A Configuration Driven Modeling Approach for Resources Optimization of Multi-tenant Applications

Xiang Huang
Guang Dong Electric Power Design Institute, China Energy Engineering Group Co. Ltd., Guangdong, China
School of Information Science and Technology, Sun Yat-Sen University, Guangdong China
Email:huangxiang@gedi.com.cn

Zhi-gang Chen
Guang Dong Electric Power Design Institute, China Energy Engineering Group Co. Ltd., Guangdong, China
Email: chenzhigang@gedi.com.cn

***Abstract*—Multi-tenancy application is one of the important resource sharing approaches which enables sharing of resources across a large pool of users. A key requirement to manage the resources fairly among tenants is that one should know the impacts of the resource allocations strategies. Performance model has the native advantages to predict the impacts in advance. However, as the platform become more difficult than ever before, it is hard to build the model manually. In this paper, we provide a configuration driven modeling approach to automatically build the configurations' performance impacts. In our work, we firstly decouple the impacts into several templates which can be instanced with the runtime configurations. And then, these models can be merged with the tenant's model as a holistic performance model, which can be used to predict performance under different configurations. Experimental results derived from testing the approach by using an online application deployed on a widely-used platform illustrate the potential of this approach.**

***Index Terms*—Configuration Driven, Resources Optimization, Multi-tenant**

I. Introduction

Multi-tenancy is one of the important application models of cloud computing [1]. The most advantage of this application model is that it let lots of tenants (application) share the same platform. Therefore, it can reduce the operation and management cost for each tenant. As show in figure 1, there are 3 typical multi-tenancy deployment types: the method based on virtual machines (VM-Based), the method based on process (P-Based), and the method based on shared middleware (M-Based). VM-Based method allocates one VM instance for each tenant, such as the Amazon's EC2 [2]. P-Based method allocates one process for each tenant, such as Google's App Engine [3]. M-Based method share all tenant applications in one middleware, such as Intalio [4] and Salesforce [5].

Compare with other types, M-Based type can share the whole platform except the data related to the users, therefore the tenants can share the resources more reasonably, and reduce the cost introduced by the underline platform [6]. For example, on the same physical machine, VM-based method can only support 3 tenants in one CPU, but M-based method can support dozens or hundreds tenants [6]. However, as the resources are shared by many tenants, resources are easily occupied by CPU intensive tenants. Thus, other tenants' performance will be dramatically deteriorated. This phenomenon makes it hard to optimize the resources allocations.

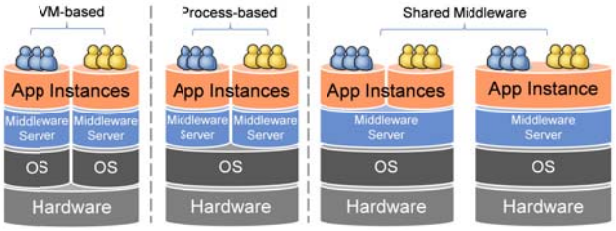

Figure 1. Different type of multi-tenancy application

A key requirement for performance optimization is the knowledge of the impacts of different configurations under different kinds of configurations in advance. Performance models [7], e.g. Layer Queue Network (LQN) [8] [9] model, have native advantages to make costly decisions, thanks to their relative low costs of performance predication ability. But there are two main challenges to build such a performance model for a multi-tenancy application. First, some critical configuration parameters of the platform, such as resource sharing policy and concurrency level, can only be determined at runtime. Thus it is unable to build a performance model for it directly. Second, although the platform's impact is very complex, it can be reused among different applications. To build the model for each particular deployment scenes is a time consume and error-prone task.

In this paper, we propose a configuration driven modeling approach to analyze the impacts of the platform configurations. Firstly, the platform is decoupled into

doi:10.4304/jsw.8.10.2432-2438

several interacting aspect models. The configuration parameters that can only be determined at runtime are modeled as dynamic elements in these models. Secondly, these aspect models can be instantiated with their runtime status and weaved with the tenant models into a holistic model of the entire system. Then the optimal configuration can be found out based on the predicted results. Our experimental results have shown the effectiveness of our approach.

