Kaye

Worker

id	employer
789	Lutron

Student

id	school
456	Penn
789	MIT

StudentPerson

id	name	school
456	Jay	Penn
789	Kaye	MIT

WorkerPerson

id	name	employer
789	Kaye	Lutron

<https://tinyurl.com/cis545-lecture-02-16-22>

More Generally...

- In Spark, we can take any Dataframe and **persist** it...

```
same_city_sdf = spark.sql('select b1.name, b2.name as name2  '\n    from yelp_business b1 join yelp_business b2 '\n    ' on b1.city = b2.city and b1.name <> b2.name')\nsame_city_sdf.persist()
```

- Now any time we reference same_city_sdf it will use the stored version!

<https://tinyurl.com/cis545-lecture-02-16-22>

Other Uses for Materialization

- Commonly used subqueries
- Generated reports or hierarchical data
- Recursive computations (we'll see these over graphs)

<https://tinyurl.com/cis545-lecture-02-16-22>

Brief Review

<https://canvas.upenn.edu/courses/1636888/quizzes/2771500> (09E)

If we use inheritance in an E-R diagram, the tables are naturally partitioned such that

- a. we only store instances in the child tables
- b. instances show up in parent and child tables, but columns other than ID are split
- c. the same columns show up in parent and child tables, but instances are split
- d. we repeat both instances and all columns in parent and child tables

View materialization is accomplished by

- e. calling `materialize()` on a dataframe
- f. creating a view in SparkSQL
- g. saving the input CSV
- h. calling `persist()` on a dataframe

<https://tinyurl.com/cis545-lecture-02-16-22>

Recap

- View materialization sacrifices storage (and cost of updating) for query performance
- Very commonly used in big data scenarios
- Can be done by saving a result directly, or by `DataFrame.persist()`

<https://tinyurl.com/cis545-lecture-02-16-22>

Module Wrap-up

- As we scale to bigger and more complex data, need to harness compute clusters
- Spark runs across multiple workers, shuffles data as necessary for joins and grouping
 - Query optimizer seeks to minimize these costs
- We have a series of options for storing our data
- Sometimes it's useful to trade off space for query

<https://ln.yum.com.cn/545-lecture-02-16-22>

More Complex Relationships

- Most of our discussion has been about “direct” relationships
 - Student TAKES a class
 - a student ISA person
- In the real world, lots of transitive relationships!
 - Real and digital social networks, the Internet, road networks, supplier networks, ...
 - Leads to Part 3: graphs!

<https://tinyurl.com/cis545-lecture-02-16-22>