CHALLENGES IN PERFORMANCE SIMULATION OF ICT SOLUTION STACKS

Martijn Goorden, Joanna v.d. Mortel-Fronczak and Michel Reniers
Department of Mechanical Engineering
Eindhoven University of Technology
De Groene Loper
Eindhoven
email: m.a.goorden@tue.nl

Matti Kinder and Willem van Veggel
Department of Information Technology
ASML
De Run 6501
5504 DR, Veldhoven

KEYWORDS
Computer systems, Computer networks, Performance analysis, Discrete simulation, Queueing

ABSTRACT

Companies find themselves in a constant balancing act to provide highly available and well performing information technology services, while implementing and endless flow of potentially disrupting changes in business requirements. The impact of changes on existing information and communication technology (ICT) solution stacks is poorly understood upfront. We propose a method to create simulation models to analyze the end-to-end performance of ICT solution stacks. Queueing models of different components are implemented in the simulation language χ and combined to form a network representing the ICT solution stack. Challenges related to the simulation of these models are discussed.

INTRODUCTION

ICT is becoming an integrated part of most business models. ICT capabilities increasingly drive revenue, enable business critical operations and satisfy marketplace initiatives. With this ICT dependence comes the risk of a company being heavily impacted by failures within its ICT infrastructure. Staying competitive requires flexibility to scale, replace, upgrade and integrate new solutions as they are being brought to the market.
In this paper, we define an ICT solution stack as a set of software, middleware and hardware subsystems or components needed to complete a particular task or application. ICT solution stacks are becoming increasingly complex by stacking and virtualizing hardware and software components from various suppliers, using different means of global delivery to different types of local devices.
Expert knowledge turns out to be insufficient to find the right balance due to the complexity of current ICT solution stacks. Questions that ICT architects face are, for example:

1. Given an ICT solution stack design, what is the performance bottleneck? How can the bottleneck be solved?
2. How many web / corporate / database servers are needed to meet user requirements?
3. Which hardware is needed (e.g. number of CPUs, RAM size, clock speed, server settings)?

To answer these questions, ICT architects would benefit from a simulation model. This model should represent a single, but complete ICT solution stack, such that relationships between individual parameter settings and the end-to-end performance can be simulated. Furthermore, the simulation model should be modular, such that changes in the components can be incorporated easily in the model. The simulation model should run offline in the sense that no information is required of the actual system during the simulation.
Several research projects are going on in the field of software engineering with the focus on performance modeling of a complete solution stack or architecture or with the focus of individual components within the ICT solution stack. See (Goorden 2015, Goorden et al. 2016) for a literature review.
None of the found studies describe models or modeling methods which can be directly used for our problem. A rich literature exists with models of individual components, but there is a lack of integration between these components into a single ICT solution stack. Furthermore, currently available architecture-focused models are developed with the perspective of software engineering, while we are interested in the performance of given software together with middleware and hardware components.
The main objective is to come up with repeatable, easy to maintain and cost effective means to accurately simulate end-to-end performance and reliability of ICT solutions, from data center floor to user device, while (part of) the ICT solution has not yet been physically built. The goal of this paper is to explain the simulation implementation of the model developed in (Goorden 2015) used as a proof of concept.
The contribution of this paper is to show how models of ICT solution stacks can be implemented in a simulation model and to point out challenges for further research and tooling development.

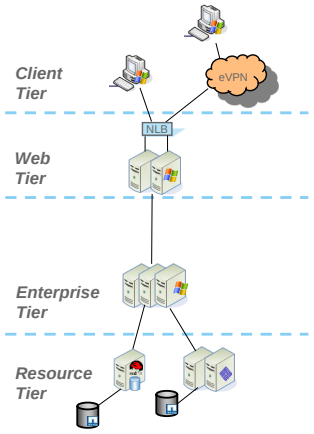

Figure 1: A simplified functional design of an ICT solution stack.

SYSTEM DESCRIPTION

An ICT solution stack is a set of software and hardware subsystems or components needed to complete a particular task or application. Examples of such components are web servers, operating systems, databases, network cables, firewalls and switches. Normally, the different components of a solution stack are developed by different architects independently. The architecture design of an enterprise application can be split into three subdesigns: the functional design, logical design and physical design. In this paper, only the functional design is considered.

