The $M^3$ (Measure-Measure-Model) Tool-Chain for Performance Prediction of Multi-tier Applications

Devidas Gawali  Varsha Apte

Department of Computer Science and Engineering
Indian Institute of Technology, Bombay, India.

{$devidas,varsha}@cse.iitb.ac.in

QUDOS 2016
Saarland University, Saarbrucken, Germany.

July 21, 2016
Overview

1. Problem Statement
2. Approach
3. The $M^3$ Toolchain
4. Validation
5. Experiments
6. Conclusion and Future Work
Production hardware most often \textit{different} from testbed hardware
Problem Statement

Given the results of performance measurement of an application A on a testbed platform X, Predict the resource utilization, response time and throughput of application A on a target platform Y.

Challenges:

- May be difficult/impossible to deploy entire original application on target and load test it
- Modeling also requires resource service demands of application on target
Key idea: Performance “Clone”

- Generate a simple “clone” (A program that mimics the performance of the application on the testbed.)
- Deploy and measure the clone (instead of the original application) on the target
- Measure resource demand of clone on the target
- Use this as an estimate of resource demand in a queueing system model of the application on the target

In this work focus is on CPU-intensive Web tier
The Clone - Architecture

- Mimics the application, matches CPU service demand
- Works with only a “dummy” back-end tier
The Clone Code

Figure: Clone Template

```c
/*
Server Address
*/

/*
socket calls writes and receives the same amount of bytes to the network as the original call.
*/

while(loopvalue){
   //Dummy code instructions
}
```

Code snippet:

- /* Integer Type Instruction */
- /* String Type Instruction */
- /* Float Type Instruction */
The $M^3$ Toolchain

We use a “chain” of measurement and modeling tools developed in-house to implement the clone-based performance prediction approach.

**Figure:** Three-step hybrid measurement and modeling approach

- **Measure**
  - Measure CPU Service Demand of the Application on Testbed machine
  - Generates Clone.

- **Measure**
  - Measure CPU Service Demand of a Clone on Target machine.

- **Model**

AutoPerf CloneGen AutoPerf Model
The $M^3$ Toolchain

**Tool pipeline**

1. **AutoPerf:**
   Profile the application performance on the testbed platform.

2. **CloneGen:**
   Generates clones of the application server-side request codes that are easier to run on the target platform.

3. **AutoPerf:**
   Measure the clones performance on the target platform.

4. **PerfCenter:**
   Produce application performance metrics on the target.
Figure: Profile the application
Generating the Clone

**Figure:** Desired Service demand achieved by tuning loopvalue

- **App**
  - CPU Service Demand & Other Profiling Data
  - App

- **Clone Generator**
  - Service Time Matched
  - Yes
  - No
  - Clone Generator

- **Profiler**
  - Autopercf Load Generator
  - http request
  - http response

- **Testbed Server**
  - Web
  - DB

- **5. Final Clone**
- **1. Deploy Clone**
Input

1. Number of back end (DB) calls.
2. Sent and received bytes exchanged between web and DB server.
3. Service time of the Application on web and DB server.
4. Type of instructions of code snippet used in the Clone benchmark.
sample input file

number_of_BackEnd_Calls = 4
serviceDemand_of_App_on_Web = 0.0016
serviceDemand_of_App_on_db = 0.0019
bytes_from_web_to_db = 120
bytes_from_db_to_web = 400
bytes_from_web_to_client = 1100
type_of_instructions = string
client_machine_server_address = 10.129.X.X
testbed_machine_server_address = 10.129.X.X
target_machine_server_address = 10.129.X.X
Toolchain: Measuring Clone Service Demand on Target, using it in a model

Figure: Measure Clones Service demand on Target

Figure: Predict App Performance
Input

1. Measured Service demand: **Clone’s CPU Service Demand - from Toolchain**
2. Hardware details of Target: **Number of devices, Number of CPU’s - manually specified**
3. Message flow details: **bytes exchange between servers - measured separately and specified**

