15-319 / 15-619
Cloud Computing

Recitation 3

September 10th & 12th, 2019

Important Notice

- **DON'T EXPOSE YOUR AWS CREDENTIALS!**
  - Github
  - Bitbucket
  - Anywhere public…
- **DON'T EXPOSE YOUR GCP CREDENTIALS!**
- **DON'T EXPOSE YOUR Azure CREDENTIALS!**
  - ApplicationId, ApplicationKey
  - StorageAccountKey, EndpointUrl

Reflection

- Conceptual content on OLI
  - Modules 1, 2, Quiz 1
- Project theme - **Sequential Analysis**
  - **Wiki Data Processing Task**
    - Sequential Analysis of **100s MB** of wikipedia data, adopt Test Driven Development (TDD)
  - **Data Analytics Task**
    - Write analytics code on Jupyter Notebook using the pandas library
  - Identify the limitations of sequential programs

This Week

- **Quiz 2 (OLI Modules 3 & 4)**
  - Due on **Friday**, Sep 13, 2019, 11:59PM ET
- **Project 1.2**
  - Due on **Sunday**, Sep 15, 2019, 11:59PM ET
- **Project 1.1 Reflection Feedback**
  - Due on **Sunday**, Sep 15 2019, 11:59PM ET

Guideline for Project Reflection

- Describe your approach in solving each task in this project
  - Please share your
    - approach, challenges faced, how you overcame issues, and lessons learned.
  - However, please:
    - Do not share your code or pseudocode
    - Do not share details about your solution

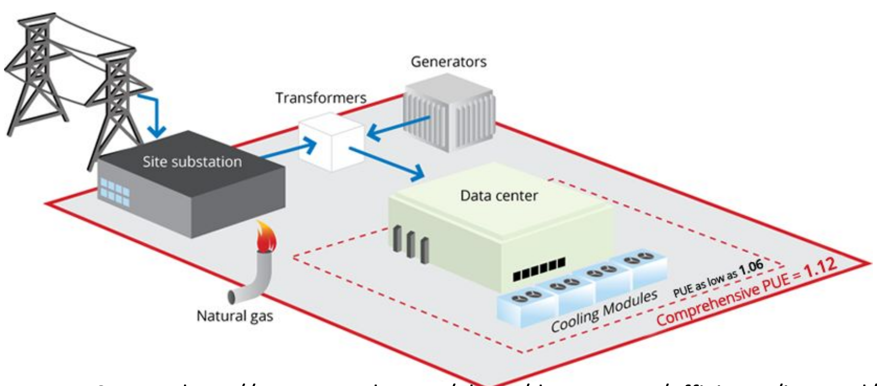
Module 3: Data Center Trends

- Definition & Origins
  - Infrastructure dedicated to housing computer and networking equipment, including power, cooling, and networking.
- Growth
  - Size (No. of racks and cabinets)
  - Density
- Efficiency
  - Servers
  - Server Components
  - Power
  - Cooling

Facebook data center

Module 4: Data Center Components

- IT Equipment
  - Servers : rack-mounted
    - Motherboard
    - Expansion cards
  - Type of Storage
    - Direct attached storage (DAS)
    - Storage area network (SAN)
    - Network attached storage (NAS)
  - Networking
    - Ethernet, protocols, etc.
- Facilities
  - Server room
  - Power (distribution)
  - Cooling
  - Safety



Source: http://www.google.com/about/datacenters/efficiency/internal/

Project 1

- Identify Trending Topics on Wikipedia
  - Use the hourly pageviews dataset.
- Project 1.1: (Last Week)
  - Find trends from a single hour of data.
- Project 1.2: (This Week)
  - Find trends with a 30-day dataset using MapReduce.
    - Data from March 8 to April 6 in 2018

Limitations of sequential programs

- Your data-preprocessing program might work well with an hourly dataset, but will fall short to process a large dataset
  - Might take too long
  - Might run out of resources
- Methods to process a large dataset
  - Sequential program might not scale ⇒ a parallel solution
  - A single EC2 machine might not have adequate memory and computational capabilities either ⇒ a large distributed cluster
- Challenges to overcome for your program to work in a distributed system
  - How would you partition and distribute the tasks and data?
  - How would the nodes communicate and collaborate?
  - What if a node fails?

