CSE 239 – Advanced Cloud Computing

Homework Assignment #2

Guidelines

- This homework is due by 11:59am PDT on 11/02/25.
    - Note: You may use up to **3 total grace days** during the quarter.
- Submit this homework as a PDF on Gradescope.
- Please ensure the homework is clearly legible.
- **Legibility:**
    - Typed submissions are preferred.
    - Scanned handwritten work is acceptable only if clearly legible.
- **Explain your work**. No credit will be be given for answers without explanations.
- This assignment is out of 100 points.
- This assignment may be done in pairs of 2 students, but each student must submit their own solutions, and write down the names of their collaborators.

Question 1 (100 points) – Dockerizing a MapReduce-style Word Count

As you noticed from the last homework, parallel systems leverage multi-cores to improve the process time of large-scale applications. However, MapReduce is a programming model in which a problem is divided into smaller units of work that can be executed in parallel not only across multiple cores, but more generally, across *multiple machines*.

In this assignment, you'll build a MapReduce system that runs inside Docker containers. You'll implement a worker process that calls application Map and Reduce functions and handles reading and writing files, and a coordinator process that hands out tasks to workers and copes with failed workers.

Your job is to implement a distributed MapReduce, consisting of two programs, the `coordinator` and the `worker`. There will be just one coordinator process, and one or more worker processes executing tasks in parallel. In a real system the workers would run on many different machines, but for this lab you'll run them all on many Docker containers that will represent machines. The workers should communicate with the coordinator (via RPC), which will aggregate their completed tasks. You must use RPyC to implement the coordination between workers and coordinator. Each worker process will: (i) receive a task from the coordinator, (ii) read the task's input from one or more files, (iii) execute the task, write the task's output to one or more files, and again ask the coordinator for a new task. The coordinator should notice if a worker hasn't completed its task in a reasonable amount of time (for this lab, use 20 seconds), and give the same task to a different worker.

Requirements

1. The solution should implement a **`Coordinator`** and a **`Worker`**, each in its own Docker container. Note that multiple workers should be used (3+). Communication between Coordinator and Workers must be implemented via RPyC (Remote Python Calls), and must occur via defined hostnames (e.g., “`worker-1`”, “`coordinator`”) resolved through Docker networking.
2. Coordinator should allow the user to specify a URL to download a dataset. Assume it can only download UTF-8 files.
3. Workers will periodically check **or** receive Tasks with/from the Coordinator via RPyC. A Task can either be a Map or Reduce task.
4. Map tasks produce results that may be either returned to the Coordinator, or buffered in memory in the Worker machine. The Map Worker is also responsible for partitioning the intermediate pairs into its respective Regions. This is done using a partitioning function. When a Worker completes a Map Task, it communicates with the Coordinator.
5. Reduce tasks are responsible for reducing the intermediate value keys found in all the Worker containers. These are coordinated with Workers by the Coordinator.
6. The Coordinator aggregates the final outputs after all Reduce Workers return their results. If an implementation allows concurrent updates to a shared state (e.g., a global counter or store), those updates must be mutually exclusive (e.g., using locks, semaphores, or transactional primitives) to ensure correctness under concurrent access.
7. Map Tasks performed by a failed container need to be reassigned. If a Worker fails or becomes unresponsive, its assigned Map (or Reduce) Task must be reassigned to another available Worker. Failure can be assumed if a Worker exceeds a predefined

timeout period for completing its assigned Task. The Coordinator is responsible for detecting such failures and rescheduling tasks.
8. The Coordinator can exit when all Map and Reduce Tasks have finished.
9. Number of Map and Reduce Workers should be configurable via command-line arguments or environment variables.

As mentioned, your implementation should be split into two files: `coordinator.py and` `worker.py`. Below we illustrate how each may look like:

RPyC MapReduce Schematic

```
                    +----------------------+
                    |    COORDINATOR       |  ← Docker Container
                    +----------------------+
                     |     |       |      |
            map() →  |     |       |      |  ← reduce() [Docker networking]
                     ↓     ↓       ↓      ↓
        +----------------+  +----------------+  +----------------+
        |   WORKER 1     |  |   WORKER 2     |  |   WORKER 3     |   ← Docker Containers
        |  map_service() |  |reduce_service()|  |reduce_service()|
        +----------------+  +----------------+  +----------------+
```

`coordinator.py`:

```python
import rpyc
import string
import collections
import itertools
import time
import operator
import glob

WORKERS = [
    ("docker-worker1", 18861),
    ("docker-worker2", 18861),
    ("docker-worker3", 18861)
]

def mapreduce_wordcount(text):
    # 1. Split text into chunks
    # 2. Connect to workers
    # 3. MAP PHASE: Send chunks and get intermediate pairs
    # 4. SHUFFLE PHASE: Group intermediate pairs by key
    # 5. REDUCE PHASE: Send grouped data to reducers
    # 6. FINAL AGGREGATION
    return total_counts

def split_text(text, n):

def partition_dict(d, n):

def download(url='https://mattmahoney.net/dc/enwik9.zip'):
    """Downloads and unzips a wikipedia dataset in txt/."""

if __name__ == "__main__":
    # DOWNLOAD AND UNZIP DATASET
    text = download(sys.argv[1:])

    start_time = time.time()
    input_files = glob.glob('txt/*')
    word_counts = mapreduce_wordcount(input_files)
    print('\nTOP 20 WORDS BY FREQUENCY\n')
    top20 = word_counts[0:20]
    longest = max(len(word) for word, count in top20)
    i = 1
    for word, count in top20:
        print('%s.\t%-*s: %5s' % (i, longest+1, word, count))
        i = i + 1

    end_time = time.time()
    elapsed_time = end_time - start_time
    print("Elapsed Time: {} seconds".format(elapsed_time))
```

worker.py
```python
import rpyc

class MapReduceService(rpyc.Service):
    def exposed_map(self, text_chunk):
        """Map step: tokenize and count words in text chunk."""

    def exposed_reduce(self, grouped_items):
        """Reduce step: sum counts for a subset of words."""

if __name__ == "__main__":
    from rpyc.utils.server import ThreadedServer
    t = ThreadedServer(MapReduceService, port=18861)
    t.start()
```

A **`Dockerfile`** and a **`docker-compose.yml`** (files that will respectively build and run your image) should be provided through a public Git repository, along with a `README.md` explaining how to deploy your code (including how to download/clone it). The content of this **`README.md`** should be pasted below. Implement a `download()` to fetch the enwik9.zip dataset, which should unzipped in the **`txt/`** directory (skip downloading a file if it already exists). You may test your deployment using the dataset enwik8.zip (~30 MB). When executed, the script outputs the 20 most frequent words along with the total execution time.

a) [10 points] Execution Time

At the end of execution, given the number of containers, the Coordinator should reports the total runtime. Record the elapsed time below, according to the number of containers:

Elapsed time to run on your computer (1 worker container):	37	seconds
Elapsed time to run on your computer (2 worker containers):	41	seconds
Elapsed time to run on your computer (4 worker containers):	50	seconds
Elapsed time to run on your computer (8 worker containers):	63	seconds

b) [10 points] Scaling

Is the improvement with increasing number of containers similar to the improvement you achieved in homework 1? Explain.

In this situation, we saw (thanks to added metrics) that map and reduce phase are improved by adding more containers. But the overall time is worst because of the aggregation part is longer each time we add more containers.

c) [80 points] README.md

Paste here the content of your README.md. The file should contain all commands to deploy, build, and run your Docker images that will execute your RPyC-MapReduce implementation. Note all requirements listed above.

README.md: