Model-based Adaptive Performance Load Testing

Cornel Barna, Marin Litoiu

cornel@cse.yorku.ca, mlitoiu@yorku.ca

YORK UNIVERSITÉ UNIVERSITY

Performance Load Testing

Performance testing, including load stressing, is fundamental in assessing the performance of software components as well as of an entire software system. In general, the testing is done under operational conditions, that is, the testing is typically based on the expected usage of the system once is deployed and on the expected workload. It consists of the types of executed scenarios and the rate of these scenarios. A performance test usually lasts for several hours or even a few days.

A major goal of performance testing is to uncover functional and performance problems under load and the root cause of those problems. Functional problems are often bugs, deadlocks and memory management bugs. Performance problems often refer to high response time or low throughput under load.

Performance testing, for most part, is done manually.

The Framework

We propose a framework that

- models the application, the corresponding workloads and the environment performance;
- correlates or establishes quantitative dependencies between different metrics of the system and the workloads;
- computes and runs the worst case workloads.

The framework is adaptive, based on an autonomic computing loop in which we monitor the execution of each workload, analyze the current performance, plan a new workload based on the analysis results and then execute the new workload.

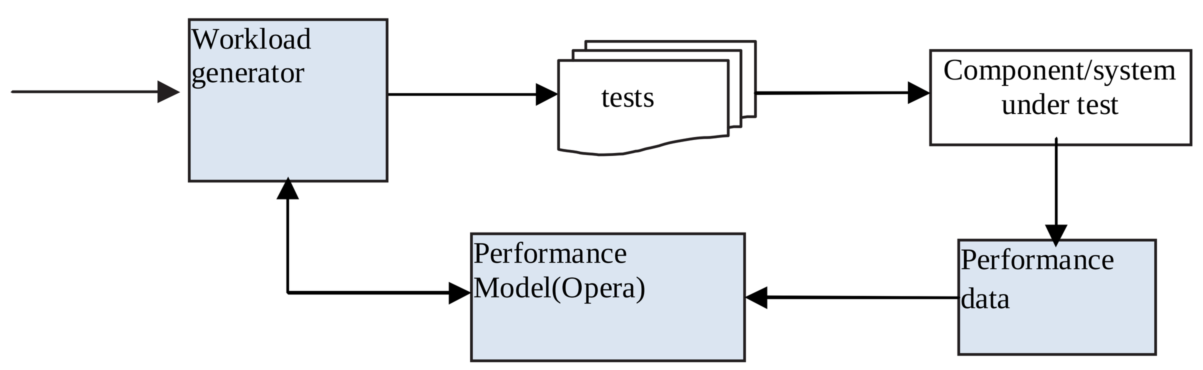


Figure 1: Adaptive performance load testing.

A *performance model* captures quantitative relationships between inputs, states, environment and output metrics of the software system.

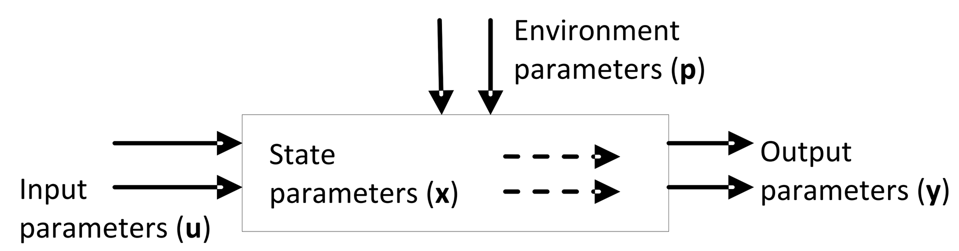


Figure 2: A software subsystem with its performance parameters.

Worst Workload Mixes

A common way to model the user interaction with the transactional systems is to define *classes of services* or *classes* in short. A class is a service or a group of services with a similar statistical behaviour and specific performance requirements.

- $\mathcal{C}$ – the set of classes;
- $\mathcal{K}$ – the set of resources;
- U_K – the total utilization for $K \in \mathcal{K}$;
- $U_{K,C}$ – the utilization of $K \in \mathcal{K}$ by requests of $C \in \mathcal{C}$;
- $D_{K,C}$ – the demand of $K \in \mathcal{K}$ in $C \in \mathcal{C}$;
- $\mathcal{N} = \langle N_1, N_2, \ldots, N_{|\mathcal{C}|} \rangle$ – the *workload vector*, where N_i is the number of users in $C_i \in \mathcal{C}$;
- $N = N_1 + N_2 + \cdots + N_{|\mathcal{C}|}$ – *workload intensity*;

When the workload mix changes, the bottleneck in the system can change as well. Workload mixes yield per class utilization at each resource; the sum of per class utilizations equals the total utilization of that resource:

$$U_K = \sum_{C \in \mathcal{C}} \frac{D_{K,C}}{D_{R,C}} U_{R,C} = 1, \quad \text{for all} \quad K \in \mathcal{K} \qquad (1)$$

where $R \in \mathcal{K}$ is a reference resource shared by all classes of request.