The MapReduce programming model

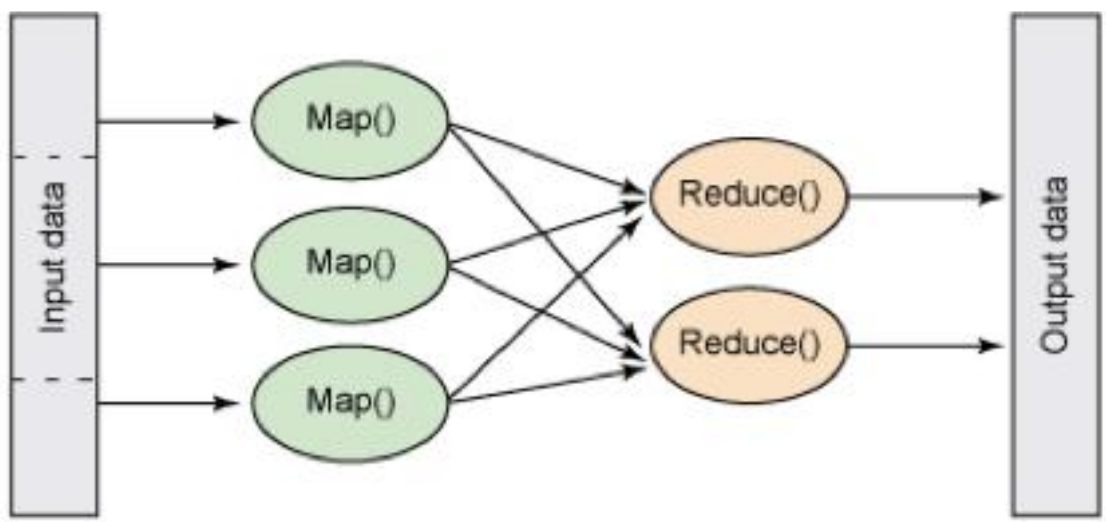
- The MapReduce programming model simplifies parallel processing by abstracting away the complexities involved in working with distributed systems
  - Data partition and distribution
  - Management of communication across nodes
  - Deal with unreliable hardware and software

The MapReduce programming model

- Handling failure gracefully
  - Failure of a single machine will not cause the failure of the whole job
  - A task failure on one node can be resolved by rerunning the task on other nodes
- Reduce the communication cost
  - Data is stored in a distributed manner with replication
  - Exploiting data locality
- Easy to program
  - The minimal code you need to implement is only the map and reduce functions

