Python Implementation

1. Checkpoint/Restart Mechanism

This snippet simulates creating checkpoints and restarting from the last saved checkpoint.

1. import pickle

2. import os

3.

4. *# Simulated component tree computation state*

5. computation\_state = {"step": 0, "results": []}

6.

7. def create\_checkpoint(state, filename="latest\_checkpoint.pkl"):

8. """Save the current state to a file."""

9. with open(filename, "wb") as f:

10. pickle.dump(state, f)

11. print(f"Checkpoint created at step {state['step']}.")

12.

13. def restart\_from\_checkpoint(filename="latest\_checkpoint.pkl"):

14. """Load the last saved state if exists."""

15. if os.path.exists(filename):

16. with open(filename, "rb") as f:

17. return pickle.load(f)

18. return None

19.

20. def simulate\_computation():

21. """Simulate the computation process with checkpointing."""

22. state = restart\_from\_checkpoint()

23. if not state:

24. state = computation\_state

25.

26. for \_ in range(10): *# Simulate 10 computation steps*

27. state["step"] += 1

28. state["results"].append(state["step"] \* 2)

29.

30. if state["step"] % 3 == 0: *# Create a checkpoint every 3 steps*

31. create\_checkpoint(state)

32.

33. print(f"Processing step {state['step']}, results: {state['results']}")

34.

35. simulate\_computation()

36.

**Expected Output:**

Given the simulation's setup, the output will display progress messages for each computation step, including when a checkpoint is created. If the simulation runs without interruption, it will create checkpoints at steps 3, 6, and 9. Here's an example of what the output might look like when the simulation runs for the first time:

1. Processing step 1, results: [2]

2. Processing step 2, results: [2, 4]

3. Processing step 3, results: [2, 4, 6]

4. Checkpoint created at step 3.

5. Processing step 4, results: [2, 4, 6, 8]

6. Processing step 5, results: [2, 4, 6, 8, 10]

7. Processing step 6, results: [2, 4, 6, 8, 10, 12]

8. Checkpoint created at step 6.

9. Processing step 7, results: [2, 4, 6, 8, 10, 12, 14]

10. Processing step 8, results: [2, 4, 6, 8, 10, 12, 14, 16]

11. Processing step 9, results: [2, 4, 6, 8, 10, 12, 14, 16, 18]

12. Checkpoint created at step 9.

13. Processing step 10, results: [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]

If the simulation experiences a "failure" after step 7 and is restarted, it will resume from the last checkpoint saved at step 6, thanks to the checkpoint/restart mechanism. Assuming the script simulates a restart by calling simulate\_computation() again (and the state from step 6 was the last saved state), the output upon restart might begin like this:

1. Restarting from checkpoint...

2. Processing step 7, results: [2, 4, 6, 8, 10, 12, 14]

3. ...

4.

2. Data Replication for Fault Tolerance

This snippet simulates data replication by copying critical data to a "secondary" storage location.

1. def replicate\_data(data, primary\_file="primary\_data.pkl", secondary\_file="secondary\_data.pkl"):

2. """Replicate data to a secondary storage."""

3. with open(primary\_file, "wb") as f:

4. pickle.dump(data, f)

5. with open(secondary\_file, "wb") as f:

6. pickle.dump(data, f)

7. print("Data replicated to secondary storage.")

8.

9. def recover\_data(primary\_file="primary\_data.pkl", secondary\_file="secondary\_data.pkl"):

10. """Recover data from secondary storage if primary is unavailable."""

11. try:

12. with open(primary\_file, "rb") as f:

13. return pickle.load(f)

14. except FileNotFoundError:

15. with open(secondary\_file, "rb") as f:

16. return pickle.load(f)

17.

18. *# Simulate data replication and recovery*

19. sample\_data = {"important": "This is very important data"}

20. replicate\_data(sample\_data)

21.

22. *# Simulate primary data loss*

23. os.remove("primary\_data.pkl")

24. recovered\_data = recover\_data()

25. print(f"Recovered data: {recovered\_data}")

26.

Python Code for Simulated Fault Tolerant Component Tree Computation

1. class TreeNode:

2. def \_\_init\_\_(self, value=0, children=None):

3. self.value = value

4. if children is None:

5. self.children = []

6. else:

7. self.children = children

8.

9. class GeneralTree:

10. def \_\_init\_\_(self):

11. self.root = None

12. self.checkpoint = None

13.

14. def insert(self, value, parent=None):

15. new\_node = TreeNode(value)

16. if self.root is None:

17. self.root = new\_node

18. elif parent:

19. parent.children.append(new\_node)

20. else:

21. print("Parent not specified for non-root insertion.")

22.

23. def create\_checkpoint(self):

24. import copy

25. self.checkpoint = copy.deepcopy(self.root)

26. print("Checkpoint created.")

27.

28. def restart\_from\_checkpoint(self):

29. if self.checkpoint:

30. self.root = self.checkpoint

31. print("Restarted from checkpoint.")

32. else:

33. print("No checkpoint found.")

34.

35. def simulate\_failure(self):

36. self.root = None

37. print("Simulated failure: Tree data lost.")

38.

39. def print\_tree(self, node, level=0):

40. if node is not None:

41. print(' ' \* 4 \* level + '->', node.value)

42. for child in node.children:

43. self.print\_tree(child, level + 1)

44.

input

1. *complex tree structure*

2. tree = GeneralTree()

3. tree.insert(1) *# Root node*

4.

5. *# Adding children to the root node*

6. tree.insert(2, tree.root)

7. tree.insert(3, tree.root)

8.

9. *# Adding children to the first child of the root*

10. tree.insert(4, tree.root.children[0])

11. tree.insert(5, tree.root.children[0])

12.

13. *# Adding a child to the second child of the root*

14. tree.insert(6, tree.root.children[1])

15.

16. *# Adding more levels*

17. tree.insert(7, tree.root.children[0].children[1]) *# Child of node 5*

output

1. Original tree:

2. -> 1

3. -> 3

4. -> 6

5. -> 2

6. -> 5

7. -> 7

8. -> 4

9. Checkpoint created.

**Modified Tree Before Failure**

**If you were to add more nodes after creating the checkpoint and before simulating a failure:**

1. tree.insert(8, tree.root.children[1]) *# Adding a child to node 3*

**Output showing the tree just before simulating the failure:**

1. Modified tree before failure:

2. -> 1

3. -> 3

4. -> 8

5. -> 6

6. -> 2

7. -> 5

8. -> 7

9. -> 4

**After Simulating Failure and Restarting from Checkpoint**

**Simulating a failure wipes out the current tree data, and restarting from the checkpoint restores the tree to its state at the time of the checkpoint:**

1. tree.simulate\_failure()

2. tree.restart\_from\_checkpoint()

**output showing the tree restored to its checkpoint state, which does not include node 8 added after the checkpoint:**

1. Simulated failure: Tree data lost.

2. Restarted from checkpoint.

3. Tree after restart from checkpoint:

4. -> 1

5. -> 3

6. -> 6

7. -> 2

8. -> 5

9. -> 7

10. -> 4