DESiDE:Discrete Event Simulation Developers Environment

Benny Mathew Dheeraj Chahal

{benny1.m|d.chahal}@tcs.com
TCS Innovation Labs
Mumbai, India

ABSTRACT

Performance prediction of an application early in its Systems Development Life Cycle (SDLC) is essential due to stringent application performance Service Level Agreements (SLAs). Accurate sizing during requirement analysis phase helps in meeting application SLAs and also in reducing expensive performance tuning efforts, hardware upgrades and redesign after the application has been developed and migrated to production environment. There is growing need to use scientific and accurate application performance prediction techniques to reduce cost and increase profitability. We present our tool called DESiDE that uses systematic profiling of components of previously developed applications to accurately size infrastructure for new application during requirement or design phase using discrete event simulation.

Keywords

Performance prediction; discrete event simulations

1. INTRODUCTION

Often it is required to estimate the total cost of implementing an IT solution at the time of requirement analysis. Cost of the required infrastructure is one of the factors that contribute to the overall cost of the solution. Estimating infrastructure cost accurately when the solution artifacts are not ready is a challenging task. Infrastructure requirements are bound by application performance SLAs. Hence application performance modeling is essential and inevitable during the requirement gathering phase.

There exists performance prediction tools like TeamQuest's Predictor [1] and Netuitive [2]. These tools can be used only after the system goes live. Some other products like OPNET [3] have built in models of web, application and database servers. However technical architects find it difficult to relate their application requirements to default options provided in these products.

Roy et al. [3] proposed a technique to break down existing applications into components and carry out performance testing of each of these components followed by analytical approach to predict performance of new applications using the data available from these components.

Another popular approach is use of discrete event simulation (DES) for prediction instead of analytical approach so that systems can be modeled with higher complexity. There are many freeware and open source tools for DES like JMT [2] and OMNET [4]. These tools have enough APIs required for DES but sometimes there is a need to manipulate the dependent libraries to perform complex modeling e.g. dynamic addition and removal of a resource and all its scheduled events, time of day and day of week aware simulations, metrics like 90 percentile wait times, application code implementation in the simulator etc.

We address these limitations and enhances the capabilities of the existing DES tools in our tool called DESiDE. DESiDE also has capabilities to carry out several *what-if* studies such as how much benefit can be derived by improving infrastructure specifications or if the workload is expected to change in future, how that is going to affect the application performance.

2. DESIDE ARCHITECTURE AND FEATURES

DESiDE Architecture is as shown in Figure 1. It comprises of 4 layers. The interface layer helps connect to different input/output modes like XML, Excel (through Apache-POI library) and databases. The Simulation Model layer comprising of built-in as well as user created resource and entity models. DESiDE has a few built-in models that can be reused or extended. The built-in server model has attributes like number of CPUs, speed factor and queuing discipline. The supported queuing disciplines are FIFO, LIFO, SIRO (Service in Random Order), priority, pre-emptive priority and processor sharing. The built-in load generator model is used to model workloads in terms of transactions and inter-arrival times.

The Core Simulation Layer provides libraries to manage, run and collect statistics required discrete event simulations. The random variate library supports rich set of continuous

[1] www.teamquest.com
[2] www.netuitive.com
[3] http://www.opnet.com
[4] https://omnetpp.org/

ICPE'17 April 22-26, 2017, L'Aquila, Italy

ACM ISBN 978-1-4503-4404-3/17/04.

DOI: http://dx.doi.org/10.1145/3030207.3053672

and discrete distributions that help to model random behavior of the system. This is particularly useful to model inter-arrival times, service time, packet size and error conditions. The inherent variation in demand can also be defined in terms of one of the available distributions. Load dependent demand is also supported in DESiDE using linear, quadratic, cubic and quartic curve fitting functions defined in random variate library. The flow between servers is specified using entity flow APIs provided by the routing library. If there are more than one destination resource, the routing library provides four options: probabilistic, RR (Round Robin), SNF (Shortest Number), and named resource. In the named resource option, the source server itself specifies the destination server. The future event library contains the list of scheduled events and controls the simulation. Metrics library not only computes mean, variance but also reports the confidence level.
The CERN′s COLT [5] library provides random number generation and various mathematical libraries that can be used by end-user to build resource models.

3. MODELING IT SYSTEMS USING DESIDE

3.1 Modeling IT Servers

We first build resource models representing each tier of an IT system by extending the built-in server models and creating models of web server, application server and database server.

