Filling the Gap: a Tool to Automate Parameter Estimation for Software Performance Models

Weikun Wang, Juan F. Pérez, Giuliano Casale
Department of Computing
Imperial College London
UK
weikun.wang11,j.perez-bernal,g.casale@imperial.ac.uk *

ABSTRACT

Software performance engineering heavily relies on application and resource models that enable the prediction of Quality-of-Service metrics. Critical to these models is the accuracy of their parameters, the value of which can change with the application and the resources where it is deployed. In this paper we introduce the Filling-the-gap (FG) tool, which automates the parameter estimation of application performance models. This tool implements a set of statistical routines to estimate the parameters of performance models, which are automatically executed using monitoring information kept in a local database.

1. INTRODUCTION

DevOps [5] is a recent trend in software engineering that bridges the gap between software development and operations, putting the developer in greater control of the application operational environment. To support Quality-of-Service (QoS) analysis, the developer may rely on software performance models. However, to provide reliable estimates, the input parameters must be continuously updated and accurately estimated. Accurate estimation is challenging because some parameters are not explicitly tracked by log files, requiring deep monitoring instrumentation that poses large overheads, unacceptable in production environments.

In this paper we present the first release of the Filling-the-Gap (FG) tool, a tool for continuous performance model parametrization that implements the research agenda set in [4]. The FG tool implements a set of statistical estimation algorithms to parameterize performance models from runtime monitoring data. Multiple algorithms are included, allowing for alternative ways to obtain estimates for different metrics, but with an emphasis on resource demand estimation, which has recently been contemplated also in tools such as LibReDE [6]. A distinguishing feature of FG tool is that it supports advanced algorithms to estimate parameters based on response times and queue-length data, which makes the tool useful in particular for applications running in virtualized environments where utilization readings are not always available. In addition, the FG tool offers support for parallel computations, integrates monitoring data acquisition, and generates performance reports.

*The research leading to these results has received funding from the European Union Seventh Framework Programme FP7/2007-2013 under grant agreement no. 318484.

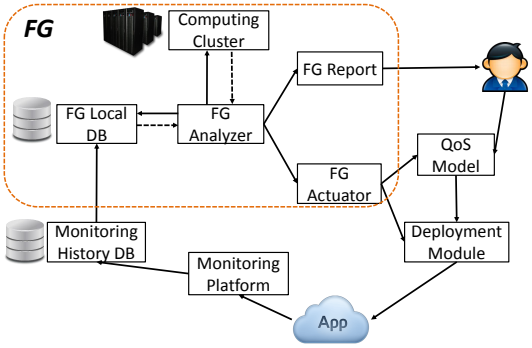


Figure 1: FG Architecture

The FG tool is composed of four main components: the FG Local DB, the FG Analyzer, the FG Actuator, and the FG Reporter. We now describe these components and their interactions, depicted in Figure 1.

FG Local DB: This local database periodically acquires and stores the monitoring data that is relevant for FG analysis and used by the FG Analyzer. The Local DB is a Fuseki[1] database, which keeps data in RDF format.

FG Analyzer: The FG Analyzer executes the statistical methods necessary to obtain the estimates of the performance models parameters, relying on the monitoring information available on the Local DB. It has the ability to connect with a Condor cluster to process estimation routines in parallel. The FG Analyzer relies on the Matlab Compiler Runtime for execution.

FG Actuator: The FG Actuator updates the parameters of the models, e.g., resource demands, think times, etc., which are obtained from the FG Analyzer. The update is performed on both the performance and the deployment model, e.g. a Palladio Component Model[2], making this information available to the developer.

[1] `http://jena.apache.org/documentation/serving_data/`
[2] `https://sdqweb.ipd.kit.edu/wiki/Palladio_Component_Model`

FG Reporter: The FG Reporter is in charge of providing the developers with information regarding the application runtime behavior to help them evaluate how well the application responds under different conditions. The FG Reporter is based on DynamicReports[3].

2. SUPPORTED DEMAND ESTIMATION ALGORITHMS

Queueing networks and layered queueing networks are popular abstractions used as application performance models. To parametrize these models, the FG tool uses the monitoring data collected at runtime and executes statistical methods, improving the accuracy of the models. Among the set of model parameters, resource consumption, also called demand, is difficult to estimate since extensive monitoring poses unacceptable overheads. To tackle this problem, the FG Analyzer implements several algorithms relying on different monitoring metrics. Below we provide a short description of some of these methods. Here we use D_r to represent the service demand of request class r.

CI: the Complete Information (CI) method [3] requires a full trace, that is, the times at which every request arrives and departs from the resource. Consider a class-r request that arrives at t_1 and departs at t_I, where I is the number of observed events (request arrivals or departures) during the request execution. Then the demand of that request on that resource is:

$$D_r = \sum_{i=1}^{I-1}(t_{i+1} - t_i)\min(n(t_i^+), V)/n(t_i^+)$$

where $n(t_i^+)$ is the number of requests in execution just after time t_i and V is the number of CPUs.