Our main contribution is that we propose a stateful aspect-oriented modeling approach for automatically including the shared resources managed by the platform into multi-tenancy application. Compared with existing works, our approach distinguishes itself by: 1) its dynamic manipulation of configuration parameters according to the resource sharing policy and 2) its automatically building of the performance models of the entire system. We also present a solid case study of modeling a complex multi-tenancy application to evaluate our approach.

The rest of this paper is organized as follows. Section 2 describes the overview of our work. Section 3 presents our modeling approach in detail. Section 4 evaluates our approach through a series of experiments. Section 5 introduces related works. The last section makes a summary.

II. Overview of Our Approach

Component based development method [10] is the most popular software paradigm for multi-tenancy applications, such as JEE, .NET, CORBA, .COM, and so on. A component-based application is built upon component types, each of which plays a specialized role in a system and is described by an interface. The application should be deployed on a middleware, which provide the basic service for runtime requirement.

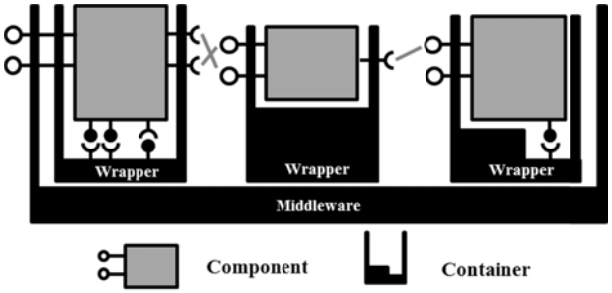

Figure 2. Component model of multi-tenancy application

The component cannot run by itself, it should be supported by a wrapper [11] (which is a micro middleware for a particular component). All of the wrappers are contained in the middleware. Middleware manage all the resources which the application needed if a component wants to use a resource, it first sends the request the wrapper, and the wrapper will forward the request to the middleware.

In this paper, we decouple the platform's impacts into several parts according to the wrapper's type, and each part is modeled as aspect model. Before the aspect models are woven with the tenants' model, we will instance them and generate a resource pool according to the configurations. Therefore, all of the resources merged into the tenants' model are collected from the pool. That is means the pool is stand for the middleware, and the aspect model is like a wrapper which just describes how does the component use the resources. The overview of our approach is given in figure 3.

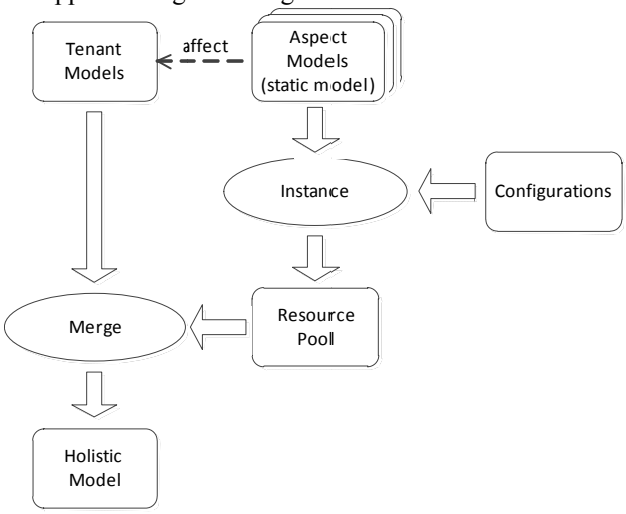

Figure 3. The overview of our approach

The key points of our approach are as follows: 1) Collecting the configuration parameters of the platform. 2) Instantiating the aspect models with their configurations to build a resources pool for the platform. 3) Merging the resources pool with the tenant model into a holistic LQN model of the entire multi-tenancy application. 4) Predicting the performance with the LQN model to optimize the resource allocation.

III. Modeling Approach

In this section, we introduce how to instance the resources, and how to merge them with the tenant model.

