Load Testing

Arnav Gupta

April 22, 2024

Contents

1 Motivation

Large-scale software systems have rapid growth and varying usage patterns.

Load testing mimics multiple users repeatedly performing the same tasks, taking hours or days.

2 Experimental Design

The goal of a proper **experimental design** is to obtain the maximum info with the min number of experiments.

Response variable: outcome of an experiment

Factor: each variable that affects the response variable and has several alternatives

Levels: the values that a factor can have

Replication: repetition of all or some experiments

Interaction effects: 2 factors A and B interact if the effect of 1 depends on the other

Ad-hoc approach is to iteratively go through each (discrete and continuous) factors and identify factors which impact performance.

A t way covering array for a given input space model is a set of configurations in which each valid combination of factor values for every combination of t factors appears at least once.

Combinatorial Interaction Testing (CIT) models a system under test as a set of factors, each of which takes its values from a particular domain. Generates a sample that meets the specific coverage criteria.

3 Load Test Design

3.1 Designing Realistic Loads

Loads should be realistic, characterized by workload mix and workload intensity.

Steady load is easier to measure, but could have memory leaks.

Step-wise load has same workload mix, but different workload intensity.

Derive testing loads from historical data.

In the case of missing past usage data, testing loads can be extrapolated from beta-usage data, interviews with domain experts, and competitors' data.

Can derive realistic usage model from UML use case diagrams, Markov chains, or stochastic form-oriented models (pages are ovals, actions are boxes).

3.2 Designing Fault-Inducing Loads

Analyze code or formulate model from system.

Data flow analysis identifies potential load sensitive modules and regions for load sensitive faults for artifacts like memory leaks or incorrect dynamic

memory allocation.

Load sensitivity index (LSI) indicates the net increase/decrease of heap space used for each iteration: the difference of heap size before and after each iteration. Write test cases that exercise code regions with high LSI values.

Symbolic executions can be done by path performance estimation (response time) or memory analysis.

System models can be made using genetic algorithms using mutations and crossover:

- genes are a particular type of web service
- chromosomes are resulting workflow
- fitness function is the risky workflow with high response time

3.3 Reductions

To reduce load test effort and cost, use extrapolation for step-wise load testing:

- only examine a few load levels and extrapolate system performance for other load levels

Can be done with probability mass coverage, associating probability with number of states associated.

Realistic load tests are based on historical field workloads, but these change over time. Field workload can evolve over time, so must periodically update load profiles.

4 Load Test Execution

Can be live-user based or driver based.

Live-user based test execution reflects realistic user behaviour through real user feedback on performance and correctness:

- hard to scale
- limited test complexity due to manual coordination
- users selected based on different testing criteria

Driver-based test execution loads driver configurations:

- easy to automate
- scale to large number of requests
- hard to track some system behaviour
- uses specialized benchmarking tools, centralized load drivers, and peer-to-peer load drivers

General aspects when executing a load test are setup, load generation/termination, and test monitoring and data collection.

Field load testing is costly but realistic. Hardware is either dedicated or cloud-based. Create realistic databases by important data or sanitizing the field database. Mimick network traffic by keeping track of network latency and network spoofing. Do not deploy drivers on the same machines with the system under test.

Setup for live-user is tester recruitment, setup, and training.

Setup for driver-based is programming, store-and-replay configuration, and model configurations.

Load germination and termination can be static configuration, which is timer-driven, counter-driven, or statistic-driven, or dynamic feedback, which dynamically steers testing loads based on system feedback.

For dynamic feedback with system identification techniques, start with random testing to identify performance sensitive input.

For dynamic feedback with two-layered queueing models, use a stress goal (target performance threshold) to create tests.

Live-user based is static, driver-based is static and dynamic.

Source code level instrumentation includes ad-hoc manual, automated, and performance frameworks.

Measurement bias is hard to avoid and unpredictable. To help, use repeated measurement or randomize experiment setup.

5 Load Test Analysis

Verify against threshold values:

- straightforward comparison

- comparison against processed data: maximum, medium/average, or 90-percentile value
- comparison against derived data: deriving thresholds and deriving target data

Detecting known problems using patterns:

- patterns in the memory utilizations: memory leak detection
- patterns in the logs: error keywords

Check that performance data is under steady load, unstable indicates deadlocks or memory leak.

Automated detection of anomalous behaviour by automatically deriving expected/normal behaviour and flaging anomalous behaviour.

- done with data mining and queueing theory
- check using control charts for past good tests and flag new test as anomalous if there are many violations in the control charts
- use performance rules: extract rules from past tests and flag tests where the rule does not hold (ex. request queue size high)
- automated functional analysis if methods are together that shouldn't be