Lab Manual: Performance Testing of a Local Web CRUD Application Using Locust

Background Theory

What is Performance Testing?

Performance testing is a type of non-functional testing conducted to evaluate the speed, scalability, and stability of a software application under a given workload. This testing is crucial for ensuring that applications can handle expected and unexpected user loads without degradation in performance.

Types of Performance Testing

1. **Load Testing**: Evaluates how the application behaves under expected load conditions. It simulates multiple users accessing the application simultaneously to measure response times and throughput.
2. **Stress Testing**: Determines the application's breaking point by applying extreme loads. The goal is to identify how the system behaves under stress and to find the maximum capacity it can handle before failing.
3. **Spike Testing**: Tests the application's response to sudden increases in load. This helps ensure that the application can handle unexpected traffic spikes without crashing.
4. **Endurance Testing**: Assesses the application's stability and performance over an extended period. This test helps identify memory leaks and performance degradation over time.

Introduction to Locust

Locust is an open-source load testing tool that allows users to define user behavior in Python code. It is particularly useful for testing web applications and APIs. Locust's features include:

- **Easy to Use**: Define user behavior with simple Python code.
- **Scalable**: Can simulate thousands of users with minimal overhead.
- **Web-Based UI**: Provides a web interface for monitoring test execution in real time.

Structured Approach to Performance Testing

1. Load Testing: `load_test.py`

User Story

- **As a** system administrator, **I want to** assess how the system performs under normal load conditions, **so that** I can ensure it meets performance expectations during peak usage.

Test Cases

1. **Create User**: Simulate multiple users creating new users concurrently.
2. **Retrieve Users**: Simulate multiple users retrieving the list of users.
3. **Update User**: Simulate concurrent updates to existing user records.
4. **Delete User**: Simulate concurrent deletions of user records.

Implementation

```
from locust import HttpUser, task, between, events

class UserApiUser(HttpUser):
    wait_time = between(1, 5)

    @task(1)
    def create_user(self):
        self.client.post("/users", json={"name": "User Load Test"})

    @task(2)
    def retrieve_users(self):
        self.client.get("/users")

    @task(3)
    def update_user(self):
        self.client.put("/users/1", json={"name": "Updated User Load Test"})

    @task(4)
    def delete_user(self):
        self.client.delete("/users/1")

@events.test_start.add_listener
def on_test_start(environment, **kwargs):
    print("Starting Load Test")
```

2. Stress Testing: `stress_test.py`

User Story

- **As a** developer, **I want to** identify the system's breaking point under extreme load, **so that** I can ensure the system can handle unexpected surges in traffic.

Test Cases

1. **High Concurrent Requests**: Simulate a high number of users requesting user data.
2. **Rapid User Creation**: Simulate rapid creation of multiple users.
3. **Frequent Updates**: Simulate frequent updates to user records.

4. **Frequent Deletions**: Simulate frequent deletions of user accounts.

Implementation

```
from locust import HttpUser, task, between

class StressTest(HttpUser):
    wait_time = between(0, 1)  # No wait time for stress testing

    @task
    def stress_test(self):
        self.client.get("/users")

    @task
    def create_user(self):
        self.client.post("/users", json={"name": "Stress Test User"})

    @task
    def update_user(self):
        self.client.put("/users/1", json={"name": "Updated Stress User"})

    @task
    def delete_user(self):
        self.client.delete("/users/1")
```

3. Spike Testing: `spike_test.py`

User Story

- **As a** product manager, **I want to** test the system's response to sudden traffic spikes, **so that** I can ensure it remains stable and responsive under unexpected high load.

Test Cases

1. **Create Users During Spike**: Simulate a sudden increase in user creation requests.
2. **Retrieve Users During Spike**: Simulate a sudden increase in retrieval requests.

Implementation

```
from locust import HttpUser, task, between

class SpikeTest(HttpUser):
    wait_time = between(1, 5)

    @task
    def spike_test_create(self):
        self.client.post("/users", json={"name": "Spike Test User"})

    @task
    def spike_test_retrieve(self):
        self.client.get("/users")
```

4. Endurance Testing: `endurance_test.py`

User Story

- **As a** system architect, **I want to** evaluate the system's performance over an extended period, **so that** I can identify potential memory leaks and ensure stability.

Test Cases

1. **Continuous Retrieval**: Continuously retrieve user data over an extended period.
2. **Continuous User Creation**: Continuously create new users under a steady load.