A. Basic Elements of Aspect Model

An aspect model is composed of three parts: pointcut, advice and template. Pointcut is used to find the join points where the applications will interact with the platform. Templates describe a wrapper's internal logic. And advices is used merge the tenants model and the templates.

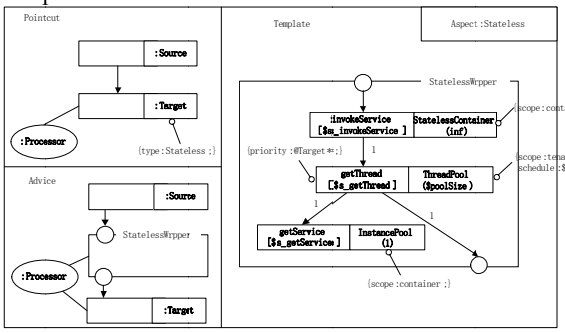

Figure 4. Aspect model of stateless component

In Figure 4, we give an example of stateless component. There are three middleware's components in the template: *container*, *thread pool* and *instance pool*. In advice, the internal detail of the wrapper is ignored, and only the position where the template should be weaved with the application is given. We also give a configurable profile extension for LQN to support the dynamic attributes. The details of the profile are given in table 1.

TABLE 1.
CONFIGURABLE PROFILE EXTENSION

	attribute	specification
symbol	$	Declare a parameter
	*	An optional parameter
	@	Reference to a specific component
	:	Reference the name of the matched component in advice
token	Scope	The scope of the resource： ● Component: for each component ● Tenant: for each tenant ● Container: for the whole container ● Server: for the whole server
	Schedule	Schedule strategies, FIFO (default),HOL,PPR
	Priority	Priority of the request
	Type	The type of the component
	Method	The type of the method supported by the component

For example in figure 4, the thread pool's capacity (*$poolSize*) is a mandatory parameter, and the schedule strategies (*$schedule**) is an optional parameter, whose priority is inherited from Target component (*@Target**). The thread pool would only share with the same tenant (*scope:tenant*), which means different tenant would use different thread pool. Therefore, the concurrency users of each tenant can be controlled. For each service or function of a component, there is a parameter for resource demands, such as *$s_invokeService* in *StatelessContainer*. There are several methods [12][13] to get this type of parameter.

B. Basic Aspect Models

Besides stateless component, there are several other typical component types, such as stateful component and data access components. The details of these types' wrapper are given as follows:

1) Stateful Component

Some component required to save the state of the user's operation. For example, in an online shopping application, same component need to save which items has already been choice by the user.

Figure 5 gives an example of stateful component's aspect model. There are three type of method: *create*, *remove* and *invoke*. Method *create* and *remove* is used for state management, and *invoke* is the other normal methods. Other part of the aspect model is the same as the stateless one.

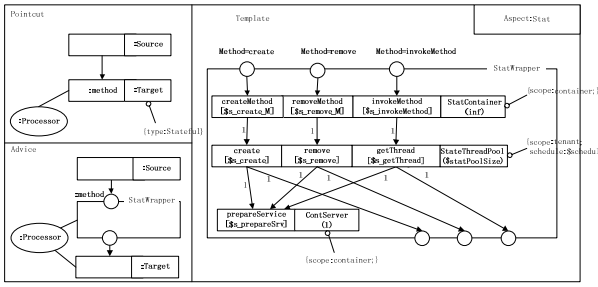

Figure 5. Aspect model of stateful component

2) Data Set Component

Data set component manage the relations between the persistent object and the storage data in database. Each persistent object corresponds to one row in database. To create or remove a persistent object means insert or delete one row in database.