An example of a simplified functional design is shown in Figure 1. In a functional design, components are connected to each other by means of their functional need. The functional design is drawn with multiple tiers or layers, resulting in a n-tier design. In a tier, software and hardware are grouped by function. For example, the first tier is the client or user tier where either a thin or rich client is running. The last tier is the resource tier where the actual data is stored. Between those two tiers, zero or more tiers can be added to provide more functionality and/or flexibility.

The ICT solution stack as shown in Figure 1 is used as a working example in this paper. Clients can send requests to the web servers. A network load balancer is used to direct the requests to the web server with the lowest load. The request is then further redirected to one of the corporate servers at the enterprise tier. On these servers, the actual application runs. Finally, (meta)data is stored at one or more database servers.

The 4-tier functional architecture is modeled as a network of queueing models representing the individual components of the functional architecture. These queueing models are taken from the literature when possible. A detailed description of these models can be found in (Goorden 2015).

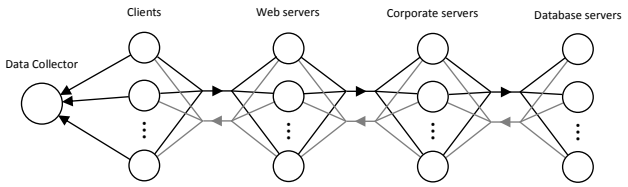

Figure 2: Overview of the used simulation model as implemented in χ.

SIMULATION MODEL

In this section, the theoretical models are implemented in the simulation language χ (van Beek et al. 2006). Below motivation is given for this choice.

The hybrid simulation language χ is developed at Eindhoven University of Technology. This simulation language allows the user to implement high-level behavior of the individual components, while the simulation language itself takes care of the continuous and discrete-event bookkeeping behind the scenes. Furthermore, χ allows the user to program processes from scratch rather than using standardized blocks as in, for example, MathWorks' SimEvents, see (Gray 2007). Therefore, the modeler can simulate more complex flows and processes beyond standard queues and server models.

The simulation language χ allows the modeler to build the simulation model in a modular way. This is a big advantage for our problem. First, we can build the simulation model step by step and debug each module separately before integration. Secondly, in this way, we can replace different models easily. First, process definitions are programmed for each individual process. A process may contain several other processes (encapsulation). In the model definition, the processes are invoked and connected with each other by communication channels. Finally, due to the stochastic nature of queueing models, experiments can be defined which run a model multiple times.

For a more thorough introduction and χ tutorial, one is referred to the website of the language (Eindhoven University of Technology 2015). The software is free to use and licensed under the MIT open source license. χ is implemented as a plug-in to Eclipse.

Implementation of the high-level structure

Figure 2 shows the overview of the implemented model in χ. The four tiers can be clearly seen in this schematic view. Each circle in this picture represents an instantiation of a process definition, which may embed other process definitions. So at the client tier, one or more client processes, which can be different, may be instantiation. The same situation applies to the web tier, corporate tier and database tier. One additional process, the data collector, collects data from the simulation and processes

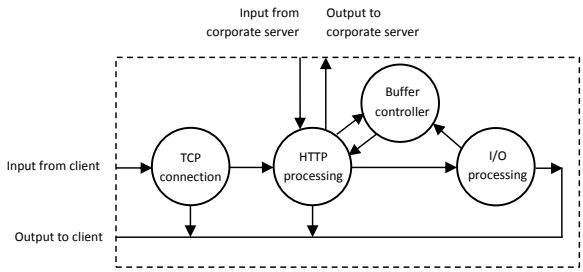

Figure 3: Overview of the simulation model for the detailed web server as implemented in χ.

it to analyze the simulated process. In Figure 2, due to readability, only arrows are drawn from the clients to the data collector. The data collector can also collect data from all other processes.

As can be seen in the schematic view of the simulation model, it is rather easy to remove or add tiers. Furthermore, if one wants to include the communication network between components explicitly, processes describing these communication networks can be inserted into this model in a straightforward manner.

