This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2586067, IEEE Transactions on Cloud Computing

Statistical Model Checking-Based Evaluation and Optimization for Cloud Workflow Resource Allocation

Mingsong Chen, *Member, IEEE*, Saijie Huang, Xin Fu, *Member, IEEE*, Xiao Liu, *Member, IEEE*, and Jifeng He

Abstract—Due to the existence of resource variations, it is very challenging for Cloud workflow resource allocation strategies to guarantee a reliable Quality of Service (QoS). Although dozens of resource allocation heuristics have been developed to improve the QoS of Cloud workflow, it is hard to predict their performance under variations because of the lack of accurate modeling and evaluation methods. So far, there is no comprehensive approach that can quantitatively reason the capability of resource allocation strategies or enable the tuning of parameters to optimize resource allocation solutions under variations. To address the above problems, this paper proposes a novel framework that can evaluate and optimize resource allocation strategies effectively and quantitatively. By using the statistical model checker UPPAAL-SMC and supervised learning approaches, our framework can: i) conduct complex QoS queries on resource allocation instances considering resource variations; ii) make quantitative and qualitative comparisons among resource allocation strategies; iii) enable the tuning of parameters to improve the overall QoS; and iv) support the quick optimization of overall workflow QoS under customer requirements and resource variations. The experimental results demonstrate that our automated framework can support both the Service Level Agreement (SLA) negotiation and workflow resource allocation optimization efficiently.

Index Terms—Cloud Computing, Statistical Model Checking, Optimization, Resource Allocation Strategy, Service Level Agreement.

1 Introduction

By adopting the Software as a Service (SaaS) model, Cloud workflow systems are becoming the mainstream platform to facilitate the automated distribution of data and computation-intensive enterprise applications [1]. Figure 1 shows how customer requests are served in different layers of a Cloud workflow system. Initially, a customer sends a workflow request to an SaaS provider with specified QoS requirements. Such requirements usually contain the information of expected price, response time, reliability (bearable failure ratio), etc. Since different underlying Virtual Machines (VM) (e.g., Amazon EC2, Microsoft Windows Azure) offered by different Platform as a Service (PaaS) and Infrastructure as a Service (IaaS) providers have different capacity (e.g., CPU, memory, storage) and on-demand prices, SaaS providers need to conduct the evaluation of multiple resource allocation solutions and figure out a suitable one, which should be profitable and can meet all the customer requirements. If the requirements cannot be satisfied by the evaluation, before signing Service Level Agreement (SLA), the SaaS provider will negotiate the price and QoS details with the customer.

To achieve increasing profit in the fierce Cloud computing market, SaaS providers need to explore efficient resource allocation solutions [2] that can minimize the cost of infrastructure services without adversely affecting Quality of Services (QoS) [3]. Since resource allocation for Cloud workflows is an NP-complete problem, various strategies have been proposed to quickly find a solution (i.e., a resource allocation instance) [4]. However, due to the underlying resource variations (e.g., cost, execution time), it is hard to determine which resource allocation strategy works best for a given workflow coupled with QoS requirements. Therefore, quantitative evaluation of resource allocation strategies is becoming an important issue to guarantee the QoS in Cloud computing. In order to satisfy customer requirements as well as achieve an acceptable profit for SaaS providers, resource allocation strategy evaluation must address the following issues: i) How to accurately model variation-aware workflow services and customer requirements to enable the quantitative resource allocation evaluation? ii) How to quickly conduct negotiation properly with customers if the given requirements are not satisfied? iii) How to quickly find an optimized resource allocation solution that can further promote the customer satisfaction without affecting the customer requirements? Although simplified probability-based approaches can be used to model execution variation, few of them can accurately model the parallel service execution. Moreover, existing constraint solving-based approaches can only answer *yes* or *no* for a given workflow and user requirements. Few of them can be used to quantitatively reason why the QoS cannot be guaranteed and answer how to improve the QoS. Clearly, the bottleneck is the lack

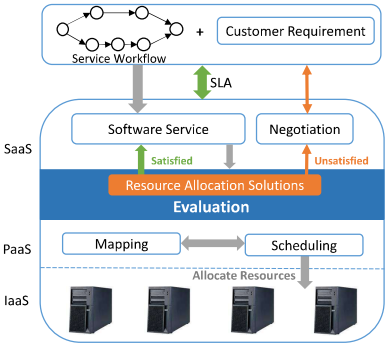


Fig. 1. Resource allocation for Cloud workflow

- *Mingsong Chen, Saijie Huang and Jifeng He are with the Shanghai Key Lab of Trustworthy Computing at East China Normal University, Shanghai, 200062, China (email: {mschen, sjhuang, jifeng}@sei.ecnu.edu.cn). Xin Fu is with the Electrical and Computer Engineering Department at University of Houston, Texas, 77004, USA (email: xfu8@central.uh.edu). Xiao Liu is with the School of Information Technology, Deakin University, Melbourne, Australia (email: xiao.liu@deakin.edu.au). Xiao Liu is the corresponding author.*

Manuscript received XXXX XX, 20XX; revised YYYY YY, 20YY.

of powerful evaluation approaches which can help SaaS providers to make choices among different resource allocation strategies.

To address this problem, we propose a novel framework using Statistical Model Checking (SMC) [6] and supervised learning approaches [7] that can systematically evaluate and optimize resource allocation strategies. SMC is a technique that relies on the monitoring of random simulation runs of systems. The simulation results are analyzed using statistical methods (i.e., sequential hypothesis testing or Monte Carlo simulation) to estimate the satisfaction probability of a specified property. Compared with formal model checking approaches, SMC requires far less memory and time, which allows high scalable validation approximation. Moreover, some interesting quantitative performance properties which cannot be expressed in traditional model checking can be analyzed in SMC. To achieve an approximate-optimal Resource Allocation Instance (RAI) under resource variations, it is required to make the comparison among a large set of RAIs. However, since each RAI evaluation using SMC is time-consuming (typically needs several minutes), the overall optimization time can be extremely large. Supervised learning [7] is a promising approach to accurately predict the outputs for the given inputs with training only a small subset of the labeled inputs. Supervised learning approaches are quite suitable in finding an approximate optimal RAI in our approach, since only a small set of RAIs are evaluated using SMC. Therefore, the overall RAI optimization time can be reduced. Our approach makes three major contributions as follows.

- We propose a novel framework that can evaluate resource allocation strategies by automatically converting their solutions with variation information into a network of priced timed automata and conducting quantitative analysis against specified performance queries.
- We present an automated negotiation mechanism that facilitates the bargain between SaaS providers and customers.
- Based on supervised learning techniques, we propose an efficient method that can quickly obtain an approximate optimal result from a large set of feasible resource allocation solutions that can maximize the QoS under the same customer requirements.

The remainder of the paper is organized as follows. Section 2 presents related works on QoS-oriented resource allocation strategies in Cloud. After the introduction of the preliminary knowledge of UPPAAL-SMC [5] in Section 3, Section 4 describes our resource allocation strategy evaluation and optimization framework in detail. To demonstrate the efficiency of our approach, Section 5 presents various experimental results using an industrial Cloud-based application. Finally, Section 6 concludes the paper.

2 Related Work

Unlike traditional market-based resource allocation methods which are non-pricing-based, various SLA- or QoS-based profit maximization resource allocation approaches have been proposed in Cloud. Based on a novel pricing model for Clouds, Lee et al. [8] investigated the profit-driven service request scheduling for workflows. To manage the dynamic change of customers, Wu et al. [9] proposed resource allocation algorithms for SaaS providers who want to minimize infrastructure cost and SLA violations. In [3], Wu et al. proposed a service-level hierarchical scheduling strategy, which outperforms existing approaches such as genetic algorithm and etc. However, so far, existing approaches focus on the modeling and optimization rather than the quantitative evaluation for resource allocation strategies.

SLA negotiation is an important topic in Cloud, since through bilateral bargaining between customers and SaaS providers in Cloud marketplace [10] can maximize their return-on-investment. However, majority of existing SLAs are formed by service providers without providing customers with sufficient negotiation opportunities. With the emergence of SaaS broker models such as ViTLive [11], some preliminary research has been conducted to investigate SLA negotiation approaches in Cloud. In [12], Zulkernine and Martin proposed a policy-based broker framework for SLA's automated negotiation. In [13], Dastjerdi and Buyya proposed negotiation strategies for the infrastructure layer which depends heavily on provider resource capabilities. To maximize profits and improve customer satisfaction levels for brokers, Wu et al. [14] proposed an automated negotiation framework where an SaaS broker is utilized as the one-stop-shop for customers when negotiating with multiple providers. However, few of the above approaches considered the QoS with resource variations.

Machine learning-based algorithms are widely investigated in Cloud to facilitate resource allocation and management. In [15], Xiong et al. presented a cost-aware resource management system SmartLA, which uses machine learning techniques to achieve the optimum profits. In order to maximize computing performance, Huang et al. [16] adopted Genetic Algorithms to optimize the re-allocation of CPU and RAM resources in Cloud. In [17], Menache et al. proposed an online learning approach for resource allocation to balance the computation cost and performance. It is based on the learning from the performance of prior job executions while incorporating history of spot prices and workload characteristics. Although various heuristics were proposed to improve the overall performance, none of existing machine learning-based approaches considers the resource variations in Cloud workflow.

Due to the scalability and efficacy in quantitatively reasoning performance metrics of systems, SMC is widely used in the evaluation of system designs. In [20], Du et al. adopted UPPAAL-SMC to conduct quantitative evaluation on project schedules. Their approach supports various evaluation queries for the schedule comparison. Based on UPPAAL-SMC, Chen et al. [18] proposed a framework that can conduct evaluation for MPSoC task allocation and scheduling strategies under variations. In [19], Gu et al. extended the semantics of UML activity diagrams to enable the quantitative timing analysis of stochastic behaviors by using UPPAAL-SMC. In [21], by exploiting a stochastic semantics together with simulation, David et al. presented an SMC-based approach that can achieve best values for model parameters to enable design optimizations. So far, SMC-based methods are seldom used for Cloud resource allocation evaluation. Moreover, there is no approach that combines both machine learning and SMC techniques to search for optimal solutions.

Currently, there exists many simulation-based approaches that support the performance evaluation of workflows. For example, based on CloudSim [22], WorkflowSim [23] is developed to enable the full-fledged analysis of Cloud-based workflow models with failures. In [24], Rozinat et al. proposed a simulation system that supports to incorporate historical information to predict potential near-future behavior for different scenarios. However, few of them support the performance evaluation for workflows under resource variations. Although stochastic QoS parameters in service-based workflows have been investigated in [25], its proposed aggregation functions and optimization model cannot accurately model the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2586067, IEEE Transactions on Cloud Computing

concurrent workflow behaviors under resource variations in the context of Cloud. Moreover, its optimization goal is to reduce the total cost for brokers, while our approach tries to automatically find a best success ratio for the given customer requirement with a small overhead. In fact, due to the complex queries supported by UPPAAL-SMC, our approach can be extended to solve various types of optimization problems. To the best of our knowledge, our work is the first SMC-based approach that can not only evaluate and optimize resource allocation strategies for Cloud workflows under variation, but also can support the decision making in automated SLA negotiation.

3 Preliminary

As an extension of the model checker UPPAAL, UPPAAL-SMC [5] enables accurate reasoning of complex system behaviors under a stochastic semantics. In our approach, we use a Network of Priced Timed Automata (NPTA) [5], [6] to formally specify the behaviors of a Cloud workflow. An NPTA comprises a set of correlated Priced Timed Automata (PTAs) which can model the stochastic executions of workflow services. Within an NPTA, PTAs are synchronized over broadcast channels. Note that our approach employs *urgent channels* by default because we assume that there is no delay before triggering a synchronization. Since UPPAAL-SMC provides a user-friendly interface for NPTA-based analysis, it can be used to quantitatively evaluate various variations in the context of Cloud workflow resource allocation.

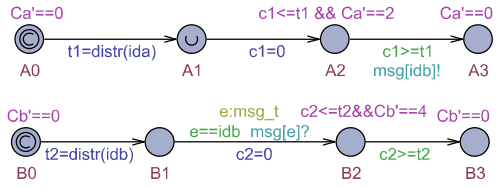


Fig. 2. An NPTA, (A | B)

To demonstrate the stochastic execution of an NPTA, Figure 2 shows an example with two PTAs A (id=ida) and B (id=idb), where each PTA has four locations, three edges and two local clocks (e.g., c_1 and C_a in A), respectively. Unlike traditional timed automata, clocks in PTAs can evolve with different rates (i.e., unit price) in different locations. By default, the rate of clocks is 1. However, NPTA allows changing the rate of a clock C by specifying a constant value to the corresponding primed clock C'. For example, the invariant C'_a==*2* of location A_2 denotes that the rate of C_a is 2 in location A_2. For the purpose of synchronization, this example adopts an array of urgent broadcast channels *msg*, where $msg[idb]$ indicates a specific channel for the synchronization between A and B. In our approach, we adopt the non-deterministic *selections* to filter mismatching synchronizations. For example, on the outgoing edge of location B_1, the *selections e:msg_t* and the *guard e==idb* are used to filter synchronizations which are not performed over channel $msg[idb]$. Assume that PTA A broadcasts over the channel using "$msg[idb]$!". Only when *e==idb* holds and there exists a complementary receiving action "$msg[e]$?" in PTA B, the synchronization can be triggered.

Since we focus on the evaluation of resource allocation instances, we need to model the stochastic behaviors of PTAs. Currently, UPPAAL-SMC only supports the uniform and exponential distributions explicitly, which cannot cover various complex scenarios in practical designs. To enable the modeling of different kinds of stochastic behaviors, we employ the pattern shown in Figure 2. Since UPPAAL-SMC supports almost the same programming constructs as in C programming language, by proper programming using the built-in function *random*(), the self-defined function *distri*() can produce values following a large set of commonly used distributions (e.g., normal distribution, Poisson distribution). For example, the *Box-Muller method* can be used to generate normally distributed random delays for PTAs. In the pattern, since location A_2 sets an upper bound for clock c_1 (i.e., $c_1 <= t_1$) and its outgoing transition sets a guard condition $c_1 >= t_1$, PTA A can stay in location A_2 with a delay t_1 (randomly generated using *distr(ida)*).

After each decision of an NPTA, the shortest delay will be executed and all continuous variables will be updated accordingly. Meanwhile, the PTA with the shortest delay will attempt to take a transition. Note that if there is a PTA process in a *commit* or *urgent* location (i.e., a location marked with the symbol "C" or "U"), the process will have a zero delay in this location, and the next transition must involve an edge from one of the *commit* or *urgent* locations (*commit* locations have higher priority).

Assume that PTAs A and B follow the Gaussian distribution $N(3,1^2)$ and $N(6,2^2)$, respectively. The following run is a possible transition sequence of the NPTA $(A|B)$.

$$((A_0,B_0),[c_1=0,c_2=0,C_a=0,C_b=0]) \xrightarrow{0}$$
$$((A_1,B_1),[c_1=0,c_2=0,C_a=0,C_b=0]) \xrightarrow{0}$$
$$((A_2,B_1),[c_1=0,c_2=0,C_a=0,C_b=0]) \xrightarrow{2.5} \xrightarrow{msg[idb]!}$$
$$((A_3,B_2),[c_1=2.5,c_2=0,C_a=5,C_b=0]) \xrightarrow{5.1}$$
$$((A_3,B_3),[c_1=7.6,c_2=5.1,C_a=5,C_b=20.4]) \dashrightarrow \ldots$$

The example demonstrates that the composite location (A_3,B_3) is reachable within 7.6 time units with a total cost 25.4. Assuming that there is no correlation between clocks c_1 and c_2, while more runs are simulated, we can find that the time of reaching composite location (A_3,B_3) will follow the Gaussian distribution $N(9,1^2+2^2)$. By using the message-based synchronization among PTAs, arbitrarily complex stochastic behavior can be modeled.

Based on the stochastic semantics, UPPAAL-SMC models enable the generation of random runs which are bounded by time, cost or a number of discrete steps. During the checking using UPPAAL-SMC, all these derived runs have to be monitored with some specified properties in the form of cost-constrained temporal logic [6]. At the end of checking, the probability range of each property coupled with a specified confidence will be reported. In our approach, we only use following three kinds of query formats supported by UPPAAL-SMC:

- **Qualitative check**: Pr [time <= bound] (<> expr) >= p.
- **Quantitative check**: Pr [time <= bound] (<> expr).
- **Probability comparison**: Pr [time1 <= bound1] (<> expr1) >= Pr [time2 <= bound2] (<> expr2).

In above query definitions, *bound* is a constant value, and properties are evaluated using random runs which are bounded by *time*. The expression <> *expr* asserts that the state predicate *expr* will happen eventually. The qualitative check can be used to check whether the probability of property <>*expr* is at least p or not. Such check can be used for SLA negotiation by SaaS brokers. The quantitative check can be used to conduct the interval estimate for the success ratio of the given property. It can be used to evaluate the performance of different resource allocation strategies. The property comparison can be used to filter inferior strategies.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2586067, IEEE Transactions on Cloud Computing

4 Our Approach

In this section, we formulate the variation-aware resource allocation problem for Cloud workflow (Section 4.1). We propose a novel framework (Section 4.2) based on UPPAAL-SMC which can conduct modeling and evaluation of resource allocation strategies (Section 4.3) and enable automated SLA negotiations (Section 4.4) and resource allocation optimization (Section 4.5).

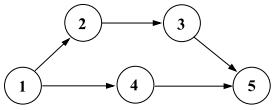
4.1 Problem Definition

To satisfy different types of customers, PaaS or IaaS providers usually offer multiple VM plans with different performance and prices. If the budget of a customer is insufficient, providers may choose VMs with lower configuration during resource allocation. Since cheaper VMs generally have worse performance and a larger variation in service response time, it is difficult to guarantee that the customer's request can be fulfilled on time. Consequently, SLA violations are inevitable. Therefore, VM related variations should be taken into account when making resource allocation decisions. Such information can be queried from PaaS or IaaS providers, or can be obtained from the service history of SaaS providers.

During the resource allocation, our approach only considers three factors: the unit price, time variation and QoS (i.e., the success ratio of completing service workflow on time). Since unit price can be considered as a special resource, our approach can be easily extended to solve the problems in other domains. To simplify the modeling of resource allocation problem, our approaches assumes that the profit ratio required by SaaS providers is a constant (e.g., 20%). In other words, regardless of profit, the cost information in our model is the maximum budget of Cloud service providers that can spend on VMs. In practice, proper optimization for some resource allocation strategy can further save part of this cost as the margin profit. Since directed acyclic graph (DAG) is widely used in describing Cloud workflows [27], [30], the problem studied in this work is formulated as follows. Given

- A DAG $G = (V,E)$ indicating a service workflow, where each node in $V = \{v_1, \ldots, v_n\}$ represents a service. E is a set of directed arcs which represent dependence relations between services. Virtual machine set $VM = \{vm_1, \ldots, vm_k\}$ denotes all the available VMs that can serve at least one element in V.
- A unit runtime cost function $U : V \times VM \to \mathbb{R}$, where $U(v_i, vm_j)$ represents the unit runtime cost of the i_{th} service running on the j_{th} VM. If $U(v_i, vm_j) = \infty$, it means that the i_{th} service cannot be assigned to the j_{th} VM.
- An execution time function $T_r : V \times VM \to \mathbb{R}$, where $T_r(v_i, vm_j)$ indicates real execution time of service v_i running on VM vm_j in a random simulation run.
- An end time function $ET : V \times VM \to \mathbb{R}$, where $ET(v_i, vm_j)$ indicates the real end time of service v_i running on VM vm_j assuming that the workflow starts from time 0.
- An execution time variation function $VAR : V \times VM \to DIST$, where $VAR(v_i, vm_j)$ indicates the execution time distribution of service v_i running on VM vm_j, such that $T_r(v_i, vm_j)$ follows the distribution $VAR(v_i, vm_j)$.
- A resource allocation function $AL : V \to VM$, where $AL(v_i) = vm_j$ indicates that services v_i is assigned to VM vm_j. A resource allocation instance is a binary relation $S = \{(v_1, AL(v_1)), \ldots, (v_n, AL(v_n))\}$, which corresponds to a specific resource allocation function.
- The customer defined constraint $R(C,T,SR)$, where **C** indicates the maximum cost for the resource allocation; T is the required deadline of the workflow and SR is the success ratio indicating the required percentage of the successful workflow execution before the deadline.

To achieve a feasible resource allocation instance RAI, it is required that the success ratio of workflow executions which meet the requirement $\Sigma_{i=1}^{n}(U(v_i, RAI(v_i)) \times Tr(v_i, RAI(v_i))) \leq C$ and $Max_{i=1}^{n} ET(v_i, RAI(v_i)) \leq T$ is equal to or larger than SR. An optimal RAI is a feasible RAI that has the highest success ratio.



(a) A service workflow example

	Config. 1			Config. 2		
Node	U.C.	E.T. (μ)	S.D. (σ)	U.C.	E.T. (μ)	S.D. (σ)
1	6.0	5.0	0.6	5.0	6.0	0.8
2	7.0	5.0	1.2	5.0	6.0	1.4
3	4.5	10.0	0.8	6.0	8.0	0.6
4	4.0	10.0	0.7	3.0	12.0	0.9
5	5.0	7.0	1.5	3.0	12.0	1.8

(b) VM configuration table of (a)

Fig. 3. A workflow example and its configurations

Figure 3a) shows a workflow example with five services. In this figure, due to the service dependence relation, services 2 and 4 cannot be started until service 1 finishes, and service 5 cannot be executed until both services 3 and 4 finish. Figure 3b) shows the VM configuration details of the services. In the configuration table, each service node is assigned with two configurations, which indicates that each service can be assigned to either of the two VMs. The *U.C.* column indicates the unit cost of VMs for the specified services. The columns *M.T.* and *S.D.* denote the expected execution time and corresponding standard deviation for each service, respectively. Assume that a customer wants to check whether there exists an RAI that satisfies the requirement as follows: *"the workflow should be executed with a cost less than 185 and a time less than 45, and the execution failure ratio should be less than 5%"*. Without considering VM variations, the resource allocation instances $RAI_1 = \{(V_1, VM_{1,1}), (V_2, VM_{2,2}), (V_3, VM_{3,1}), (V_4, VM_{4,1}), (V_5, VM_{5,1})\}$ and $RAI_2 = \{(V_1, VM_{1,2}), (V_2, VM_{2,1}), (V_3, VM_{3,1}), (V_4, VM_{4,1}), (V_5, VM_{5,1})\}$ can both satisfy the above requirement, where $VM_{i,j}$ indicates that V_i is assigned with its j_{th} VM configuration. Based on the expected execution time, we can get that RAI_1 needs a cost of 180, and RAI_2 needs a cost of 185. It seems that RAI_1 has more margin profit than RAI_2. However, if the time variation shown in the table is considered, the situation becomes more complex, since large accumulated variations may result in low satisfying level for customers. Although various approaches [26] are proposed to prevent temporal violations in Cloud workflow, few of them can guarantee the best performance in real VM execution environment with resource variations. Moreover, even if multiple resource allocation instances can fulfill the customers' requirements, due to the lack of evaluation tools, it is hard to select the best one from the RAI candidates.

4.2 Our Framework

Our approach adopts a two-phase evaluation process to conduct the comparison and optimization of resource allocation solutions. In the first phase, the SaaS providers need to quickly respond to the customer requirements with proper bargain capabilities. The

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2586067, IEEE Transactions on Cloud Computing

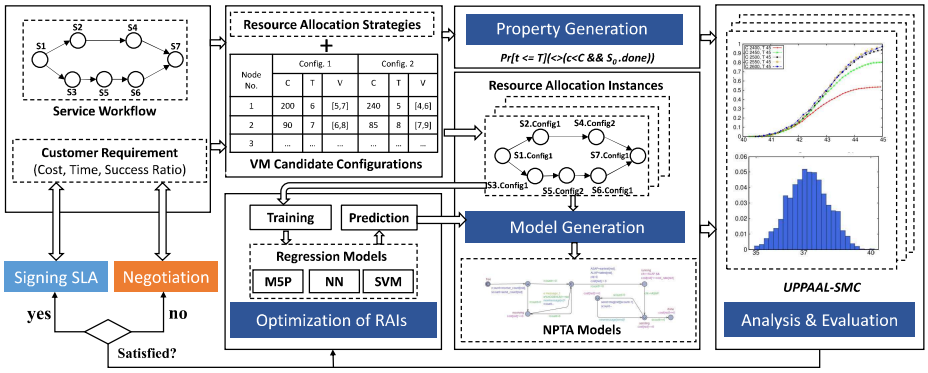


Fig. 4. Our resource allocation evaluation and optimization framework

second phase mainly handles the resource allocation optimization. Since the optimization needs to explore all the possible solution space, it takes much longer time than the time of first phase and usually the optimization is conducted in the background.

Figure 4 presents our UPPAAL-SMC-based framework. In the first phase, based on the Cloud service workflow and customer requirements extracted from some SLA contract, the SaaS provider needs to search for candidate VMs that can carry out all the services in the given workflow. Meanwhile, the unit cost, expected execution time and variation information will be queried from Paas or IaaS providers. Since no existing approaches can always generate best resource allocation instances under time and cost variations, to obtain a relative optimal RAI in a reasonable time, our evaluation framework supports the comparison between different resource allocation strategies. It is important to note that our framework does not provide any suggestions on which resource allocation strategies should to be selected. They are selected by the SaaS provider based on different purposes (e.g., time optimal, cost optimal). By using our defined mapping rules, the generated RAIs and customer requirements can be automatically converted into NPTA models and property-based queries, respectively. Based on UPPAAL-SMC, our framework can automatically analyze and evaluate the generated RAIs. If the customer requirement can be satisfied, an RAI with the best performance will be reported, and the SaaS provider will sign an SLA with the customer. Otherwise, the SaaS provider needs to negotiate with the customer based on the evaluation results. By increasing the cost properly and reevaluate the newly generated instances, the customer and the SaaS provider will finally reach a reasonable SLA after several iterations or abort the negotiation. Achieving an RAI under resource variations with optimal QoS usually requires heavy evaluation efforts, since it requires a large set of feasible RAIs to be checked by UPPAAL-SMC and each evaluation of an RAI using UPPAAL-SMC is quite time-consuming. Therefore, in the optimization phase, we employ the supervised learning to quickly locate an approximate optimal solution. Note that our approach performs resource allocation optimization with an offline mode. Once an optimal solution is obtained, the current adopted RAI will be replaced by the optimal one. Since both phases can be fully automated, both the evaluation and optimization processes can be conducted without human intervention. The following subsections will introduce the major steps of our framework in detail.

4.2.1 NPTA Model Generation

Before performance analysis, an RAI needs to be translated into an executable NPTA model first. To simplify the NPTA model construction, our approach decouples the syntax and semantics of a workflow with allocated resources. We divide the NPTA model for a Cloud workflow into two parts: front-end model and back-end configuration. All the workflows share the same front-end model. The only difference between workflows is the back-end configuration which describes both the concurrent behavior of workflows and the resource variation information.

In the DAG of a workflow, the dependence between services is indicated by edges. It is required that a service can be executed if all its precedent services have been finished. Therefore, the front-end model only needs to model the behavior of a workflow service rather than the whole workflow structure. This is because that all the services in the workflow share the same behavior template. In our approach, the back-end configuration is used to describe the structure and behavior (i.e., DAG structure and dependence relations between services) of a workflow and the resource variation details of the corresponding RAI.

Assume that there are N services in a workflow and the distribution of services follows the normal distribution. To identify each node in the workflow DAG and specify its execution time, the back-end configuration assigns each node with an ID together with an execution time distribution which specifies the expected execution time and its standard deviation for the corresponding service. The distribution is described using one two-dimensional array *distribution[N+1][2]*, where *distribution[i][0]* represents the expected execution time and distribution[i][1] indicates the standard deviation of execution time for the i_{th} allocated service. Moreover, each node is associated with a clock *cost* to record the execution cost of the service. The clock rate saved in the array *clock_rate[N+1]* indicates the unit cost information of services.

In service workflow, edge information is used to describe the dependence relations between services. According to the workflow semantics, in the back-end configuration, we need to figure out all the predecessors and successors for each node. Since our approach only cares about the service dependence to model the concurrent behaviors of a workflow, for each service we need to record how many predecessors have been finished and how many successors that need to be notified. Therefore, we define two arrays *receive_count[N+1]* and *send_count[N+1]* to indicate how many predecessors have completed their services, and how many successors need to be notified after the completion of the current service. A service without any predecessors is called *initial service*, and a service without any successors is called *final service*. Since a workflow may have multiple final services, to ease the property generation (see Section 4.2.2), we added a dummy service with ID 0 to merge all the final workflow services.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2586067, IEEE Transactions on Cloud Computing

To update the receive count *rcount* and send count *scount* of a service, we adopt the broadcast synchronization to mimic the end-to-end communication between services. In our approach, each edge in DFGs can be considered as a private channel working only for two adjacent services. We encode the message that is sent from the service with id_x to the service with id_y as follows:

$$encode\_msg(id_x, id_y) = id_x \times (N+1) + id_y.$$

This encoding consists of the ID information of both senders and receivers. By using this encoding, when a listener service (i.e., service waiting for the notifications from predecessor services) receives a broadcast message m, it will decode the message using $m\%(N+1)$ to check whether this message is sent to itself or not. If yes, the service will decrease its *rcount* by 1.

When a service finishes, it will send notifications to all its successors. After sending a message, the service will decrease its *scount* by 1. In the back-end configuration, we use a two-dimensional matrix *msg* to keep all the messages of the workflow, which implicitly represent the workflow edges. Instead of constructing a $(N+1)\times(N+1)$ size matrix, we use a matrix with size $(N+1)\times MAX(deg(v_1),\ldots,deg(v_n))$, where $deg(v_i)$ indicates the number of output edges incident to the service node. Similar to the data structure *adjacent table*, the value in $msg[i][j]$ indicates that the message is sent from the i_{th} service. If $msg[i][j] == -1$, it means the message has no destination. Otherwise the message will be sent to the $(msg[i][j]\%(N+1))_{th}$ service. The following is an example of the message matrix for the workflow shown in Figure 3(a): $msg[6][2]$ = {{-1, -1}, {8, 10}, {15,-1}, {23,-1}, {29,-1}, {30,-1}}. After the completion, the i_{th} service will check its *scount* information and find the corresponding message entries $msg[i][]$ to notify its successors via broadcasting.

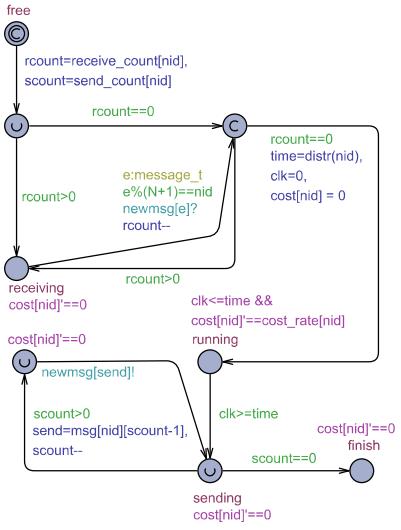


Fig. 5. Front-end model template of a workflow service

In our front-end model design, we utilize the end-to-end communication to model the dependence between services. Only when the service collects all the messages sent by its predecessors, the current service can start. When the current service completes, it will notify all its successors one by one. Assume that the ID of current service is *nid*. Figure 5 shows the template of our front-end model. The model has five major states:

1) **Free state** indicates the beginning of a service. It initializes the dependence information of the service. Since it requires no time, we set it as a commit state.
2) **Receiving state** is used to listen all the broadcast messages and tries to obtain all the notification messages from predecessor services.
3) **Running state** means that all the predecessor services are finished, and the current service is executing. In this state, the cost of the service (i.e., $cost[nid]$) will be calculated using the unit cost $cost\_rate[nid]$. The execution time generated by *distr* follows the specified distribution.
4) **Sending state** tries to notify all the successive services about the completion of the current service. Since the sending process is conducted instantly, the sending state is set to be an urgent state.
5) **Finish state** indicates the completion of a service.

It is important to note that, to record the cost of each service, only the running state has a cost rate greater than 0. In all the other states, the cost of the service will always be 0. Based on the front-end model in Figure 5 and back-end configurations, the derived NPTA model of an RAI can exactly mimic the workflow behaviors. Therefore, we can conduct the quantitative performance query on it using the statistics-based approaches.

4.2.2 Property Generation

When NPTA models are generated from different RAIs, we need to compare the QoS between them. Based on customer requirements, we can generate various properties to conduct performance queries using the model checker UPPAAL-SMC. Since our approach focuses on the QoS evaluation and SLA negotiation, customers would like to figure out *"what is the probability that the workflow can be completed using a time of x with a cost of y?"*. Since this is a safety property-based query about the QoS, our approach adopts the property in the form of $A <> p$ to check whether the customer requirement described as p can be fulfilled eventually. In UPPAAL-SMC, we analyze the above requirement using the following property

$$Pr[<=x](<> (cost[1]+\ldots+cost[N]) <= y \;\&\&\; S_0.done),$$

where $S_0.done$ indicates the completion of the whole workflow, and $cost[1]+\ldots+cost[N]$ represents the overall cost of the workflow execution. In UPPAAL-SMC, property works as a monitor to check whether a run of a given time length satisfy the property p. When the check finishes, the probability distribution of successful simulations will be reported to enable quantitative analysis.

4.3 Resource Allocation Strategy Evaluation

Since cost, response time, and success ratio in customer requirements are often conflicting with each other, it is difficult to guarantee the optimality for all these three aspects simultaneously, especially under the circumstance of service execution time variations. Given a specific requirement, SaaS providers and customers may prefer to use different resource allocation strategies, since they do have different demands. For SaaS providers, if the price and success ratio are satisfied, lower cost spent on IT infrastructures (e.g., VMs) will lead to larger margin profit. For customers, if they care more about real-time services, under the same cost and success ratio constraints, they will prefer shorter response time.

Due to the lack of existing approaches that can always find a resource allocation instance with best performance with respect to the cost and response time, it is practical for SaaS providers to conduct the comparison among multiple resource allocation strategies. Therefore, for a given set of resource allocation strategies, the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2586067, IEEE Transactions on Cloud Computing

purposes of resource allocation strategy evaluation is to filter the inferior ones and select an instance with the best performance.

To support SaaS providers' decision making, our framework provides three kinds of resource allocation strategies by default: 1) time-constrained cost minimization (TCCM), 2) cost-constrained time minimization (CCTM), and 3) x_{th}-round feasible instance (xRFI). All these strategies produce RAIs using the expected execution time without considering variations. Here, TCCM refers to the strategy that searches for a cost optimal RAI while the time constraint is satisfied. CCTM is a strategy that searches for a time optimal RAI while the cost constraint is not violated. Since TCCM and CCTM search for optimal solutions, they need to enumerate all the feasible RAIs. Though a give resource allocation problem may have multiple feasible instances, a typical exhaustive search will terminate when finding the first feasible solution. To investigate more feasible schedules, we adopt the xRFI approach which returns the x_{th} feasible RAI encountered in the exhaustive search. It is important to note that, due to the independence between RAI generation and evaluation, other resource allocation strategies can be easily integrated into our framework.

Based on our proposed framework, we can conduct the automated analysis using different evaluation strategies as follows.

1) Single Requirement Multiple Strategies (SRMS): For a specified requirement, it evaluates different RAIs generated by different strategies respectively. SRMS can be used to select the best instance for the requirement.
2) Multiple Requirements Single Strategy (MRSS): It tunes the parameters of customers' requirements, and evaluates the RAIs generated from different tuned requirements using the same strategy. MRSS can be used to figure out proper requirement parameter values for the RAI.
3) Multiple Requirements Multiple Strategies (MRMS): It tunes the parameters of the customer's requirements, and generates one RAI for each combination of the strategies and requirements. MRMS can be used to compare the performance of different resource allocation strategies.

4.4 Service Level Agreement Negotiation

When all the generated RAIs cannot satisfy the customers' initial requirements by using our approach, SaaS providers need to negotiate with the customers to relax the requirement constraints using proper negotiation protocols. Negotiation protocol refers to a set of rules, steps or sequences during the negotiation process, aiming at SLA establishment. Although there are many means of negotiation such as bilateral, one-to-many, and many-to-many, our approach only considers the bilateral negotiation between one customer and one SaaS provider. We do not consider the competitions among multiple SaaS providers. Since QoS such as time and success ratio are two major concerns of customers, this paper only focuses on how to achieve a reasonable cost that satisfies the customer requirement with the fixed response time and success ratio constraints.

Let $conf_1 = (C, \mu, \sigma)$ be the current VM configuration of a service, where C indicates the unit cost; μ represents the expected value and σ denotes the standard deviation of the service execution time. Let $conf_2 = (C', \mu', \sigma')$ be a better configuration of an available VM with with higher cost, i.e., $C' > C$. We define the *cost-benefit* factor of a service that can be achieved due to the upgrade of service configuration as follows:

$$cost\_benefit(conf_1, conf_2) = \frac{(\mu - \mu') \times \sigma}{(C' - C) \times \sigma'}$$

To enable automated negotiation, our SaaS negotiation decision making system adopts the following strategy. If the current RAI cannot satisfy the customer's requirement, SaaS providers will increase the price by a specified percentage (e.g., 5%) or amount. Under the new price constraint, our framework will try to figure out a new RAI, which tries to upgrade all the services with the highest cost-benefit value (see the example in Section 5.3). The process will be iterated until: i) a desired instance is achieved with a reasonable price that the customer can afford; or ii) the price exceeds the customer's expectation. It is important to note that our evaluation framework enables SLA negotiation based on the RAI comparison. Therefore, other SLA negotiation heuristics and protocols can be easily integrated into our framework to improve the negotiation process.

4.5 Optimization of Resource Allocation Instances

The objective of SLA negotiation presented in Section 4.4 is to quickly find an RAI that satisfies the customer requirement. In fact, the selected RAI may not be the one with the highest QoS. Therefore, after signing the SLA, there should be proper optimization processes to improve the QoS. It is important to note that, resource allocation strategies are only promising in quickly finding one satisfying RAI, since they will be terminated when finding the first satisfying RAI. However, under the resource variations, it is hard to guarantee that the first found RAI is optimal or approximate optimal. To find an optimal RAI under resource variations, it is required to check all the feasible RAIs that satisfy the customer requirement.