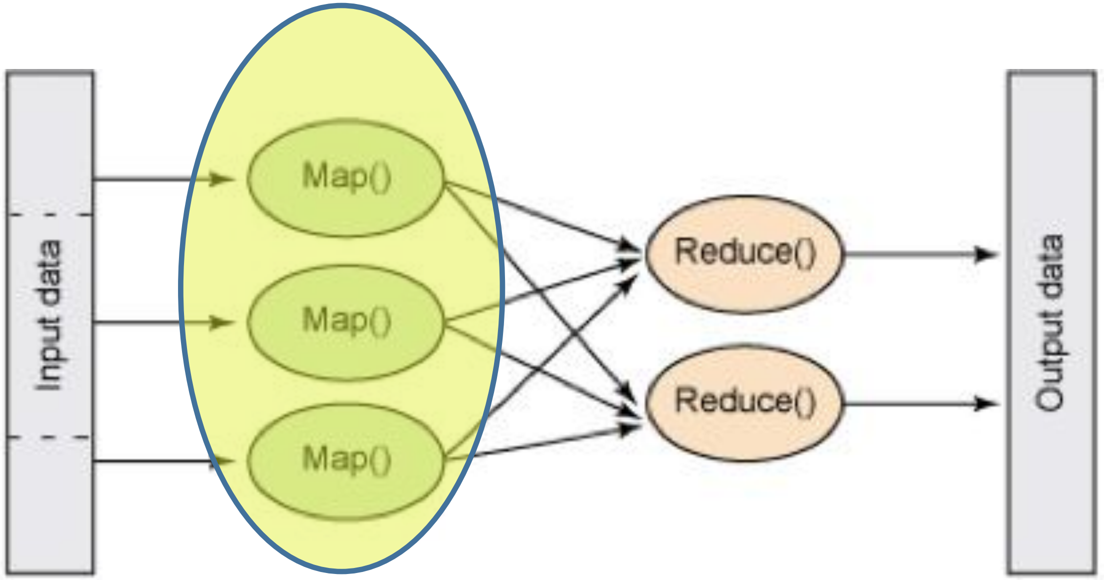
Overview of MapReduce

- Map: Process the input data in chunks and **in parallel**
- Shuffle and sort
- Reduce: Aggregate or summarize intermediate data **in parallel** and output the result



The Map phase in MapReduce

- Map `map(k1,v1) --> list(k2,v2)`
  - Map function takes input as Key-Value pairs `k1,v1`.
  - The map function produces zero or more output Key-Value pairs for one input pair. `list(k2,v2)`



The Map Phase and Intermediate Data

- Map
  - `map(k1,v1) --> list(k2,v2)`

If the input is a file, the input Key-Value pair could represent a line in the file

- keys are the position in the file
- values are the text of the line

`k2,v2` is called “intermediate key-value pair” because it is

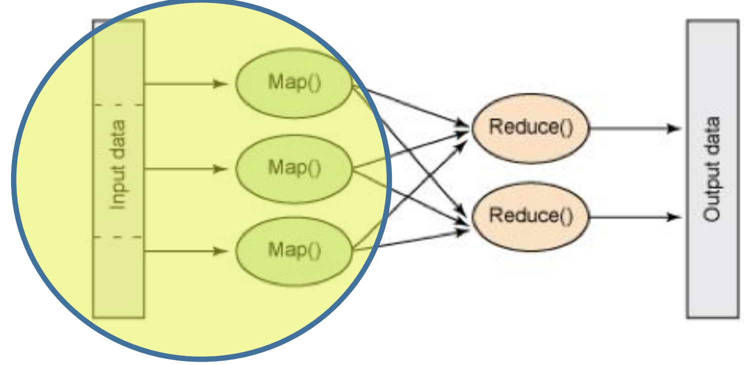
- the output of the Mapper
- the input of the Reducer

Word Count Example:

- Input ⇒ Word Count ⇒ output

- Content of one or more input files:
  - cat cow
  - duck
  - dog cat
  - cat
- Output:
  - cat, 3
  - cow, 1
  - dog, 1
  - duck, 1

Map in Word Count



- Map in the Word Count Example

Input: file1.txt

cat cow

duck

dog cat

`k1,v1 pairs:`

(pos, "cat cow")
(pos, "duck")
(pos, "dog cat")

Mapper1

`k2,v2 pairs:`

(cat, 1)
(cow, 1)
(duck, 1)
(dog, 1)
(cat, 1)

Input: file2.txt

cat

`k1,v1 pairs:`

(pos, "cat")