3.2 Component Based Modeling of Workload

We then build a component repository using various reference IT applications. Transactions of each reference application are broken down into smaller components. Service demand of each component is then measured with respect to CPU, memory, disk and network utilization. Modeling transactions at such granular level promotes reusability. This means that new transactions can be modelled just by reusing existing components. These components with their measured demand are stored in a component repository for reuse.

3.3 Simulating IT Systems

Once the server models are ready (section 3.1) and the component repository(section 3.2) is populated with demand data from many representative applications, we can use them to predict performance of an application that is still in design stage. Transactions of the new application are modeled using component repository. The application complexity factor is used to adjust the difference in complexity of the application and the component used from the repository for modeling. Likewise, server speed factor is used to adjust relative speed of the target server as compared to the server on which measurements were taken.

4. EVALUATION

As an example we show the evaluation of JPetStore [1] application. We modeled three JPetStore transactions Tx1, Tx2 and Tx3 in DESiDE using components from component library and simulated the same with the page sequence and

[5]https://dst.lbl.gov/ACSSoftware/colt/

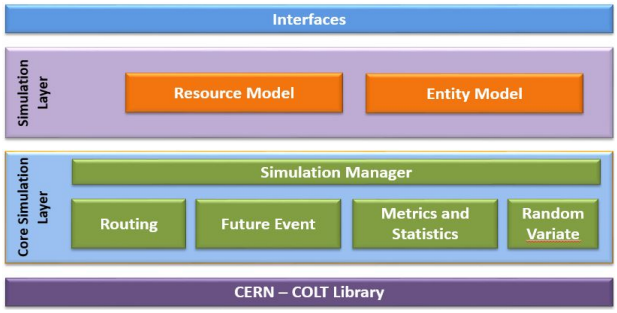

Figure 1: DESiDE Architecture

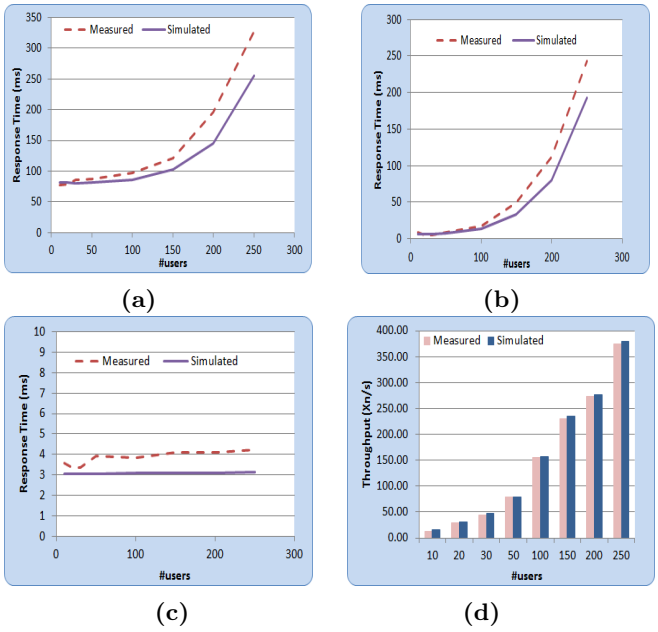

Figure 2: Measured vs simulation (a) Response time of Tx1 (b) Response time of Tx2 (c) Response time of Tx3 (d) Throughput

percentage of transaction mix as described below.
Tx1: HomePage->Searchpage->SearchAction (50%)
Tx2: HomePage->CategoryPage->SelectCategory (30%)
Tx3: HomePage->LoginPage->LoginAction (20%)
The performance comparison of experimental and simulation data using DESiDE is as shown in the Figure 2.

5. REFERENCES

[1] iBATIS JPetStore. http://sourceforge.net/projects/ibatisjpetstore/.

[2] M. Bertoli, G. Casale, and G. Serazzi. JMT: Performance Engineering Tools for System Modeling. *SIGMETRICS Perform. Eval. Rev.*, 36(4):10–15, Mar. 2009.

[3] N. Roy, A. Dubey, A. Gokhale, and L. Dowdy. A capacity planning process for performance assurance of component-based distributed systems. In *Proceedings of the 2Nd ACM/SPEC International Conference on Performance Engineering*, ICPE '11, pages 259–270, New York, NY, USA, 2011. ACM.