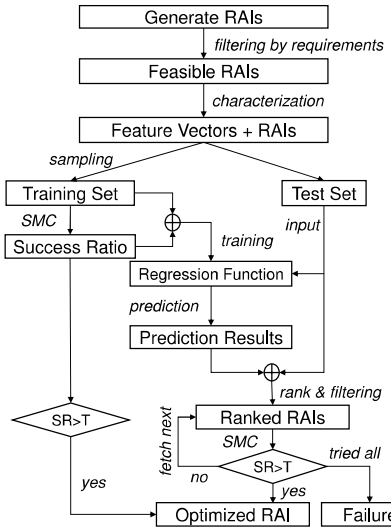


Fig. 6. The workflow of our supervised learning-based RAI optimization

Although our framework can model the system behavior more accurately than traditional methods, typically the simulation time of each RAI evaluation is long (e.g., several minutes). Therefore, evaluating all RAIs for a given Cloud workflow is infeasible in practice. To reduce the overall evaluation time for all RAIs and achieve an RAI with approximate optimal QoS, we adopt a supervise learning-based approach that is illustrated in Figure 6. Firstly, all feasible RAIs are filtered based on the customer requirements. Secondly, the VM configuration information of each RAI will be analyzed and encoded into a feature vector. Next, all the generated feature vectors coupled with corresponding RAIs will be divided into two categories: training set and test set. The

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2586067, IEEE Transactions on Cloud Computing

feature vectors in the training set will be checked using UPPAAL-SMC to achieve its success ratio (i.e., an indicator of QoS). Under the resource variations, it is hard to predict the optimal QoS of a resource allocation problem. In our framework, we introduce the threshold T, which is an optimal RAI indicator specified by SaaS providers. Instead of exhaustively finding an optimal RAI, our approach tries to find an approximate optimal RAI with much less evaluation efforts. If the success ratio reported by UPPAAL-SMC is larger than T when checking an RAI in the training set, the whole optimization can be terminated and the RAI will be reported as an approximate optimal solution. After all the target QoS values of RAIs in the training set are calculated using SMC, we can derive a regression function based on the feature vectors and target QoS values of training RAIs. The regression function can be used to predict the QoS of the RAIs in the test set. To reduce the number of RAIs for the following reevaluation, we remove all the RAIs whose QoS is worse than the current unoptimized RAI. Since the prediction time of the whole RAI test set is much shorter than the time of other steps, it can be neglected in calculating the overall optimization time. As soon as the prediction is done, all the feature vectors coupled with their RAIs will be ranked based on their predicted values in a descending order. Then, we need to validate the ranked RAIs using SMC iteratively starting from top-ranked RAIs. If one RAI is confirmed to have a success ratio larger than T using UPPAAL-SMC, the RAI will be reported as an optimal RAI. Otherwise, a failure will be reported if none of the ranked RAIs satisfies the given QoS threshold.

Algorithm 1: Our RAI optimization approach

```
Input:  i) Feasible resource allocation instances RAIs;
        ii) Regression method RM;
        iii) Optimal success ratio threshold T;
Output: An optimized RAI with success ratio no worse than T
RAIOptimization(RAIs, RM, T) begin
    1: FVs = Characterization(RAIs);
    2: (Tr, Te) = Sampling(FVs, RAIs);
    3: tr_result = {};
    for each (RAI, FV) pair in Tr do
        4: SR = SMC(RAI);
        if SR ≥ T then
            5: Return RAI;
        6: tr_result = tr_result ⋃ (RAI, FV, SR);
    7: Predictor = Regression(tr_result, RM);
    8: ranked_RAI = Rank(Predictor(Te));
    while ranked_RAI.empty() == FALSE do
        if SMC(ranked_RAI.deque()) ≥ T then
            9: Return RAI;
    10: if ranked_RAI.empty() == TRUE then
        Return NULL;
end
```

Algorithm 1 describes the details of our RAI optimization approach. Step 1 encodes the RAIs into feature vectors FVs. Step 2 divides the RAIs into the training set Tr and test set Te. Step 3 initializes an empty set to store the evaluation results for RAIs in Tr. Steps 4-6 iteratively check the RAIs in Tr using SMC and save all the trained results in tr_result. If there exists one RAI whose success ratio SR is larger than the give threshold T, it will be reported as an optimized RAI as shown in step 5. Step 7 generates the regression function with the specified regression approach and trained results. Step 8 calculates the predicted success ratio for each RAI in Te and ranks RAIs in a priority queue $ranked\_RAI$ based on the predicted value. Step 9 reevaluates the predicted RAIs one-by-one using SMC and terminates the iterations when finding one optimized RAI. Step 10 indicates that the optimization fails. In our approach, feature selection, sampling and regression function generation are the three most important steps, since they determine the accuracy of the prediction, which in turn determines the overall RAI optimization time. The following subsections will give the details of each of these steps.

4.5.1 Feature Selection for RAIs

To enable accurate prediction, it is required to figure out important features that are helpful in constructing learning machines for the prediction purpose. According to the criteria of significance, independence and diversity, our approach selects following three kinds of features of a resource allocation instance RAI: i) the unit cost and execution time distribution of each service in RAI (i.e., $U(v_i, RAI(v_i))$ and $VAR(v_i, RAI(v_i))$); ii) the overall cost of RAI without considering variation; and iii) the expected execution time of RAI. This is because that they are the most relevant factors in our variation-aware evaluation. If there are k services in the workflow, its feature vector will has $3*k+2$ elements. As an example shown in Figure 3, assume that $RAI_1 = \{(V_1, VM_{1,1}), (V_2, VM_{2,2}), (V_3, VM_{3,1}), (V_4, VM_{4,1}), (V_5, VM_{5,1})\}$ needs to be characterized. The following shows the feature vector of RAI_1, which will be used for the following regression and prediction.

$$<(6.0, 5.0, 0.6), (5.0, 6.0, 1.4), (4.5, 10.0, 0.8), (4.0, 10.0, 0.7), (5.0, 7.0, 1.5);\ 180.0;\ 28.0>$$

4.5.2 RAI Sampling

The goal of RAI sampling is to gather a set of feasible RAIs for regression function generation. To guarantee the unbiased prediction, such a set should be as representative as possible. If all the sampled training RAIs are from some conceptually local solution space, then the prediction may not be accurate. Therefore, it is required that all the sampled RAIs should be evenly distributed in the whole solution space. Based on the above observation, our approach supports two kinds of sampling heuristics to achieve the RAI training set: *instance-level sampling* and *VM-level sampling*. It is important to note that both of them are by no means the best sampling methods rather they are promising approaches in obtaining representative RAI training sets in practice.

Algorithm 2: Instance-level Sampling Approach

```
Input:  i) Sorted n services in sorted list S;
        ii) VM configuration VM[n][] for all services;
        iii) Sampling frequency f;
Output: Sampled RAI Set SRAIs
InstanceSampling(S, VM, n, i, f) begin
    for (k = 0; k < VM[i][].size() − 1; k++) do
        1. RAI[i] = VM[i][k];
        if i==n-1 then
            if Feasible(RAI) then
                2. RAIs = RAIs ⋃ RAI;
                if RAIs.size()%f==0 then
                    3. SRAIs = SRAIs ⋃ RAI;
        else
            4. InstanceSampling(S, VM, n, i+1, f);
    Return SRAIs;
end
```

Algorithm 2 shows the details of our instance-level sampling method which can uniformly sample the feasible RAIs that are sorted based on the generation order of RAIs. In this algorithm,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2586067, IEEE Transactions on Cloud Computing

RAI, *RAIs*, and *SRAIs* are all global data structures which indicate the current VM configuration of services, all collected feasible RAIs so far, and sampled RAIs, respectively. The function $Feasible(RAI)$ checks whether *RAI* is a feasible RAI. Step 1 allocates the i_{th} service with the k_{th} VM candidate $VM[i][k]$. Step 2 finds a new feasible schedule and stores it in the set *RAIs*. In step 3, if current feasible schedule *RAI* is the f_{th} feasible RAI since the last sampling, the *RAI* will be incorporated in *SRAIs* as a new sample RAI. Finally, *SRAIs* will contain all the sampled RAIs for the training purpose. It is important to note that, although the flow shown in Figure 6 separates the steps of RAI generation and sampling, in fact, they can be combined together to save the overall optimization time. Unlike instance-level sampling, the steps of RAI generation and VM-level sampling cannot be combined.