Output

1. Modeled Application Throughput
2. Modeled Utilization of host CPU
3. Modeled Application Response Time
The $M^3$ Toolchain: PerfCenter: input file

variable
  nusr 1
end
device
  markov_cpu
  raid_disk
disk_array
end
host server_host[2]
  markov_cpu count 4
  markov_cpu buffer 99999
  markov_cpu schedP fcfs
  raid_disk count 16
  raid_disk buffer 99999
  raid_disk schedP fcfs
end
task search_item
  markov_cpu servt 0.00116
end
task view_item
  markov_cpu servt 0.00132
end
task viewuserinfo
  markov_cpu servt 0.00105
end
task execute_query
  markov_cpu servt 0.00075
end
task get_response
  markov_cpu servt 0.000001
end
server web
thread count 2600
thread buffer 0
thread schedP fcfs
task search_item
task view_item
task viewuserinfo
task get_response
end
server db
thread count 2600
thread buffer 0
thread schedP fcfs
task execute_query
end
deploy web server_host[1]
deploy db server_host[2]
lan
  lan1
end
deploy server_host[1] lan1
deploy server_host[2] lan1
scenario SearchItem prob 0.3334
  search_item execute_query 222
SYNC
  execute_query get_response
836
end
scenario ViewUserInfo prob 0.3333
  view_userinfo execute_query
260
SYNC
  execute_query get_response
700
end
loadparams
  noofusers nusr
  thinktime 0.3
end
modelparams
  method simulation
  type closed
  noofrequests 50000
end
print
"Users,resp,tput,util(server_host[1]:markov_cpu),
  (server_host[2]:markov_cpu)"
for nusr = 100 to 100 incr 100
print
  nusr+","+resp()+","+tput()+","+util
Clone's CPU Service Demand
Host Specification
Server Specification
Message Flow

Validation

- We used two standard Web benchmarks (DellDVD, RUBiS) with various combinations of testbed and target platforms.
- We compared measured vs modeled metrics by using our measure-measure-model approach.
## Experiment Combinations

<table>
<thead>
<tr>
<th>Server Type</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD1</td>
<td>16 core Opteron(tm) Processor 6278 @1.4 GHz - 2.4GHz, 16GB RAM</td>
</tr>
<tr>
<td>AMD2</td>
<td>16 core Opteron(tm) Processor 6212 @1.4 GHz - 2.6GHz, 16GB RAM</td>
</tr>
<tr>
<td>Intel</td>
<td>24 core Xeon(R) CPU E5-26200 @1.2GHz - 2.60GHz, 16 GB RAM</td>
</tr>
</tbody>
</table>

*Table: Server specifications*

<table>
<thead>
<tr>
<th>Type</th>
<th>Application</th>
<th>Server (Testbed-Target)</th>
<th>Cores,Frequency (Testbed-Target)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>DellDVD</td>
<td>Intel-AMD1</td>
<td>2,2.4 - 2,1.6</td>
</tr>
<tr>
<td>C2</td>
<td>DellDVD</td>
<td>Intel-Intel</td>
<td>2,2.4 - 2,1.6</td>
</tr>
<tr>
<td>C3</td>
<td>DellDVD</td>
<td>AMD2-AMD1</td>
<td>2,2.4 - 2,2.4</td>
</tr>
<tr>
<td>C4</td>
<td>DellDVD</td>
<td>AMD1-Intel</td>
<td>2,1.6 - 2,2.4</td>
</tr>
<tr>
<td>C5</td>
<td>RUBiS</td>
<td>Intel-AMD1</td>
<td>2,1.4 - 4,1.6</td>
</tr>
<tr>
<td>C6</td>
<td>RUBiS</td>
<td>Intel-AMD1</td>
<td>2,1.4 - 4,2.4</td>
</tr>
</tbody>
</table>

*Table: Testbed and Target Combination Specification*
Measured vs Modelled: Throughput, Utilization

**Figure: Combination 1**

**Figure: Combination 2**

**Figure: Combination 3**

**Figure: Combination 4**
Measured vs Modeled: Error frequency distribution

Figure: Histogram of Measured vs Modeled Error %
Conclusion and Future work

Conclusion

- Proposed the Measure-Measure-Model ($M^3$) methodology for application performance prediction.
- Demonstrated how a tool-chain of measurement, clone generation and modeling tools can be built for the purpose of automating this methodology (partially).
- Validated our approach on two standard Web benchmarks with various combinations of testbed and target platforms.

Future work

- Support a range of resource demands over a certain frequency distribution.
- Validate and if required extend our approach to Web applications written in Java.
- Predict application performance in a virtualized environment.
Thank you.