Chapter 10
Closing the Loop Between Ops and Dev

Weikun Wang, Giuliano Casale and Gabriel Iuhasz

10.1 Introduction

DevOps [1] is a recent trend in software engineering that bridges the gap between software development and operations, putting the developer in greater control of the operational environment in which the application runs. 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.

The MODAClouds Filling-the-Gap (FG) tool is a component for parametrization of performance models designed in MODAClouds continuously at run time. 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. 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

W. Wang · G. Casale (✉)
Department of Computing, Imperial College London, 180 Queens Gate,
London SW7 2AZ, UK
e-mail: g.casale@imperial.ac.uk

W. Wang
e-mail: weikun.wang11@imperial.ac.uk

G. Iuhasz
Institute E-Austria Timişoara, West University of Timişoara, B-dul Vasile Pârvan 4,
300223 Timişoara, Romania
e-mail: iuhasz.gabriel@info.uvt.ro

E. Di Nitto et al. (eds.), *Model-Driven Development and Operation of Multi-Cloud Applications*, PoliMI SpringerBriefs,
DOI 10.1007/978-3-319-46031-4_10

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.

10.2 FG Architecture

The FG tool is consisted of four sub-components: the Local DB, the FG Analyzer, the FG Reporter and the FG Actuator. Figure 10.1 descries the relation between each component.

We show here a brief introduction of each component:

- The Local DB is a local database, which is built upon the Fuseki[1] database. The Local DB is in charge of periodically obtaining runtime monitoring data that will be used by the FG Analyzer from the Monitoring History DB. Due to the nature of Fuseki database, the monitoring data will be stored in RDF format in the local DB.
- The FG Analyzer is the main component of the FG and will be described in Sect. 10.2.1. After receiving runtime data stored in the Local DB, the FG Analyzer provides accurate estimates to parametrise the design-time Quality-of-Service (QoS) models developed in MODAClouds. These parameters include the resource demand, the think time and the total number of jobs running in the system.
- The FG Reporter, illustrated in Sect. 10.2.3, periodically generates reports on the application behavior at run time. The reports shows the performance of the application by presenting performance metrics such as the response time and the throughput of the jobs.
- The FG Actuator (see Sect. 10.2.2) is responsible for updating the IDE models and the QoS models based on the result from the FG Analyzer.

10.2.1 FG Analyzer

One of the ultimate objectives of the Filling the Gap (FG) component is to provide accurate estimates to the parameters in the design-time QoS models. These QoS models are key in the what-if analysis performed at design time, and in the decision of the optimal resource provisioning for the Cloud application. These models are initially parameterised using expert-knowledge or data collected in small deployments. Once the application has been deployed on the Cloud, possibly in a production environment, the FG analysis is deployed to obtain estimates based on monitoring data collected at run time.

[1] http://jena.apache.org/documentation/serving_data/.

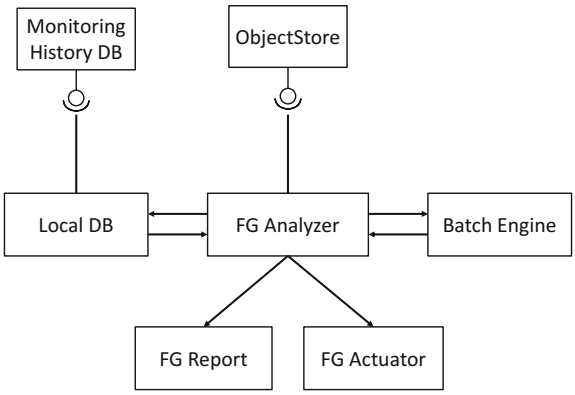


Fig. 10.1 FG architecture

The QoS models developed in MODAClouds are based on layered queueing network models, which capture the contention between users for the available hardware and software resources, and the interaction between them. In particular, we make use of closed models that are well-suited for software systems, as real applications are layered, and the interactions between layers are typically due to admission control or finite threading limits [2]. To parameterise these models, it is essential to estimate the inter-request times, modeled as think times, as well as the resource consumption exerted by each request. Inter-request times can be extracted from the information and the data that is typically tracked by application- or container-level logs. As to the gathering of the run time configuration, the FG Analyzer obtains the configuration file from the Object Store which is kept by the QoS engineer.

Resource consumptions, also referred to as demands, are however harder to obtain as this is not tracked by logs, and the deep monitoring instrumentations typically required pose unacceptably large overheads, especially at high resolutions. Since requests typically complete in a few milliseconds, individual monitoring becomes cost-expensive to perform in a production system. To address this problem, our approach is to take coarse-grained measurements and apply statistical inference to estimate mean resource demands. Most of existing mean demand estimation approaches rely on the regression against utilization data [3–13], however, utilization measurements are not always available, for instance in Platform-as-a-Service (PaaS) deployments where the resource layer is hidden to the application and thus protected from external monitoring.

In the FG Analyzers, two demand estimation algorithms, **GQL** (Gibbs sampling method with Queue Length data) and **MINPS**, have been proposed as an original contribution within the MODAClouds research [14]. The fact that utilization measurements are not required makes these methods suitable for applications deployed on both IaaS and PaaS. In addition to these two methods, the FG Analysis component implements existing demand estimation methods. In particular, the component supports the methods implemented for the Statistical Data Analyzers (SDA) in the Monitoring Platform.

Since the methods supported by the FG Analysis are computationally efficient, large sample set can be utilized for the analysis. The FG component thus supports the following three demand estimation methods: the utilization-based optimization (UBO) method from [15], the utilization-based regression (UBR) method from [12], and the Extended RPS method from [16]. A short description of these methods is provided in Sect. 10.4.

Finally, the FG Analyzer calls the Batch Engine periodically and executes several jobs on multiple nodes performing different analyses. For instance, the FG Analyzer can execute several demand estimation procedures in parallel using the Batch Engine to compare the accuracy of them during design time. It also executes the analysis corresponding to different datasets in parallel, thus speeding up the analysis phase.

10.2.2 FG Actuator

In order to improve the accuracy of the design-time QoS models developed in WP5, the FG tool estimates the parameters of the models with the monitoring information collected at runtime. Then the task of updating the actual model is fulfilled by FG Actuator, which updates the resource demand, think time, number of users circulating in the system in both the QoS models and PCM models given the input from the FG Analyzer.

Since the QoS models and PCM models may have inconsistent names for the deployed resources, the FG actuator requires a properties file indicating the mapping of the resource names between the two models. In addition, the name of the job classes could be different from the data analyzers and the models. A job class mapping file should also be provided.

Given the path to the model files, the FG Actuator first updates the resource demands in the QoS models by matching the resource and job class names. Then it obtains an id for the particular resource and class of job. This id is identical to the one defined in the PCM model. Therefore the FG Actuator uses this id to update the resource demand in the PCM model. Updating the think time and number of jobs in the system is straightforward by just changing the corresponding fields in the XML file.

10.2.3 FG Reporter

In order to provide the developer with runtime information of the application behavior at runtime, the FG periodically generates a report. The report is a PDF document containing tables and figures of performance metrics such as response time, resource demands and throughput, which helps the developer to identify periods of high and low load, as well as to understand the application behavior under the different scenarios.

The automatically report generation relies on the DynamicReports,[2] which is an open-source library based on JasperReports[3] for generating reports based on complex datasets. The DynamicReports supports a wide range of data formats, including relational databases, XML, XLS, and CVS files, among others. In particular, we utilized its ability to integrate JSON (JavaScript Object Notation) format, as this format is expressive and easily understandable.

The FG Reporter periodically receives JSON files generated from the FG Analyzer, which contains necessary information regarding the application such as the think time, response time, resource demands, etc. Based on these information, the FG Reporter generates a different report for each physical resource.

10.3 Workflow

In the previous sections we have described the essential components of the FG tool, here we present the workflow for the FG tool. The operation of the FG can be categorized into three main stages, which are:

1. Configuration: this is a design-time procedure for the QoS engineer to preconfigure the FG Analyzer through the MODAClouds IDE.
2. Analysis: this is a runtime step performed by the FG Analyzer with the Local DB.
3. Reporting/Updating: this is a step where the FG Reporter provides the developer with a report regarding the behavior of the application at runtime. The FG Actuator will also update the parameters of the QoS models given the output from FG Analyzer. This steps is performed after the application has already been running as it requires the results from the FG Analyzer.

The FG workflow is demonstrated in Fig. 10.2, which contains all the above three main stages. As mentioned in the previous section, the developer configures with the FG Analzyer through the MODAClouds IDE according to a configuration file, which is saved in the Object Store. The configuration file includes parameters such as the frequency to execute the FG Analyzer or the time period of the monitoring data to use. This configuration file is retrieved at deployment by the FG Analyzer. Then the Local DB periodically queries the Monitoring History DB to obtain the necessary information for the FG Analyzer. This data is passed to the FG Analzyer for the parameter estimation. With the estimation result, the FG Reporter will produce reports to the developer while the FG Actuator updates the QoS PCM models.