Although Algorithm 2 can evenly sample the RAIs from the feasible solutions, it cannot alway guarantee that the sampled RAIs are evenly scattered in the whole problem space. For example, during the RAI generation in a depth-first-search manner as shown in Algorithm 2, if some local part of the whole search space contains a large set of feasible solutions, then such part will have more sampled RAIs, which may bias the prediction results. To avoid such case, Algorithm 3 presents the details of our VM-level sampling approach which can backtrack non-chronologically by skipping proper number of VM candidates. In Algorithm 3, step 1 dispatches the VM resource $VM[i][k]$ to the i_{th} service. If all the services have been assigned with VMs and *RAI* satisfies customer requirement, *RAI* will be incorporated in the set of sample RAIs (i.e., *SRAIs*). Since our framework searches for the feasible RAIs in a depth-search-first manner, during the backtrack in the recursive search, steps 3 and 4 skip m candidate VM configurations for the i_{th} service to avoid local recursive sampling.

Algorithm 3: VM-level Sampling Approach

```
Input: i) Sorted n services in sorted list S;
       ii) VM configuration VM[n][] for all services;
       iii) VM-level sampling interval m;
Output: Sampled RAI Set SRAIs
VMSampling(S, VM, n, i, m) begin
    for (k = 0; k < VM[i][].size() − 1; k++) do
        1. RAI[i] = VM[i][k];
        if i==n-1 then
            if Feasible(RAI) then
                2. SRAIs = SRAIs ⋃ RAI;
                Return SRAIs;
        else
            3. VMSampling(S, VM, n, i+1, m);
            4. k = k + m − 1;
    Return SRAIs;
end
```

4.5.3 Regression Approaches for RAI QoS Prediction

Our approach adopts supervised learning approaches to conduct the prediction-based RAI optimization. After sampling as described in Section 4.5.2, all the RAIs in the training set need to be validated using UPPAAL-SMC tool to obtain their real QoS value considering variation information. The feature vectors of RAIs together with corresponding QoS values will be fed into supervised learning approaches to produce an inferred function, which can be used to predict the QoS of RAIs in the test set. Our framework supports three following most popular regression methods for optimal RAI prediction. Assume that there are n RAIs in the training set, and each feature vector of RAIs has m features.

Support Vector Regression

As a variant of Support Vector Machine (SVM), Support Vector Regression (SVR) [31] aims at finding a hyperplane in search space to predict the unknown data. In our approach, we conduct the prediction using ε-SVR approach from the SVM library LIBSVM [37]. To guarantee the prediction accuracy, we set $\varepsilon = 1$ by default. The input of the ε-SVR method is an $n \times (m+1)$ matrix, where each row indicates a feature vector together with its target value calculated by UPPAAL-SMC. For ε-SVR, we adopt the Radial Basis Function (RBF) kernel to enable the non-linear regression for the training set. When using RBF, proper hyper-parameters such as C (i.e., cost) and γ need to be tuned to achieve an expected prediction accuracy. To achieve optimal values for C and γ, a *k-fold* cross validation is required. The *k-fold* cross validation splits the training data into k folds of equal size and the ε-SVR is executed k times. Each time a different fold is chosen to serve as the training set. Therefore, if k is large, the training time using ε-*SVR* approach will be long.

Artificial Neural Network

Inspired by biological neural network, Artificial Neural Networks (ANNs) [32] are widely used to estimate or approximate functions for the purpose of regression. In our framework, we employ the Back Propagation Neural Network (BPNN) to perform the QoS prediction of RAIs. The input of the BP neural network is an $(m+1) \times n$ matrix, where each column consists of the feature vector together with its target value. The self-learning capability of BPNN relies on both the forward transfer of information and the reverse transfer of error between the expected outputs and actual outputs. A typical BPNN structure consists of three layers: input layer, hidden layer and output layer. In our approach, we use the training function *scaled conjugate gradient backpropogation* to update weight and bias values. For the BP neural network, we use the *tan-sigmoid* transfer function in the hidden layer which generates outputs between -1 and 1. The number of neurons in the hidden layer plays an important role in determining the prediction accuracy. However, more neurons will cost more training time. Therefore, proper tradeoff is required between the training time and prediction accuracy. Base on the experience in practice, our approach sets the number of neurons in the hidden layer to be 50 by default. Since the QoS value indicates the success ratio of the workflow execution, in the output layer, we use the *log-sigmoid* transfer function which generates outputs between 0 and 1 indicating the QoS of RAIs. During the training, the number of epoch iterations and the target Mean Squared Error (MSE) error are two key factors that affect the training time as well as the regression accuracy. To guarantee the low training time and regression accuracy, we set the number of iterations to 1000 epochs, and set the training *goal* of MSE to 0.001 by default.

M5 Model Tree

M5 model tree [33], [39] is a machine learning method that combines both decision tree and linear regression algorithm together. It has been successfully used as a predictor in many engineering fields [34]. By using the divide-and-conquer method, *M5 model tree* approach splits the parameter space into areas (subspaces) and builds in each of them a linear regression model. Since model trees are usually of small scale, they are very promising in tackling tasks with high dimensionality. Similar to the ε-SVR approach, the input of M5 model tree is an $n \times (m+1)$ matrix, where each row

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2586067, IEEE Transactions on Cloud Computing

denotes a feature vector together with its target value. Since few parameters are required to be tuned in the training using *M5 Model Tree* tools (e.g., Weka [39]), the input matrix can be fed into the *M5 Model Tree* tools to derive the regression function directly.

5 Case Study

This section presents the evaluation and optimization details of a Cloud-based securities exchange workflow for the Chinese Shanghai A-Share Stock Market [35] using our framework. Based on the interface (i.e., command line) provided by UPPAAL-SMC, our framework can conduct the evaluation of NPTA models with specified properties. In the experiment, we set the probability uncertainty of UPPAAL-SMC (i.e., ε) to 0.02, and set the probability of false negatives (i.e., α) to 0.01. To enable the RAI training, our framework incorporates three well-established regression tools: i) ε-SVR from the LIBSVM [37], ii) built-in BPNN from the MATLAB [38], and iii) *M5 model tree* from WEKA [39]. We implemented the remaining parts (i.e., RAI generation based on different strategies, NPTA model generation, property generation, and supervised learning-based RAI optimization)) of our framework shown in Figure 4 using the C programming language. All the experiments were conducted on a Windows 7 desktop computer with 2.8GHz Intel i7 CPU with 4 GB RAM.