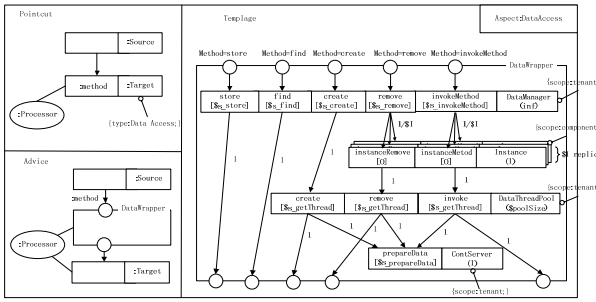

Figure 6. Aspect model of data set component

Aspect model of data set component is more complex than other type of components. Besides the thread pool and the critical section of the container, the data object is used exclusively. That means one object could not be used by two users concurrently. In template, we use instance to model the persistent object. We assume that the invocation probability of different objects is equal, so the chance of each object is *1/$I*, where *$I* is the number of objects.

C. Instantiation Approach

The aspect models describe the resources usage of a particular component, but the resources are shared by all components. In order to make holistic model consist with the runtime environment, we provide an instantiation method.

Figure 7 gives the algorithm of instantiation. The main point of our algorithm is that a new resource will be created and put into pool while there is not such resource existed in its scope, otherwise the resource in the pool.

Algorithm InstanceAdvice

Input: information, advice, generic aspect model;
Output: Advice.

1. **for** each task in generic aspect model **do**
2. get task's scope
3. **if** the scope existed in information **then**
4. add new entry into the task
5. put the task into advice

```
6.    else
7.       copy the task
8.       put it the information within the scope
9.       instance the parameter of the task
10.      add new entry into the new task
11.      put the task into advice
12.   end if
13.   if the entry need a priority then
14.      the priority is get form its scope
15.   end if
16. end for
17. connect the new entry set according to the
    generic aspect model
18. return advice
```

Figure 7. Instance algorithm

Figure 8 gives an example of instantiation. There are two templates in this example. Template 1 use 4 kinds of resources and template 2 use 3 kinds of resources. Resources c and d are shared, so there are 5 kinds of resources in the pool.

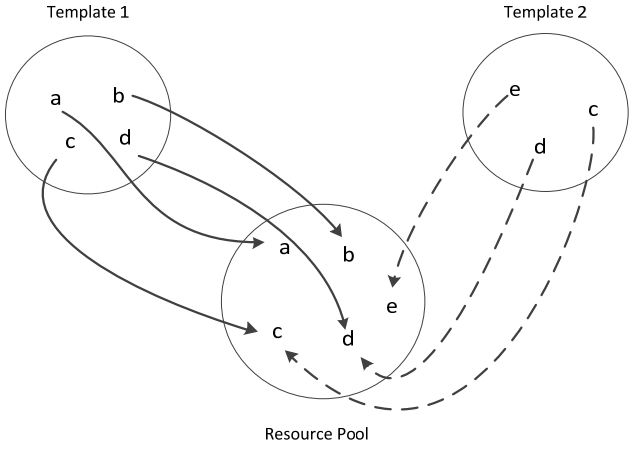

Figure 8. Example of instantiation

D. Weaving Approach

In our previous work, we can dynamically build an application model [14]. The workload characteristics, including the user scale and read/write ratio, can be modeled into a service invocation matrix with probability. In our approach, this application model should be woven with the platform model through its entrances, which are marked as jointcut. Because of the limited space, we will not describe the modeling of the application model in detail.

After the resource pool and the application model have been instantiated and built, they should be woven into a holistic LQN model. We use a third model, called pointcut, and two morphisms, which identify the elements of the application model that have to be kept, removed and added to those of the application model.

Let the application model, the pointcut model and the advice be three models, which are defined by a set of elements. The resources presented in advice are gotten from resource pool. Let *pj* and *pa* be two morphisms, such as (1) *pj* is a bijective morphism from pointcut to joinpoint, and (2) *pa* is an injective morphism from pointcut to the resource pool, and it can be obtained by the name of the elements. The morphisms *pj* is used to locate the components affected by the advice.