[2] http://www.dynamicreports.org/.

[3] https://community.jaspersoft.com/project/jasperreports-library.

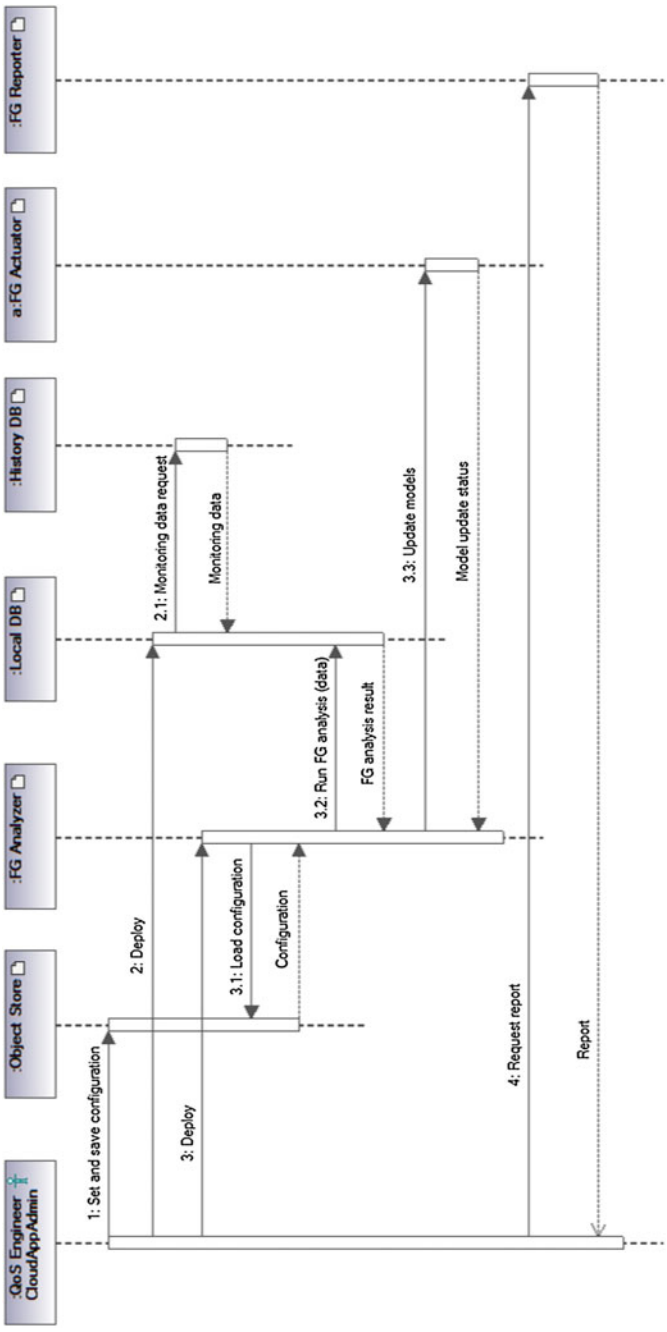


Fig. 10.2 Filling the gap workflow

10.4 Estimation Techniques for FG Analysis

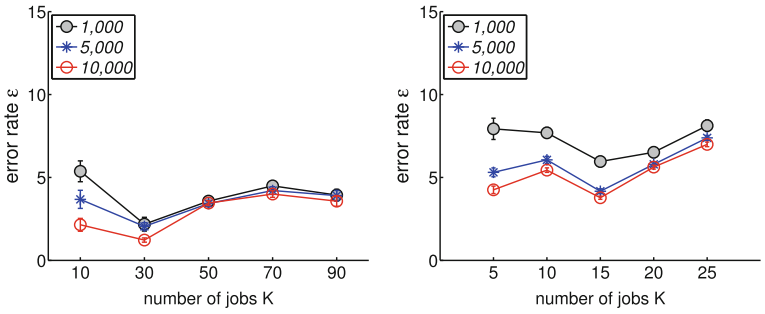
10.4.1 A Bayesian Approach Based on Queue-Lengths

Closed queueing networks have been used for analyzing web applications [12, 17]. They are popular for example in software system modelling since complex applications are layered and the interactions between layers typically happen under admission control or finite threading limits.

The proposed GQL estimation method sets out to estimate the service demand placed by requests on the resources excluding contention due to other concurrently running requests. The service demand is normally difficult to obtain directly and requires inference. To provide these estimates, out method uses observations of the number of requests in each of the queueing stations, which makes it more applicable than utilization-based and response-based methods as the latter information may not be available in certain environments, such as PaaS deployments, or require deep instrumentation of the system.

Our method uses a Bayesian approach to estimate the mean demands, of which there has already been some attention in the recent literature [8, 18]. Still, with the exception of [18], classic Bayesian methods such as Markov-Chain Monte Carlo (MCMC) have not been applied before to the problem of queueing model parameter estimation. Even though the method in [18] is promising, it currently only applies to open queueing networks and single class systems. Our method, instead, is based on MCMC estimation with Gibbs sampling, and has the advantage of applying to closed multi-class models.

Figure 10.3 presents the experiment result for the GQL method with different number of classes of requests and queueing stations. The estimation error is computed as the mean relative difference between the estimated and the exact (known) value of the resource demand. From the figure, it can be noticed that the estimation error is under 10 %, showing the good accuracy of the GQL method. The execution time of



(a) 1 class, 1 station (b) 4 classes, 4 stations

Fig. 10.3 Mean estimation error for GQL

the GQL method depends on the input parameters of the developed algorithm. The running time for the presented case is 15 min, which shows that the algorithm is able to handle systems with a larger number of processing stations and request classes.

A detailed description of this method can be found in [19].

10.4.2 A Maximum-Likelihood Approach Based on Queue-Lengths and Response Times

Another proposed method, MINPS, is similar to the GQL presented in the previous section as MINPS also attempts to estimate the mean service demands placed by requests on the physical resources.

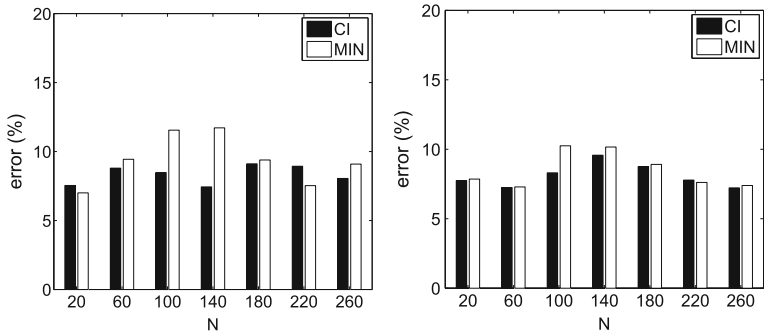
The performance model for MINPS is based on a multi-class queueing network with a single service station. It also considers the limit in the number of concurrent request in a station, which enables the analyzing of multi-threaded applications with limits on the number of threads in execution. A typical example is for applications running on a multi-threaded server, such as an application server or a servlet container with a preconfigured set of worker threads. Arriving requests to the application will stay in an admission buffer until a worker thread is available. We assume the admission control policy is first-come first-served and no workers are idle if there is a request staying in the admission buffer. Therefore the described performance model is indeed a closed queueing network similar as described in the previous section. Further, a request is able to change its class randomly after leaving the queueing station before entering the think time. This class-switching behavior represents systems where users may change the type of requests they generate.

The proposed MINPS estimation method is built on top of two new estimation approaches, RPS and MLPS. RPS is a regression based algorithm, which provides accurate estimation of mean service demand for multi-threaded application running on a single processor. For the multi-processor case, the proposed MLPS is able to solve this problem relying on a maximum likelihood demand estimation algorithm. MINPS integrates RPS and MLPS to produce accurate estimates at all loads of the multi-threaded applications.

MINPS differs from existing approaches in that, to the best of our knowledge, it is the first one to apply probabilistic descriptions in estimation problems for multi-threaded applications. For example, maximum likelihood estimations have been attempted only for simpler first-come first-served queues [8].

MINPS requires both queue lengths and response times as input metrics. These metrics can be obtained in several ways, e.g., the MODAClouds application-level data collectors, application server logs, internal application logs, etc.

Figure 10.4 demonstrates the mean estimation error of the MINPS method, compared with a baseline method CI with same sample size. As in the previous section, the estimation error is computed as the mean relative difference between the estimated and the exact (known) value of the resource demand. The CI method is an estimation method that requires the complete sample path of the requests, i.e. given



(a) 2 CPU, 8 worker threads (b) 4 CPU, 8 worker threads

Fig. 10.4 Mean estimation error for MINPS

a time window it knows all the points in time when a request is admitted and when it completes service. This information is difficult to collect, but it is useful to set a baseline for comparison, as both methods are assumed to make use of the same number of samples.