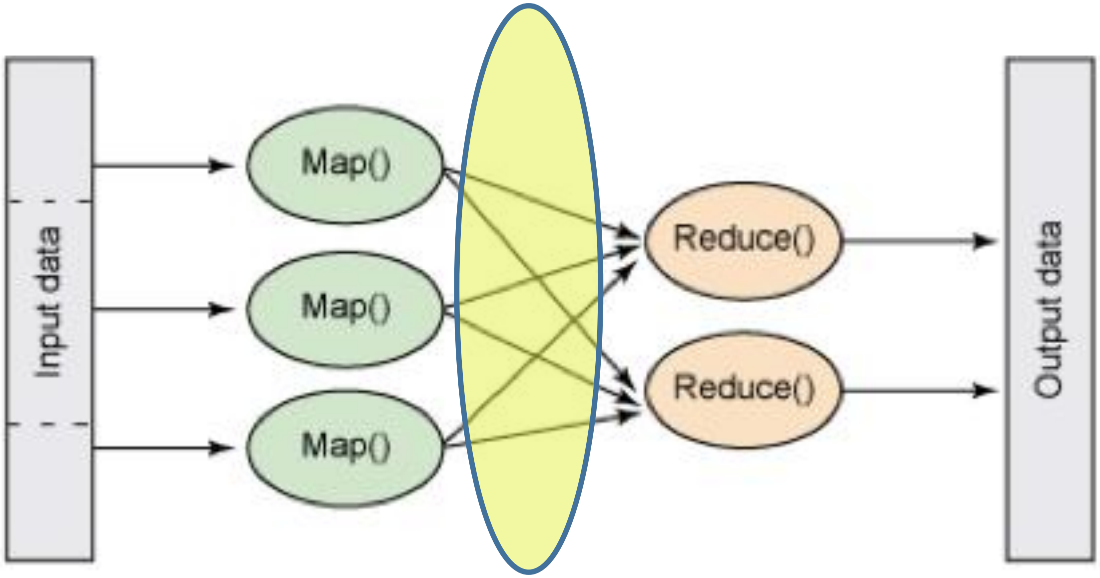
Mapper2

`k2,v2 pairs:`

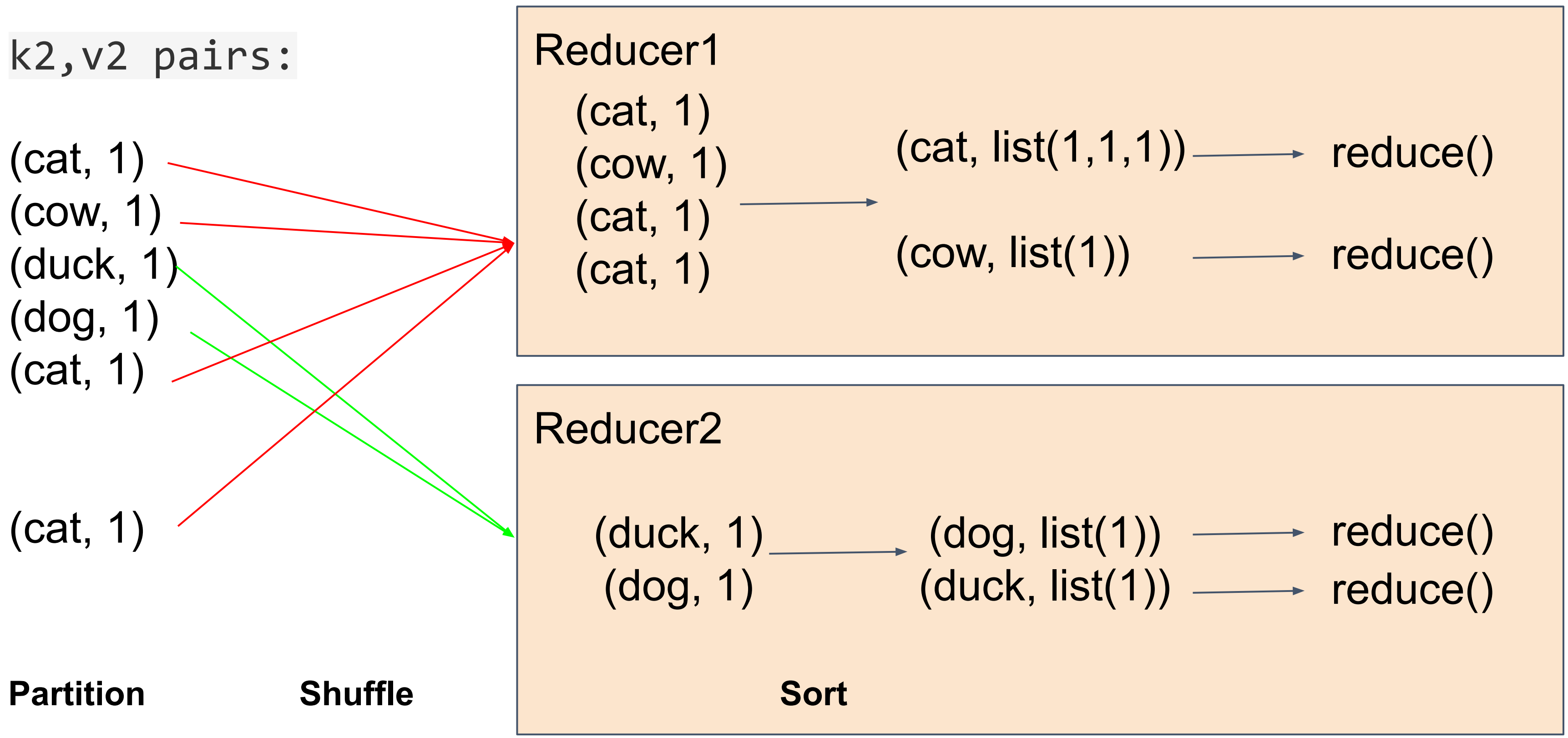
(cat, 1)

The Shuffle and Sort in MapReduce

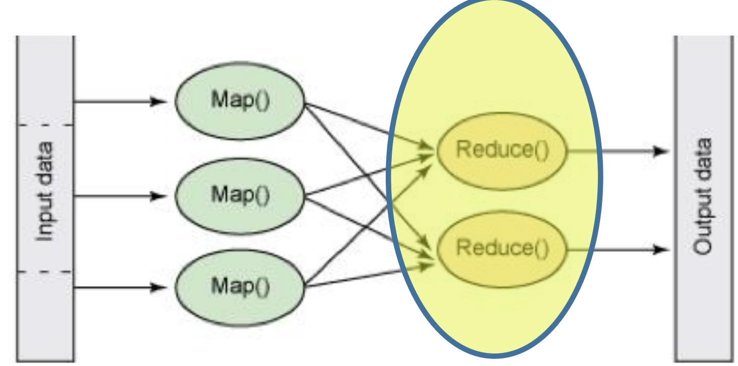
- Shuffle: transfers data from the mappers to the reducers
- Sort: sort intermediate key-value pair by key



Shuffle and sort in the Word Count Example



The Reduce Phase in MapReduce



- Reduce:
  - `reduce(k2, list(v2)) --> list(v3)`
- The reduce function is called **once for each unique key** emitted from the Mapper.
- The Reducer **has an iterator for all values for each key**.
- Produce the output to the directory defined by the MapReduce job.

Reduce in the Word Count Example

Reducer1

(cat, list(1,1,1)) → reduce() → (cat, 3)

(cow, list(1)) → reduce() → (cow, 1)