These two morphisms can divided the model *app* and *advice* into 5 sub sets:

- The set $\mathcal{R}_{keep}$ representing the set of objects of *app* which have to be kept. $\mathcal{R}_{keep} = \{e \in app | \nexists e' \in pointcut, f(e') = e\}$.
- The set $\mathcal{R}_{remove}$ representing the set of objects of *app* which have to be deleted. $\mathcal{R}_{remove} = \{e \in app | \exists e' \in pointcut, \nexists e'' \in advice, f(e') = e \wedge g(e') = e''\}$
- The set $\mathcal{R}_{duplicated}$ represents the set of elements of the *app* which have to be replaced. $\mathcal{R}_{duplicated} = \{e \in app | \exists e' \in pointcut, \exists e'' \in advice, f(e') = e''\}$.
- The set $\mathcal{R}^{ad}_{duplicated}$ represents the set of elements of the advice which describes the same objects as presented in $\mathcal{R}_{duplicated}$. $\mathcal{R}^{ad}_{duplicated} = \{e \in advice | \exists e' \in pointcut, \exists e'' \in app, g(e') = e \wedge f(e') = e''\}$.
- The set $\mathcal{R}_{add}$ represents the set of elements of the advice which have to be added in app model. $\mathcal{R}_{add} = \{e \in advice | \nexists e' \in pointcut, g(e') = e\}$.

The woven model is a set of elements and is denoted as weaved model = $\mathcal{R}_{\text{keep}} \cup \mathcal{R}_{\text{duplicated}} \cup \mathcal{R}_{\text{add}}$. That means we keep the set of elements that have not been affected by the cache system, the set of elements which have been affected but also existed in advice, and the set of objects which have been added in the application model.

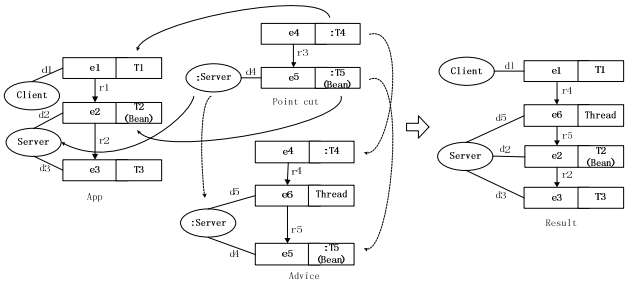

Figure 9. Weaving example

Figure 9 gives an example to illustrate the weaving algorithm. In this example, every task and event are seen as an element and the tenant model contains: {*e1*, *e2*, *e3*, *r1*, *r2*, *d1*, *d2*, *d3*,*Client*, *Server*}, The pointcut model contains: {*e4*,*e5*,*r3*,*d4*, *Server*}, And the advice model contains:{*e4*, *e5*, *e6*, *r4*, *r5*, *d4*, *d5*, *Server*}. Using the two morphisms we can get the following three sets: $\mathcal{R}_{keep} = \{e3, r2, d1, d3, Client\}$, $\mathcal{R}_{remove} = \{r1\}$, $\mathcal{R}_{duplicated} = \{e1, e2, d2, Server\}$, $\mathcal{R}^{ad}_{duplicated} = \{e4, e5, d4, :Server\}$, $\mathcal{R}_{add} = \{e6, r4, r5, d5\}$. Therefore, the result of the weaving is $result = \{e1, e2, e3, e6, r2, r4, r5, d1, d2, d3, d5, Client, Server\}$.

IV. EVALUATION

In this paper, we choice an online project named *software trust-worthy cooperation application* (Trustie) [15] as our test bed. We deployed the projected in our local environment. Trustie is a three tier system, including a portal system deployment on Ngnix, Trustie system deployment on Tomcat, and several database servers and mail servers.

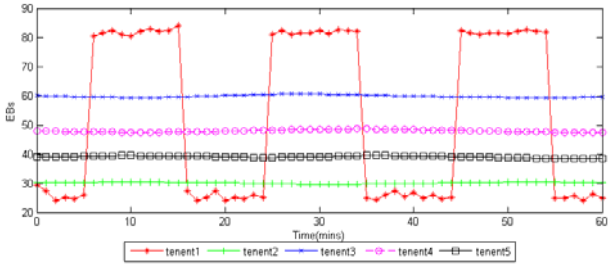

Figure 10. The workload for different tenants

There are more than 200 types of request in this system. We can divide the requests into two categories: browsing (stateless component) and processing (stateful component). The browsing type of requests require little computing resources for they only read the page from cache, and processing require plenty of resources because they need generate the rich client pages supported by Eclipse RAP[16].