From Fig. 10.4, it can be noticed that the error of the MINPS and CI is similar, which reveals the accurate performance of MINPS. Although MINPS generates a larger estimation error, it is still under 15 %.

The execution time of MINPS depends on the model and obtained samples size and varies from 1 to 40 min for small models to large models. In light of this, the technique can be run periodically as part of the FG analysis.

A detailed description of these methods and additional validation results are provided in [16].

10.5 Conclusion

In this chapter we presented the MODAClouds Filling-the-Gap tool, which is a DevOps approach aiming to fulfill the gap development and operations. The FG tool supports a set of advanced algorithms for estimating the parameters of performance models at application runtime. Algorithms differ in the way that they take into consideration of different input monitoring metrics, which makes the tool useful particularly for application deployed in Cloud. It also features generating reports regarding the behavior of the application to give developers timely feedback of the system.

References

1. Roche J (2013) Adopting DevOps practices in quality assurance. Commun ACM 56:38–43

2. Rolia JA, Sevcik KC (1995) The method of layers. IEEE Trans Softw Eng 21(8):689–700
3. Kalbasi A, Krishnamurthy D, Rolia J, Dawson S (2012) Dec: service demand estimation with confidence. IEEE Trans Softw Eng 38:561–578
4. Kalbasi A, Krishnamurthy D, Rolia J, Richter M (2011) MODE: mix driven on-line resource demand estimation. In: Proceedings of IEEE CNSM
5. Wang W, Huang X, Qin X, Zhang W, Wei J, Zhong H (2012) Application-level CPU consumption estimation: towards performance isolation of multi-tenancy web applications. In: Proceedings of the 5th IEEE CLOUD
6. Cremonesi P, Dhyani K, Sansottera A (2010) Service time estimation with a refinement enhanced hybrid clustering algorithm. In: Analytical and stochastic modeling techniques and applications, ser. Lecture notes in computer science. Springer, Berlin
7. Cremonesi P, Sansottera A (2012) Indirect estimation of service demands in the presence of structural changes. In: QEST
8. Kraft S, Pacheco-Sanchez S, Casale G, Dawson S (2009) Estimating service resource consumption from response time measurements. In: Proceedings of the 4th VALUETOOLS
9. Kumar D, Zhang L, Tantawi A (2009) Enhanced inferencing: estimation of a workload dependent performance model. In: Proceeding of the 4th VALUETOOLS
10. Menascé D (2008) Computing missing service demand parameters for performance models. In: CMG 2008, pp 241–248
11. Pacifici G, Segmuller W, Spreitzer M, Tantawi A (2008) CPU demand for web serving: measurement analysis and dynamic estimation. Perform Eval 65:531–553
12. Zhang Q, Cherkasova L, Smirni E (2007) A regression-based analytic model for dynamic resource provisioning of multi-tier applications. In: Proceedings of the 4th ICAC. Washington, DC, USA. IEEE Computer Society, p 27ff
13. Zheng T, Woodside C, Litoiu M (2008) Performance model estimation and tracking using optimal filters. IEEE Trans Softw Eng 34:391–406
14. Ardagna D, Nitto ED, Casale G, Petcu D, Mohagheghi P, Mosser S, Matthews P, Gericke A, Ballagny C, D'Andria F (2012) Modaclouds: a model-driven approach for the design and execution of applications on multiple clouds. In: Proceedings of the 4th international workshop on modeling in software engineering
15. Liu Z, Wynter L, Xia CH, Zhang F (2006) Parameter inference of queueing models for IT systems using end-to-end measurements. Perform Eval 63(1):36–60
16. Pérez JF, Pacheco-Sanchez S, Casale G (2013) An offline demand estimation method for multi-threaded applications. In: MASCOTS, pp 21–30
17. Urgaonkar B, Pacifici G, Shenoy PJ, Spreitzer M, Tantawi AN (2005) An analytical model for multi-tier internet services and its applications. In: Proceedings of ACM SIGMETRICS. ACM Press, pp 291–302
18. Sutton C, Jordan MI (2011) Bayesian inference for queueing networks and modeling of internet services. Ann Appl Stat 5(1):254–282
19. Wang W, Casale G (2013) Bayesian service demand estimation using gibbs sampling. In: MASCOTS, pp 567–576