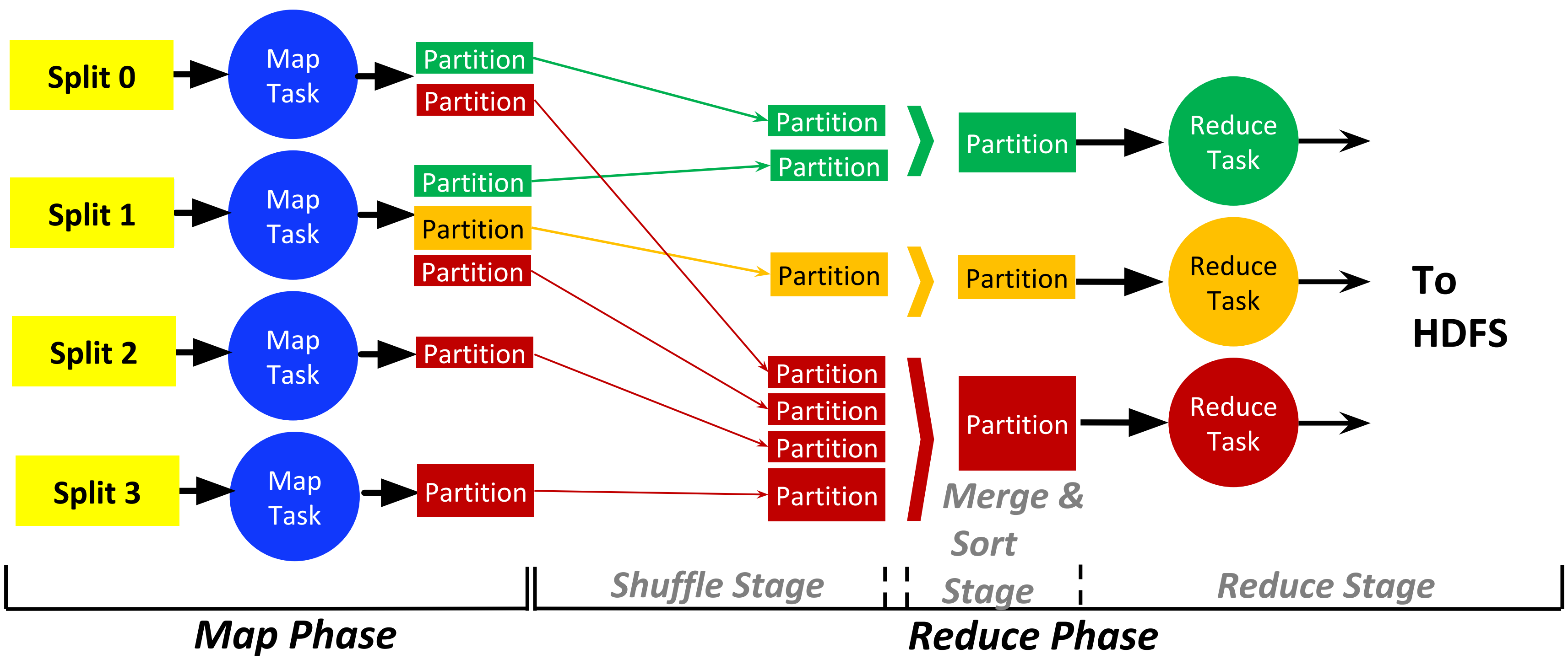
Reducer2

(dog, list(1)) → reduce() → (dog, 1)

(duck, list(1)) → reduce() → (duck,1)

MapReduce In a Nutshell

- MapReduce incorporates two phases
  - Map Phase
  - Reduce phase



Parallelism in MapReduce

- Mappers run in parallel, processing different input splits and creating intermediate Key-Value pairs
- Reducers also run in parallel, each working on a set of keys based on the partitioning function
    - By default, the partitioning function is a hash function
- Although the shuffle can start early, however, the reduce function cannot start until all mappers finish and all intermediate data is shuffled

MRUnit: TDD for MapReduce

- MRUnit is a unit test framework for MapReduce
- Allows you to define your input and expected output for the map and reduce functions
- This will allow you to test your map and reduce functions

Using MRUnit

- Tests supported
  - Map Test to test map()
  - Reduce Test to test reduce()
  - MapReduce Test to test both
- Steps to create Map Test
  - Step 1: Create your Mapper
  - Step 2: Create map test using MRUnit
  - Step 3: Set the input and output records
  - Step 4: Implement your map function
  - Step 5: Run locally to evaluate the test

MRUnit: Example map() test

```
// the test code is under the test source folder, similar to JUnit 5 test code
// run “mvn test” to run the test
public class WordCountMapTest extends TestCase {
    @Test
    public void testWordCountMapper() throws IOException {
        driver.withInput(new Text(""), new Text("cat cat dog"))
                .withOutput(new Text("cat"), new VIntWritable(1))
                .withOutput(new Text("cat"), new VIntWritable(1))
                .withOutput(new Text("dog"), new VIntWritable(1))
                .runTest(false);
    }
}
```

Test the MR workflow

- Use LocalJobRunner to test the MR jobs
  - Runs the MapReduce workflow in memory locally
- Steps to follow:
  - Define the configurations similar to the configurations of a real MapReduce job
    - Input path, output path
    - Mapper class, reducer class
    - etc.
  - Test if the job can be successful

Troubleshooting EMR and MapReduce

- As you run the jobs with the large dataset, you can still run into errors despite the tests because of:
  - Resource limit, e.g., OutOfMemory
  - Malformed input data
- Aggregate the distributed log chunks into a single file will enable you to search all logs at once
- To retrieve the aggregated logs, run the following command on the master node

```
yarn logs -applicationId <applicationId>
```

- The first 3 questions in runner.sh will help you practice how to use grep to search the log files

Task 1: Inverted Index in MapReduce

- An index maps the words to the file where they occur.

Input:
file1.txt

cat cow

duck

dog cat

Input:
file2.txt

cat dog

Output:

cat	(file1.txt,file2.txt)
cow	(file1.txt)
dog	(file1.txt,file2.txt)
duck	(file1.txt)

Your Task

Implement Inverted Index with MapReduce using TDD

- A worked example of WordCount and the test cases are provided for you to learn from
- We provide you with the test cases for Inverted Index
  - to test the implementation of map and reduce functions
  - to test if the MapReduce application can run successfully w/ LocalJobRunner on a local dataset
- Your task is to pass the test cases
- If you can pass the test cases, the LocalJobRunner will generate the output to a local path

Running a Hadoop MR Job from the Command Line

- Create a cluster as per the AWS EMR section
  - provision via Terraform
  - SSH into the master node
- Run the MapReduce job in hadoop

```
> yarn jar project1.jar
edu.cmu.scs.cc.project1.WordCount input-path
output-path
```

Task 2: Wikipedia MapReduce application

- Put what you have learned together
- Design and implement a MapReduce application to:
  - Filter out records based on the filtering rules in the data filtering task. Reuse your code.
  - Get the input filename from within a Mapper
  - Aggregate the pageviews from hourly views to daily views
  - Calculate the total pageviews for each article
  - Print the popular article that has over 100,000 page-views (100,000 excluded)

Task 3: Data Analysis with Pandas

- Now that you have filtered and aggregated the monthly data, you are ready to analyze the data to answer some interesting analytics questions.

Project 1.2 Workflow

- Launch an EC2 instance with a specified AMI
- Provision EMR cluster(s) and finish tasks:
  - Inverted Index in MapReduce
  - Wikipedia MapReduce
- Complete and run the script
  - `/home/clouduser/Project1/runner.sh`
  - Answer a set of questions by providing the code inside data_analysis.ipynb
- Submit your code for grading
  - Complete the `references` file in JSON format
  - Execute `submitter` to submit your code
- Finish Project Reflection (graded) before the deadline

Grading of Your Projects

- Code submissions are auto-graded
- We will grade all the code (both auto and manually)
- We auto grade your coding style, which is worth 5 points
- Coding style will be manually graded.
  - high quality code
  - sufficient comments
  - self-explanatory code
  - modularize code

Reminder: Deadlines

- **Quiz 2 (OLI Modules 3 & 4)**
  - Due on **Friday**, Feb 1, 2019, 11:59PM ET
- **Project 1.2**
  - Due on **Sunday**, Feb 3, 2019, 11:59PM ET
- **Project 1.1 Reflection Feedback**
  - Due on **Sunday**, Feb 3, 2019, 11:59PM ET