Workload is critical to the SLA [17]. In order to evaluate our work, we use HP LoadRunner to generate 5 different type of workload to simulate 5 different types of tenants as show in figure 10. *Tenant 1* is a type of heavy user, most of whose requests are processing request, and other tenants are gently users, most of whose requests are browsing request. And the workload of *tenant 1* is changed cyclically.

Figure 11 give the analyzed results of our approach. The utilization of the total CPU is measured by monitor, and the tenants' utilization is analyzed by our approach. It is clearly that while the load of *tenant 1* is high, the most of the resources is occupied by *tenant 1*, and other tenants have little change to response the requests. That is because while the load of tenant 1 is high, the total CPU utilization is nearly 100%, and the job of *tenant 1* is longer than others, so the CPU will be occupied by *tenant 1*.

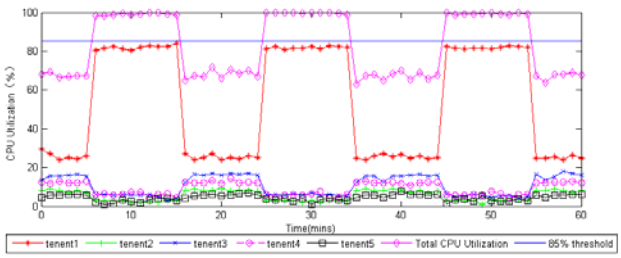

Figure 11. The resources consumption for different tenants

Past experience [18] show that the CPU utilization should controlled fewer than 85% to let the system run smoothly. Therefore, to make the resources allocation fairly amount tenants, the heavy tenant's consumption need to be controlled. In our case, it means we should reduce the concurrency users of *tenant 1* to give other tenants more change to run.

In order to find out the optimal concurrency of *tenant 1*, we should first build the holistic model, and then set the concurrency level for each tenant, finally we can find out the best configuration by search the best configurations.

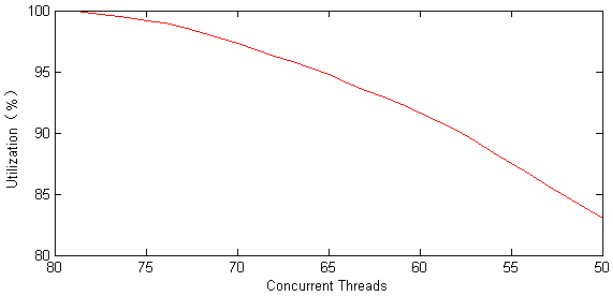

Figure 12. Searching for the optimal concurrency level

Figure 12 gives the total CPU utilization for different configuration of *tenant 1*. It can be seen that while the concurrency user is 52, the total CPU utilization is about 85%. Thus, we can set the concurrency level of *tenant 1* as 50 for the analyzed results is little more optimize than real one.

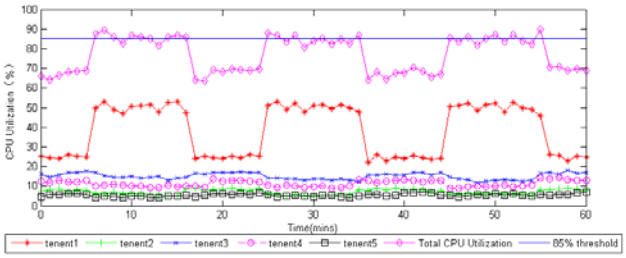

Figure 13. The CPU utilizations under concurrency control

Figure 13 gives the analyzed results under concurrency control. It is clearly that the utilization of *tenant 1* is controlled, thus the total CPU utilization is around 85%, and other tenants have more change to run. That means our approach can predict the performance of multi-tenancy application under different configurations, and give valuable advices for performance optimization.