Implementation of the client

The implementation of the client model is based on the standard generator and exit process definition as explained in the tutorial on the website of χ Eindhoven University of Technology (2015). Furthermore, an additional buffer is needed between the generator and exit process. This buffer is represented by a single integer variable since only the number of requests inside the buffer is of interest.

Implementation of web server

Figure 3 shows the overview of the process definition for the web server model. The web server process can be split into TCP connection setup, HTTP processing and I/O processing. These three stages are implemented in separate process definitions to keep the simulation model modular. Four directed communication channels cross the boundary of the web server process definition: one input channel receives requests from the client, one output channel sends requests back to the client (both succeeded and failed requests), one output channel to send requests further down the ICT solution stack and one input channel to receive requests back from the lower part of the ICT solution stack. Since multiple HTTP threads can choose to connect with multiple I/O buffers, a buffer controller is implemented to prevent undesired actions.

Modeling a web server in χ may be tricky due to the possibilities of livelocks and undesired choices. A livelock may arise in the I/O processing. The I/O processing layer consists of multiple I/O buffers and an I/O controller. The I/O controller visits the I/O buffers according to some predefined rule, e.g. a round-robin fashion, and tries to empty that I/O buffer as much as possible at that moment. After sending some data onto the network, according to the TCP protocol the I/O controller needs to wait for an acknowledgement. In the meantime, it can serve other I/O buffers. In the case that all I/O buffers are empty, the I/O controller is checking all I/O buffers. Furthermore, when one assumes that checking for emptiness can be done in neglecting time, the χ simulation is in a livelock with no progressing of time. The current solution requires a list keeping track which I/O buffers need to be served and, if all I/O buffers are empty, forces the I/O controller to stop checking for non-empty I/O buffers.

Another simulation problem arises of choosing an I/O buffer to place data which has to be send over the network in. When HTTP threads have retrieved files from the lower part of the ICT solution stack, the data has to be transferred over the network to the client. An HTTP thread can chose one of the empty I/O buffers. Each file to be send has to be handled by the same I/O buffer and an I/O buffer can start sending another file if it has completely finished processing the previous one. In this situation, each HTTP thread can select from all I/O buffers to connect, and each I/O buffer can select from all HTTP threads to connect. For each new file, a match has to be found between a HTTP thread and an I/O buffer. Without any form of control, this match making process is not working well in χ. Only the match between the first HTTP thread and the first I/O buffer is established by the χ simulator. To solve this problem, a buffer controller process is initiated with keeps track of which HTTP threads and I/O buffers want to connect. On a first-come first-serve basis, matches are made. In this way, all HTTP threads can connect with all I/O buffers.

Implementation of corporate server

The queueing model of the corporate server (the server on the enterprise tier) is an M/G/N queue. On the tutorial page of the website of χ, this system is called a parallel system. Therefore, the implementation of such a queue can be found there. The only modification to the standard server is that it is able to send a request to the database tier and waits for an answer.

Implementation of database server

When a request arrives at the database server, a splitter process determines to which CPU the request is sent, each with equal chance. After service at one of the parallel CPUs, the request moves on to the I/O processing process. Finally, the request is sent back to the

corporate server. Both the CPU process definition and the I/O processing process definition are standard single server processes, which can be found at the tutorial page at the website of χ.

