Antoine Gaton

Professor Rida Moustafa

CS356: Foundations of Big Data Analytics

30 June 2024

***Unit 3 - Individual Project***

* ***Python Program***

| ***""" Script for Key-Value Store Operations  Description: This script demonstrates a basic implementation of a key-value store using a dictionary in Python. The dictionary contains 50 state-capital pairs as key-value entries. It includes functions to enumerate the contents of the key-value pairs, list all keys, list all values, and replace the value of a specified key.  Author: Antoine Gaton Email: antoine.gaton@student.ctuonline.edu Date: June 30, 2024  Dependencies: - None (built-in Python libraries are used)  Usage: 1. Ensure you have Python installed on your system. 2. Run the script:  python key\_value\_store.py  Dictionary Data: - The dictionary contains 50 state-capital pairs for demonstration purposes. """  # Creating a dictionary with 50 state-capital pairs states\_capitals = {  'Alabama': 'Montgomery',  'Alaska': 'Juneau',  'Arizona': 'Phoenix',  'Arkansas': 'Little Rock',  'California': 'Sacramento',  'Colorado': 'Denver',  'Connecticut': 'Hartford',  'Delaware': 'Dover',  'Florida': 'Tallahassee',  'Georgia': 'Atlanta',  'Hawaii': 'Honolulu',  'Idaho': 'Boise',  'Illinois': 'Springfield',  'Indiana': 'Indianapolis',  'Iowa': 'Des Moines',  'Kansas': 'Topeka',  'Kentucky': 'Frankfort',  'Louisiana': 'Baton Rouge',  'Maine': 'Augusta',  'Maryland': 'Annapolis',  'Massachusetts': 'Boston',  'Michigan': 'Lansing',  'Minnesota': 'Saint Paul',  'Mississippi': 'Jackson',  'Missouri': 'Jefferson City',  'Montana': 'Helena',  'Nebraska': 'Lincoln',  'Nevada': 'Carson City',  'New Hampshire': 'Concord',  'New Jersey': 'Trenton',  'New Mexico': 'Santa Fe',  'New York': 'Albany',  'North Carolina': 'Raleigh',  'North Dakota': 'Bismarck',  'Ohio': 'Columbus',  'Oklahoma': 'Oklahoma City',  'Oregon': 'Salem',  'Pennsylvania': 'Harrisburg',  'Rhode Island': 'Providence',  'South Carolina': 'Columbia',  'South Dakota': 'Pierre',  'Tennessee': 'Nashville',  'Texas': 'Austin',  'Utah': 'Salt Lake City',  'Vermont': 'Montpelier',  'Virginia': 'Richmond',  'Washington': 'Olympia',  'West Virginia': 'Charleston',  'Wisconsin': 'Madison',  'Wyoming': 'Cheyenne' }  # Function to enumerate the contents of key-value pairs def enumerate\_contents(dictionary):  """  Prints the key-value pairs in the dictionary.   Parameters:  dictionary (dict): The dictionary to enumerate.  """  for key, value in dictionary.items():  print(f'The capital of {key} is {value}')  # Function to list all keys in the dictionary def list\_keys(dictionary):  """  Lists all keys in the dictionary.   Parameters:  dictionary (dict): The dictionary whose keys are to be listed.   Returns:  list: A list of keys in the dictionary.  """  return list(dictionary.keys())  # Function to list all values in the dictionary def list\_values(dictionary):  """  Lists all values in the dictionary.   Parameters:  dictionary (dict): The dictionary whose values are to be listed.   Returns:  list: A list of values in the dictionary.  """  return list(dictionary.values())  # Function to replace the value of key number "1" def replace\_value(dictionary, key, new\_value):  """  Replaces the value of the specified key with a new value.   Parameters:  dictionary (dict): The dictionary to update.  key: The key whose value is to be replaced.  new\_value: The new value to assign to the key.  """  if key in dictionary:  dictionary[key] = new\_value  else:  print(f'Key {key} not found in the dictionary.')  # Main execution if \_\_name\_\_ == "\_\_main\_\_":  # Enumerate the contents of the dictionary  print("Enumerating contents of the dictionary:")  enumerate\_contents(states\_capitals)    print("\n------------------------------------------\n")    # List all keys  print("List of all keys:")  print(list\_keys(states\_capitals))    print("\n------------------------------------------\n")    # List all values  print("List of all values:")  print(list\_values(states\_capitals))   print("\n------------------------------------------\n")   # Replace value of key 'Alabama' with a new value 'New Montgomery'  print("Replacing value of key 'Alabama' with 'New Montgomery':")  replace\_value(states\_capitals, 'Alabama', 'New Montgomery')  print("Updated dictionary:")  enumerate\_contents(states\_capitals)*** |
| --- |

* ***Discussion***
  + *How could this program operate in a distributed environment where the dictionary is spread across a cluster of nodes?*
    - In a distributed environment, the dictionary can be divided into smaller chunks and distributed across multiple nodes in a cluster. Each node can handle a portion of the key-value pairs. A framework like Hadoop's MapReduce can be used to perform operations on these distributed key-value pairs.
* **Map Phase:** The dictionary is split, and each part is sent to different nodes. The nodes perform the mapping function (enumerate contents, list keys, list values, etc.) on their portion of the data.
* **Reduce Phase:** The results from all nodes are aggregated to produce the final output.
  + *What would be the limitations of this architecture?*
    - Limitations of this architecture are the following:
      * **Network Latency:** Communication between nodes can introduce latency.
      * **Data Consistency:** Ensuring data consistency across nodes can be challenging.
      * **Fault Tolerance:** Handling node failures and ensuring the system remains operational can be complex.
      * **Overhead:** The overhead of managing a distributed system can outweigh the benefits for smaller datasets.
  + *Is there a threshold for the amount of data and the overhead of operating on a cluster?*
    - The threshold at which it becomes beneficial to use a distributed system depends on various factors such as the size of the data, the complexity of operations, and the available infrastructure. Generally, distributed systems are advantageous for very large datasets (terabytes or more) where single-node processing becomes impractical. The overhead of maintaining a cluster includes managing distributed storage, ensuring fault tolerance, and coordinating between nodes, which should be justified by the need for parallel processing and scalability.