V. RELATED WORK

Traditional performance modeling approaches usually focus on applications. In [19], Becker and Dencker present an approach that automatically generates performance prototypes based solely on a design model with performance annotations. The main contribution of their work is building the mapping of instances of the Palladio Component Model (PCM) to prototype implementations, but it does not analyze the performance impacts of the underline platform, such as concurrency policy, hence it is not enough to predict the performance of connectors.

Woodside proposed an automated construction of LQN models from the traces collected by the code added by a

tool integrated in the design environment [20]. In their approaches, one needs to insert additional code into the source code based on the design abstraction to collect execution traces and resource cost from special test cases. However, the approaches do not consider the impacts of the middleware also.

There are some approaches emphasizing the influence on the middleware on the performance of a component-based system. Llado [21] created an EQN based model to describe the performance of an EJB server. Cecchet [22] provided a benchmark for Java Enterprise application servers and found that the most influencing factor on the performance was caused by the middleware. Denaro [23] generated a prototype for a component-based application of an EJB system, but did not provide model building or tool support. Chen [24] gives a simple method to determine optimal thread pool sizes for Java application servers. However, these methods do not care about how to build performance model automatically and change with the environment.

Verdickt [25] introduces a model transformation framework to include the performance overhead of middleware layers into a component-based system model. They use UML model, such as UML activity, deployment and collaboration diagrams, to build the performance models. Following MDA approaches, a model transformation maps a platform independent model into a platform specific model with a repository of middleware models which are also built with UML models. In their case study, the authors demonstrate how to include the performance impacts of the CORBA Object Request Broker (ORB) into the models. The main drawback of this approach is that it depends on the designers' experience of performance modeling, because the application model should be provided by designers and how the middleware affects the application model should also be given by the designer.

Grassi [26] provides a connector refinement transformation method which can transform the model from high level UML 2.0 architecture models into the KLAPER modeling language. KLAPER models can be transformed into queuing networks or Markov chains for performance analysis. This approach assumes the UML stereotype can be labeled with component connectors. For example, a connector could separate a static or dynamic synchronous client/server interaction. The authors suggest building a library of parametric connector behaviors' models for each of the stereotypes.

VI. Conclusion

Multi-tenancy application is one of the most important techniques to share the platform. The performance of a multi-tenancy application is largely affected by the platform's configurations. In this paper, we proposed a configuration driven modeling approach for this type of applications, which not only presents a feasible solution for modeling configuration aware systems, but also gains us valuable experiences of modeling the complex platforms.

Acknowledgment

This work is partially supported by National Natural Science Foundation of China (61272013).