Implementation

```
from locust import HttpUser, task, between

class EnduranceTest(HttpUser):
    wait_time = between(1, 2)

    @task
    def endurance_test(self):
        self.client.get("/users")

    @task
    def create_user(self):
        self.client.post("/users", json={"name": "Endurance Test User"})
```

Running Each Test

Precondition: Using the terminal, run:

1. python app_using_json_server.py
   or
2. python app_using_sqlite_server.py

To execute each type of performance test, use the following commands in your terminal:

1. **Load Test**:

   ```
   locust -f load_test.py --host=http://127.0.0.1:5000
   ```

2. **Stress Test**:

   ```
   locust -f stress_test.py --host=http://127.0.0.1:5000
   ```

3. **Spike Test**:

   ```
   locust -f spike_test.py --host=http://127.0.0.1:5000
   ```

4. **Endurance Test**:

   ```
   locust -f endurance_test.py --host=http://127.0.0.1:5000
   ```

Ensure Locust is Installed

Make sure Locust is installed in your Python environment with the following command:

```
pip install locust
```

Conclusion

This lab manual provides a comprehensive guide to conducting performance testing on a local web CRUD application using Locust. By following the structured approach, you can effectively assess the application's performance under various conditions, ensuring it meets the necessary performance criteria before deployment

Performance Test Report

The expected reports from performance testing types (Load Testing, Stress Testing, Spike Testing, and Endurance Testing) will provide insights into how the system behaves under different conditions. Below are the key metrics and elements that should be included in each report.

1. Load Testing Report

Objective: Assess system performance under expected load.

Expected Metrics:

- **Response Time**: Average, median, and 95th percentile response times for requests.
- **Throughput**: Number of requests per second (RPS).
- **Error Rate**: Percentage of failed requests.
- **CPU and Memory Usage**: Resource utilization on the server during the test.
- **Successful Requests**: Count of successful responses versus total requests.

Sample Format:

asciidoc
Copy

```
Load Testing Report
--------------------
- Total Requests: 5000
- Total Successful Requests: 4900
- Total Failed Requests: 100
- Average Response Time: 200 ms
- 95th Percentile Response Time: 350 ms
- Requests per Second (RPS): 100
- CPU Usage: 70%
- Memory Usage: 65%
```

2. Stress Testing Report

Objective: Determine the system's breaking point.

Expected Metrics:

- **Response Time**: How response time changes as load increases.
- **Maximum Load Before Failure**: The number of concurrent users or requests at which the system fails.
- **Error Rate**: Increase in errors as load approaches maximum capacity.
- **Resource Utilization**: CPU and memory usage trends as load increases.
- **Performance Degradation**: Observations on how performance decreases under stress.

Sample Format:

sql_more
Copy

```
Stress Testing Report
---------------------
- Maximum Concurrent Users: 500
- Load at Which System Failed: 600 Users
- Average Response Time at Max Load: 800 ms
- Error Rate at Max Load: 20%
- CPU Usage at Max Load: 90%
- Memory Usage at Max Load: 85%
```

3. Spike Testing Report

Objective: Observe system behavior under sudden traffic spikes.

Expected Metrics:

- **Response Time**: How quickly the system responds after a spike.
- **Recovery Time**: Time taken for the system to return to normal performance after a spike.
- **Error Rate**: Number of errors during the spike period.
- **Throughput**: Requests per second during the spike.
- **Resource Utilization**: CPU and memory usage during and after the spike.

Sample Format:

asciidoc
Copy

```
Spike Testing Report
--------------------
- Peak Load: 1000 Users
- Response Time During Spike: 1200 ms
- Recovery Time: 5 minutes
- Error Rate During Spike: 30%
- Requests per Second During Spike: 150
```

```
- CPU Usage During Spike: 95%
```

4. Endurance Testing Report

Objective: Evaluate system performance over an extended period.

Expected Metrics:

- **Response Time**: Average response times over the duration of the test.
- **Memory Leaks**: Increase in memory usage over time.
- **CPU Usage**: Average CPU usage throughout the test.
- **Error Rate**: Any errors that occur during the test period.
- **Stability**: Consistency of performance metrics over time.

Sample Format:

asciidoc
Copy

```
Endurance Testing Report
-------------------------
- Test Duration: 6 hours
- Average Response Time: 300 ms
- Peak Response Time: 500 ms
- Average Memory Usage: 70%
- Maximum Memory Usage: 85%
- Total Errors: 50
- Error Rate: 1%
```

Summary

Each performance testing type provides unique insights into how the system behaves under different conditions. By compiling the metrics mentioned above into structured reports, you can assess system performance, identify bottlenecks, and ensure that the application can handle expected user loads. These reports can then be used for further optimization and planning for future capacity needs.