If U^*_{R,C_i} is a solution for equations (1) and we know N then we can compute the workload vector $\mathcal{N}$ by solving the equations:

$$\beta^*_i = \frac{N_i}{N} = U^*_{R,C_i} \qquad (2)$$

The Algorithm

input: N – the initial number of users
input: U_t – the target utilization for device t

Tune the LQM by measuring and adjusting the service demands for each scenario;
Find all extreme points by solving the equations (1);
Compute the switching points, $\mathcal{P}$, by using (2);
foreach *switching point* $p \in \mathcal{P}$ **do**
  $u_{e,t} \leftarrow -1$; // *estimated utilization*
  $u_{m,t} \leftarrow -1$; // *measured utilization*
  // *Stop when the estimated utilization is within 5% from the target utilization*
  while $\left|1 - \frac{u_{e,t}}{U_t}\right| > 0.05$ **do**
    Compute $\langle N_1, N_2, \ldots, N_{|\mathcal{C}|} \rangle$ for N and p;
    Solve model for $\langle N_1, N_2, \ldots, N_{|\mathcal{C}|} \rangle$;
    Update $u_{e,t}$ with the estimated value;
    if $\left|1 - \frac{u_{e,t}}{U_t}\right| > 0.05$ **then**
      Update N using a hill climbing strategy;
  while $\left|1 - \frac{u_{m,t}}{U_t}\right| > 0.05$ **do**
    Compute $\langle N_1, N_2, \ldots, N_{|\mathcal{C}|} \rangle$ for N and p;
    Generate workload and measure the metrics;
    Update $u_{m,t}$ with the value measured;
    if $\left|1 - \frac{u_{m,t}}{U_t}\right| > 0.05$ **then**
      Update N using a hill climbing strategy;

Experiments

We created a cluster with two Web Servers (Tomcat), one Database Server (MySQL) and one Workload Balancer (Apache) to distribute the incoming web requests to the two web servers.

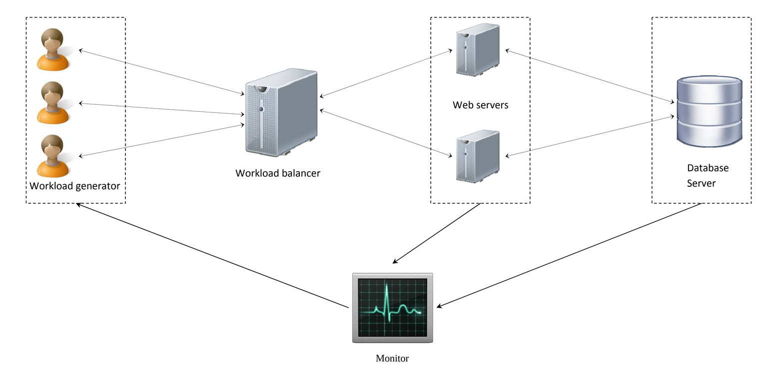


Figure 3: The cluster used for experiments.

We implemented three simple scenarios (classes of service) as servlets:

insert	insert a record in database
update	update a record from database
select	select 1000 records from database

The goal is to find the number of users, N, and the workload mixes that will generate a CPU utilization above a certain threshold.

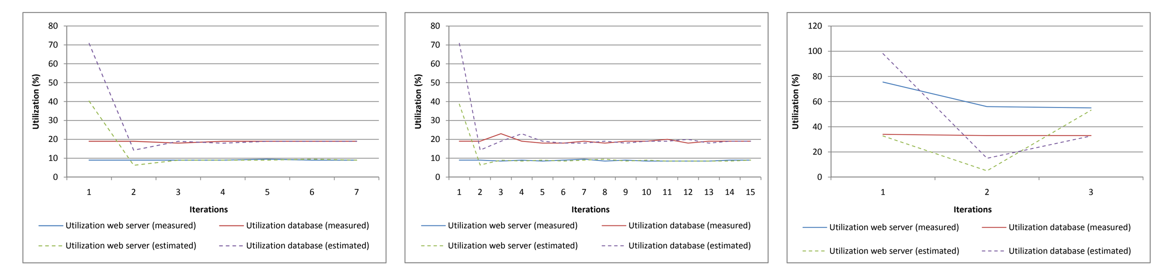
Kalman filters

Tunning the LQM is not always easy because some parameters cannot be directly measured. This is the case for service demand. To estimate the correct values we used Kalman filters and the following algorithm:

while $\left|1 - \frac{u_{e,t}}{u_{m,t}}\right| > 0.05$ **do**
  Solve model;
  Update $u_{e,t}$ with the model estimated value;
  Update $u_{m,t}$ with the measured value;
  if $\left|1 - \frac{u_{e,t}}{u_{m,t}}\right| > 0.05$ **then**
    Estimate service demands using Kalman filters;
    Update model with the estimated service demands;

Results

Using Kalman filters we were able to find the service demands fast:



(a) *Insert* scenario. (b) *Update* scenario. (c) *Select* scenario.

Figure 4: Finding demands for LQM using Kalman filters.

We updated the model with the demands found by Kalman filters and tried to predict the number of users and the workload mixes that will generate an utilization above 50% on web servers or database server. The results are summarized in the next table:

Switching points	Predicted users #	Measured users #
$\langle 1.00, 0.00, 0.00 \rangle$	284	284
$\langle 0.00, 1.00, 0.00 \rangle$	284	284
$\langle 0.00, 0.00, 1.00 \rangle$	94	94
$\langle 0.95, 0.00, 0.05 \rangle$	259	259
$\langle 0.98, 0.00, 0.02 \rangle$	284	284
$\langle 0.00, 0.90, 0.10 \rangle$	259	259
$\langle 0.00, 0.97, 0.03 \rangle$	284	284

We showed that our algorithm is capable to correctly predict the workload required to have an utilization of CPU above a specified threshold.