References

[1] Jiehui Ju, Jiyi Wu, Jianqing Fu, Zhijie Lin, Jianlin Zhang. A Survey on Cloud Storage. *Journal of Computers*, Vol.6, NO.8, 2011.
[2] Amazon EC2, URL: https://aws.amazon.com/ec2/
[3] Google App Engine, URL: https://appengine.google.com/
[4] Intalio, URL: http://www.intalio.com/
[5] Salesforce, URL: http://www.salesforce.com/
[6] C. J. Guo, W. Sun, Y. Huang, Z. H. Wang and B. Gao. A Framework for Native Multi-Tenancy Application Development and Management. CEC/EEE, *IEEE Computer Society*, pp. 551-558, 2007
[7] Chin-Ling Chen, Chia-Chun Yu. Performance Evaluation of Active Queue Management Using A Hybrid Approach. *Journal of Computers*, Vol.7, NO.5, 2012.
[8] M. Woodside, J. E. Neilson, D. C. Petriu, S. Majumdar. The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-like Distributed Software". *In: IEEE Transactions on Computers*, vol.44, no.1, pp.20-34, 1995.
[9] Rolia, J. A. , Sevcik, K. C.. The Method of Layers". *IEEE Trans. On Software Engineering*, vol.21, no.8, pp.689-700, 1995.
[10] Kung-Kiu Lau and Zheng Wang. Software Component Models. *IEEE Transactions on Software Engineering*, Vol.33, No.10, 2007.
[11] Ivica Crnkovic, Brahim Hnich, Torsten Jonsson. Specification, Implementation, and Deployment of Components. *Communication of the ACM*, Vol.45, No.10, pp.35-40, 2002.
[12] Wei Wang, Xiang Huang, Xiulei Qin, Wenbo Zhang. Application-Level CPU Consumption Estimation: Towards Performance Isolation of Multi-tenancy Web Applications. *IEEE Fifth International Conference on Cloud Computing*, pp.439-446, 2012.
[13] Wei Wang, Xiang Huang, Yunkui Song. A Statistical Approach for Estimating CPU Consumption in Shared Java Middleware Server. *IEEE 35th Annual Computer Software and Applications Conference*, 2011 Munich, Germany, pp.541-546, 2011.
[14] Xiang Huang, Wei Wang. An Adaptive Performance Modeling Approach to Performance Profiling of Multi-Service Web Applications. *In proceeding of Annual Computer Software and Applications Conference*, pp.4-13, 2011.
[15] Trustie forge, URL: http://www.trustie.net/trustie/forge?language=en
[16] Eclipse RAP, URL: http://www.eclipse.org/rap/
[17] Congfeng Jiang, Jian Wan, Xindong You. Power Aware Job Scheduling in Multi-Processor System with Service Level Agreements Constraints. *Journal of Computers*, Vol. 5, No. 8, 2010.
[18] Steffen Becker. Coupled model transformations for QoS enabled component-based software design, Ph. D. *Thesis, University of Oldenburg*, Germany, January 2008.
[19] M. Woodside, C. Hrischuk, B. Selic, and S. Brayarov. Automated Performance Modeling of Software Generated by a Design Environment, *Performance Evaluation*, vol. 45, pp.107-123,2001.
[20] Curtis E. Hrischuk, Murray Woodside, Jerome A. RoliaJerome, A. Rolia. Trace-Based Load Characterization

for Generating Performance Software Models. *IEEE Transactions on software engineering*, vol. 25, no. 1, 1999.

[21] Catalina M. Llado, Peter G. Harrison. Performance evaluation of an enterprise JavaBean server implementation, *in: Proc. 2nd Int. Workshop on Software and Performance*, WOSP'00, ACM, New York, NY, USA, pp. 180-188, 2000.

[22] Emmanuel Cecchet, Julie Marguerite, Willy Zwaenepoel. Performance and scalability of ejb applications, *in: Proc. 17th ACM SIGPLAN Conf. on Object-oriented programming, systems, languages, and applications*, OOPSLA'02, ACM, New York, NY, USA, pp. 246-261, 2002.

[23] Giovanni Denaro, Andrea Polini, Wolfgang Emmerich. Early performance testing of distributed software applications, *in: Proc. 4th Int.Workshop on Software and Performance*, WOSP'04, ACM, New York, NY, USA, pp. 94-103, 2004.

[24] Shiping Chen, Yan Liu, Ian Gorton, Anna Liu. Performance prediction of component-based applications, *Journal of Systems and Software*, Vol.4, No.1, pp.35-43, 2005.

[25] Tom Verdickt, Bart Dhoedt, Frank Gielen, Piet Demeester, \Automatic inclusion of middleware performance attributes into architectural uml software models", *IEEE Transactions on Software Engineering*, Vol.31, No.8, pp. 695-771, 2005.

[26] A model transformation approach for the early performance and reliability analysis of component-based systems, *in: Proc. 9th Int. Symposium on Component-Based Software Engineering*, CB-SE'06, in: LNCS, vol. 4063, Springer, pp. 270-284, 2006.

Xiang Huang received the M.Sc. degree from the South China University of Technology, Guangzhou, China, in 2007 and Ph.D. degree in computer science and technology from Chinese academy of sciences, Beijing, China, in 2012.
His main research interests include distributed systems, system performance analysis, theory of computation etc.

Zhi-gang Chen is a senior engineer. His main research interests include system planning, theory of computation etc.