STOCHASTIC SIMULATION DESIGN

In (Kelton and Law 2000) it is observed that most studies using simulation lack a proper simulation methodology, such that the conclusions drawn may be erroneous. First, one has to notice that the output data from simulations is not independent and identically distributed (i.i.d.) (Nakayama 2008). Furthermore, a distinction has to be made between transient (or terminating) performance simulations and steady-state (or long-run) performance simulations. For our problem, steady-state simulations are performed.
Let the output stochastic process of a nonterminating simulation be $Y_1, Y_2, \ldots$ Let $F_i(y|I) = P[Y_i \leq y|I]$ for $i = 1, 2, \ldots$ be the distribution function of Y_i given the initial conditions I. If $F_i(y|I) \to F(y)$ as $i \to \infty$ for all y and I, then F is called the steady-state distribution. In practice, most of the time there is an index k after which the distributions are approximately the same as the steady-state distribution. Two problems arises with steady-state simulations. The distribution functions Y_i differ from the steady-state distribution F, since it will generally not be possible to choose the initial conditions I to be representative of steady-state behavior (Kelton and Law 2000). The second problem arises when the central limit theorem is used to calculate the confidence interval of the performance metric $v = \mathbb{E}[Y]$. This requires the knowledge of the time-average variance constant $\overline{\sigma}$ (Nakayama 2008). Estimating this parameter by the sample variance S^2 is not valid for non-i.i.d. data.
Welch's procedure is used to provide an estimation for the warm-up period length k, see (Kelton and Law 2000, Welch 1983). For each simulated model, 5 simulations were used for Welch's method, resulting in an estimation l of k. The length of the performance simulations m after determining the initial transient length is set to $m = 4l$. To deal with non-i.i.d. data, the method of multiple replications with initial-data deletion is used, see (Kelton and Law 2000) for an introduction. We make $n' = 5$ replications for each simulated model, each with length $m = 4l$. Furthermore, the first l observations are assumed to be contaminated with the initial transient bias and all observations after l are not significantly biased. Let Y_{ji} be the i-th observation of the j-th run and let X_j be given by

$$X_j = \frac{\sum_{i=l+1}^{m} Y_{ji}}{m-l} \quad \text{for } j = 1, 2', \ldots, n'.$$

Then the X_j's are i.i.d. random variables(Kelton and Law 2000). The sample mean $\overline{X}(n')$ is then an approximately unbiased point estimator for v and the approximate $100(1-\alpha)$ percent confidence interval for v is given by

$$\overline{X}(n') \pm t_{n'-1,1-\alpha/2}\sqrt{\frac{S^2(n')}{n'}}$$

where the value of $t_{n'-1,1-\alpha/2}$ can be found in any standard t-distribution table, for example in (Kelton and Law 2000).

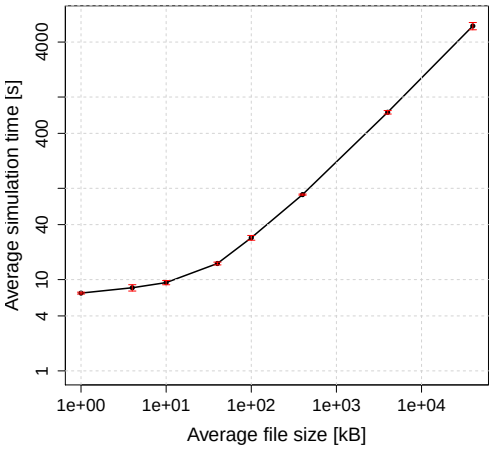

Figure 4: Average simulation time as a function of the average file size. The file sizes are geometrically distributed.

CHALLENGES

Several problems and challenges are related to simulating the ICT solution stack and is subcomponents. These problems are described in this section.

Increase simulation speed

The simulation time becomes a problem when real life ICT solution stacks are simulated. The simulation time seems to be independent of the number of database servers, scales linearly with the number of clients and the number of web servers, but the simulation time scales exponentially with the number of corporate servers, see (Goorden 2015). Furthermore, the average simulation time increases rapidly with the size of the files retrieved from the ICT solution stack, see Figure 4. The reason for this dependency on the file size is due to the way the TCP protocol is implemented in the simulation model. Each individual packet send is simulated. Stochastic fluid models may provide a solution to increase the speed of the simulation (Olsén 2003).
Beside this, the stochastic output analysis requires multiple simulations, first to estimate the initial transient period, then to generate output data to be analyzed. Furthermore, with the method of multiple replications in combination with initial data deletion discards in total ln' data points. In (Whitt 1991) a comparison is made between single-replication and multiple replica-

tion methods. Only when the initial transient period l is large or when the covariance function decreases more quickly, a single-replication method is preferred over a multiple replication method. Otherwise, they are both equally efficient.

Automatic initial transient period estimation

For most algorithms analyzing the stochastic output process it is needed that the initial transient is filtered out of the data. Most of the current techniques to do this filtering require knowledge or intervention from the modeler.