GQL: the Gibbs sampling with Queue Lengths (GQL) method [7] uses queue-length samples collected at run time to estimate the service demand with the Bayes' theorem:

$$P(\boldsymbol{D}|\boldsymbol{N}) = P(\boldsymbol{N}|\boldsymbol{D})P(\boldsymbol{D})/P(\boldsymbol{N}) \approx \prod_{\boldsymbol{n}\in\boldsymbol{N}} P(\boldsymbol{n}|\boldsymbol{D})P(\boldsymbol{D}),$$

where $P(\boldsymbol{n}|\boldsymbol{D})$ is the steady state probability of a product form queueing network, $\boldsymbol{N}$ is the observed queue length dataset and $\boldsymbol{n}$ is one entry of the dataset $\boldsymbol{N}$, i.e. n_{ir} is the number of class-r jobs at station i. Gibbs sampling is employed to obtain the demand $\boldsymbol{D}$.

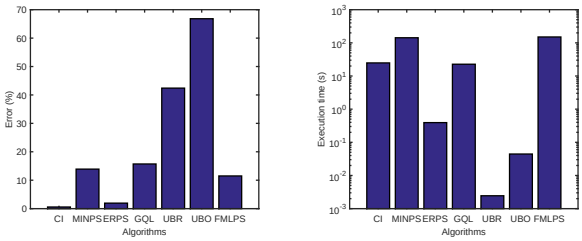
MINPS/FMLPS: the MINPS method [3] is a maximum likelihood method based on a Markov Chain representation of the response time given the observed queue length. It requires response times and queue lengths observed upon arrival. Similarly, FMLPS [3] is also a maximum likelihood estimator but uses a fluid approximation to obtain estimates for large systems.

ERPS: the Extended Regression-Based (RPS) approach [3] makes use of the response times and queue lengths observed at arrival times as in the equation

$$E[R_r] = E[D_r]E[\bar{A}^r]/\min\left\{V, 1/I\sum_{i=1}^{I} n(t_i)\right\},$$

where A^r is the queue-length seen upon arrival by class-r jobs, and R_r is their response time. The service demand is estimated using linear regression.

[3] `http://www.dynamicreports.org/`



(a) Error (%) (b) Execution time (s)

Figure 2: Comparison between demand estimation algorithms

FCFS: estimations for FCFS servers [1] rely on

$$E[R_r] = E[T_r] + \sum_{k=1}^{K} E[D_r](1_{k,r} + E[A_k^r]),$$

where $1_{k,r} = 1$ if $k = c$ otherwise $1_{k,r} = 0$, T_r is the residual time to completion of a class-r request, and A_k^r is the queue-length of class-k jobs seen upon arrival by each class-r request. Demand estimates $E[D_r]$ are obtained with regression methods given A_k^r and R_r.

2.1 Evaluation

One advantage of the FG tool is the availability of different estimation algorithms in a common environment. We make use of the tool to provide a novel comparison of these algorithms. We simulate the underlying Markov chain of a closed network with 4 request classes, 2 queueing stations, and one delay node. The queueing stations have 2 servers and the number of jobs for each class is $[7, 7, 1, 5]$. We simulate a total of 200000 arrival and departure events, and generate all the metrics required by the estimation algorithms.

Figure 2 shows the experiment result, including the UBR [8] and UBO [2] methods, which use CPU utilization measurements. We observe that CI achieves the highest accuracy while UBR is the most efficient one.

3. REFERENCES

[1] Kraft, S., Pacheco-Sanchez, S., Casale, G., Dawson, S.: Estimating service resource consumption from response time measurements. In: VALUETOOLS, 2009.

[2] Liu, Z., Wynter, L., Xia, C.H., Zhang, F.: Parameter inference of queueing models for IT systems using end-to-end measurements. Perf. Eval, 63(1):36–60,2006.

[3] Perez, J.F., Casale, G., Pacheco-Sanchez, S.: Estimating computational requirements in multi threaded applications. IEEE TSE 41(3): 264–278, 2014.

[4] Perez, J.F., Wang, W., Casale, G.: Towards a devops approach for software quality engineering. In: WOSP, 2015.

[5] Roche, J.: Adopting DevOps practices in quality assurance. CACM, 56(11), 2013.

[6] Spinner, S., Casale, G., Zhu, X., Kounev, S.: Librede: a library for resource demand estimation. In: ICPE, 227–228 ,2014.

[7] Wang, W., Huang, X., Qin, X., Zhang, W., Wei, J., Zhong, H.: Application-level cpu consumption estimation: Towards performance isolation of multi tenancy web applications. In: IEEE CLOUD, 2012.

[8] Zhang, Q., Cherkasova, L., Smirni, E.: A regression-based analytic model for dynamic resource provisioning of multi-tier applications. In: ICAC, 2007.