Automating this process of determining the initial transient period has two advantages. First, it allows the modeler to fasten the simulation preparation process. Secondly, it allows for automatic stochastic simulation design. Therefore, the ICT architect does not need to have a thorough knowledge of stochastic analysis.

This problem is known in the literature, see for example (Hoad et al. 2010, Robinson 2005). Unfortunately, both papers lack a clear solution for automatic initial transient period estimation.

Rare event simulations

An interesting aspect of simulation is to consider several what-if scenarios. One of these situations is failure behavior of the system. Failures do not occur often in the system, so the simulation time of a standard simulation to analyze the failure behavior would be high. The area of rare event simulation focusses on developing efficient simulation algorithms. The addition of rare event simulations allows the ICT architect to analyze the reliability of the current ICT solution stack design.

Good references are available in the literature. Two introductions to this research area are (Bucklew 2013, Görg et al. 2002). Some specific papers of different rare event methods are (Garvels 2000, Glasserman et al. 1996, Lagnoux 2006, Rubinstein 1997).

CONCLUSION

A simple ICT solution stack model is implemented in the simulation language χ. Small scale problems can be simulated with the current implementation. Scaling the simulation model to the size of a real life system poses some challenges. Especially the way the TCP protocol is implemented limits the simulation of transfer of large files. To have an impact in the industry, the simulation model should be developed further.

REFERENCES

Bucklew J., 2013. *Introduction to rare event simulation.* Springer Science & Business Media.

Eindhoven University of Technology, 2015. *Chi 3 home page.* `http://chi.se.wtb.tue.nl/index.html`. Accessed on: 10-03-2015.

Garvels M.J.J., 2000. *The splitting method in rare event simulation.* Universiteit Twente.

Glasserman P.; Heidelberger P.; Shahabuddin P.; and Zajic T., 1996. *Splitting for rare event simulation: analysis of simple cases.* In *Proceedings of the 28th conference on Winter simulation.* IEEE Computer Society, 302–308.

Goorden M., 2015. *End-to-End Performance Modeling of ICT Solution Stacks.* Masters thesis, Eindhoven University of Technology.

Goorden M.; van de Mortel-Fronczak J.; Reniers M.; Kinder M.; and van Veggel W., 2016. *End-to-end Performance Modeling of ICT Solution Stacks.* Submitted.

Görg C.; Lamers E.; Fuß O.; and Heegaard P., 2002. *Rare event simulation.* Springer.

Gray M.A., 2007. *Discrete event simulation: A review of SimEvents. Computing in Science & Engineering*, 9, no. 6, 62–66.

Hoad K.; Robinson S.; and Davies R., 2010. *Automating Warm-up Length Estimation. Journal of the Operational Research Society*, 61, no. 9, 1389–1403.

Kelton W.D. and Law A.M., 2000. *Simulation Modeling and Analysis.* McGraw Hill Boston.

Lagnoux A., 2006. *Rare event simulation. Probability in the Engineering and Informational Sciences*, 20, no. 01, 45–66.

Nakayama M.K., 2008. *Statistical Analysis of Simulation Output.* In *Proceedings of the 40th Conference on Winter Simulation.* Winter Simulation Conference, 62–72.

Olsén J., 2003. *Stochastic modeling and simulation of the TCP protocol.* Ph.D. thesis, Mathematics and Computer Science, Uppsala University.

Robinson S., 2005. *Automated analysis of simulation output data.* In *Proceedings of the Winter Simulation Conference.* IEEE, 8–pp.

Rubinstein R.Y., 1997. *Optimization of computer simulation models with rare events. European Journal of Operational Research*, 99, no. 1, 89–112.

van Beek D.A.; Man K.L.; Reniers M.A.; Rooda J.E.; and Schiffelers R.R., 2006. *Syntax and consistent equation semantics of hybrid Chi. The Journal of Logic and Algebraic Programming*, 68, no. 1, 129–210.

Welch P.D., 1983. *The statistical analysis of simulation results. The computer performance modeling handbook*, 22, 268–328.

Whitt W., 1991. *The efficiency of one long run versus independent replications in steady-state simulation. Management Science*, 37, no. 6, 645–666.