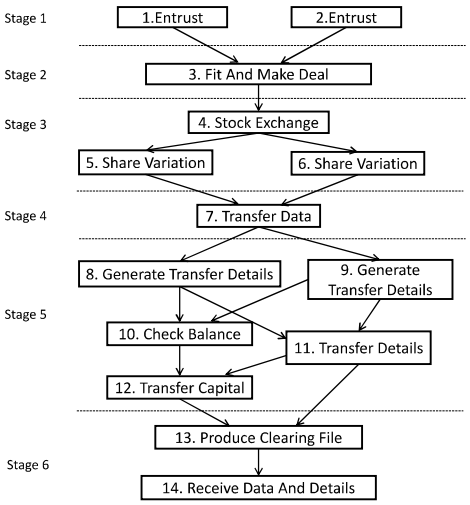


Fig. 7. DAG of the Securities exchange workflow [29]

5.1 Workflow Description

Figure 7 illustrates the DAG of the securities exchange workflow for the Chinese Shanghai A-Share Stock Market [29], [35]. The workflow consists of 14 nodes which represent the major workflow activities. Since a securities exchange workflow is a typical instance and computation intensive process, it is quite suitable to be deployed on a Cloud computing platform where the computing resources are provisioned by Cloud service providers according to customer's request and budget. As an example, in the stage 5 of the securities exchange workflow, services 8-12 are provided with multiple VM configurations as shown in Table 2. For example, the service 8 has three optional VM configurations. For the first VM, its unit cost is 180, and its expected execution time (i.e., mean time) and standard deviation are 10 and 1.05, respectively. In the following subsections, we will conduct the VM allocation evaluation and optimization based on the above securities exchange workflow and corresponding configurations. It is important to note that, to simplify the illustration of the effectiveness of our approach, in the experiment of strategy evaluation (Section 5.2) and SLA negotiation (Section 5.3), we only focus on the resource allocation of the first two VM configurations (i.e., *configuration 1* and *configuration 2*) for the services in the stage 5 of the securities exchange workflow. We assume that the other 9 nodes in the workflow have fixed execution time without considering variations. Unlike RAI evaluation and negotiation, we take all the VM configurations into account in the experiment of RAI optimization (Section 5.4).

5.2 Strategy Evaluation

In this experiment, we assumed that the customer wants *the whole workflow to be completed within 115 time units and 13500 cost units, and the success ratio to be no lower than* 80%. We evaluated the securities exchange workflow using the self-contained default strategies (i.e., TCCM, CCTM, and xRFI) implemented in our framework. For the purpose of evaluation, we only generated one RAI for each strategy. It is important to note that the time of NPTA model generation from an RAI is quite small ($<$ 0.01 second) and can be negligible, since it only needs to figure out the back-end configuration. Generally, the RAI evaluation time is significantly larger than the RAI generation time. In this experiment, due to the similar complexity of properties and generated NPTA models from RAIs, the evaluation time for each RAI is similar (4 to 5 minutes on average), which dominates the overall strategy evaluation time.

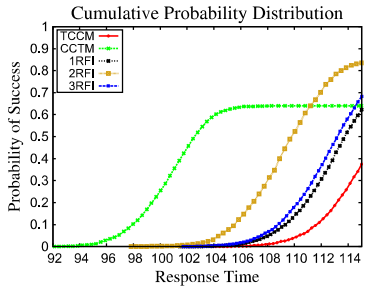


Fig. 8. CPD for R(13500,115, 80%)

As soon as receiving a customer request, our framework firstly applies the SRMS approach on the generated RAIs to check whether such requirement can be satisfied or not with our default strategies. In this example, since we set $x=3$ for xRFI, we obtained 5 resource allocation instances using the strategies TCCM, CCTM, 1RFI, 2RFI, and 3RFI. Since xRFI does not need to explore all feasible RAIs, strategy xRFI ($<$ 0.01 second) costs much less time than both the strategy TCCM (0.27 second) and CCTM (0.29 second). Figure 8 presents the Cumulative Probability Distribution (CPD) of the response time of successful simulation runs. Interestingly, we can find that TCCM and CCTM did not achieve the best performance in this case, though their targets are to find cost-optimal and time-optimal solutions, respectively. Since the workflow needs to be completed with a success ratio no lower than 80%, the RAIs generated by TCCM, CCTM, 1RFI and 3RFI need to be discarded. According to the evaluation report of our framework, the 2RFI instance has a confidence of 0.99 to obtain a success ratio within [0.816, 0.856], which is higher than all the other RAIs. Therefore, the instance 2RFI was selected finally.

To investigate the effect of different requirement parameters, we applied the *MRMS* approach. Based on the example shown in Figure 8, we tuned the cost (increased by 300 cost units) and time (increased by 5 time units) respectively as shown in Figure 9 and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2586067, IEEE Transactions on Cloud Computing

TABLE 1
VM Configurations for the Securities Exchange Workflow

	Config. 1			Config. 2			Config. 3			Config. 4		
Node	U.C.	E.T. (μ)	S.D. (σ)	U.C.	E.T. (μ)	S.D. (σ)	U.C.	E.T. (μ)	S.D. (σ)	U.C.	E.T. (μ)	S.D. (σ)
1	100	15	1.05	80	16	1.10	50	18	1.12	40	20	1.50
2	60	8	1.10	45	10	1.20						
3	40	10	1.05	30	12	1.10						
4	100	7	1.05	80	8	1.10	70	9	1.12	60	10	1.50
5	60	12	1.05	40	15	1.10						
6	100	8	1.05	70	9	1.10	60	12	1.20			
7	100	8	1.05	70	9	1.10	60	12	1.20			
8	180	10	1.05	150	12	1.10	120	14	1.20			
9	100	8	1.05	80	9	1.10	70	10	1.20			
10	60	8	1.05	40	10	1.20	50	9	1.10			
11	120	12	1.05	90	15	1.20	100	14	1.10			
12	100	14	1.10	120	12	1.05	90	15	1.20			
13	100	8	1.05	80	10	1.10						
14	120	10	1.10	100	11	1.20						

Figure 10. From Figure 9, we can find that, due to the increase of the overall cost, the workflow can get better VMs in their RAIs. Therefore, the instances CCTM, 1RFI and 2RFI can have better success ratio except the TCCM and 3RFI instances. For the TCCM strategy, increasing the cost did not make any change to the TCCM instance shown in Figure 8. This is because that the TCCM instances shown in Figure 8 and Figure 9 are the same, since TCCM tries to find the cheapest configurations for the workflow. However, when the user relaxes the time limit to 120 as shown in Figure 10, we can observe the increase of success ratio for all the RAIs except the CCTM instance. Especially for the TCCM instance, we can find a significant improvement. This is because that the TCCM instance suffers from the tight response time constraints. It is important to note that, if we set the time limit to 115 in Figure 10, we can find that the success ratio of all RAIs is almost the same as the success ratio of RAIs in Figure 8. In Figure 10, if the customer cares more about the response time, the 2RFI instance will be selected at last.

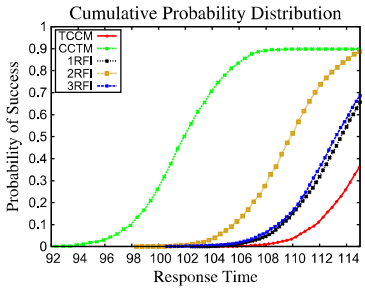


Fig. 9. CPD for R(13800,115, 80%)

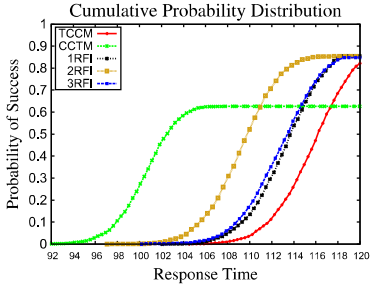


Fig. 10. CPD for R(13500,120, 80%)

MRSS approach can be used to investigate the performance of a single strategy under different requirements. Figure 11 and Figure 12 show the application of MRSS for both the TCCM and CCTM strategies, respectively. In Figure 11, the legend item *C 13500(12650), T 118(117)* means that the requirement provided by the customer is *R(13500, 118, -)*. But by using the TCCM strategy, we can get an RAI which needs 12650 cost units and 117 time units based on the expected response time of each service in the workflow. Since TCCM strategy focuses on the effects of response time on the cost, in Figure 11, we created different requirements with the same cost but different response time. For the CCTM strategy, Figure 12 adopts different requirements with the same response time but different cost.

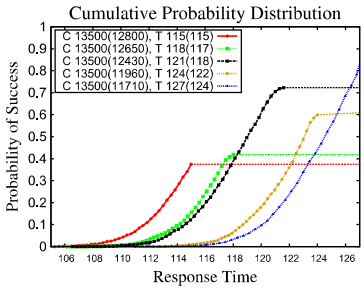


Fig. 11. CPD for TCCM Strategy

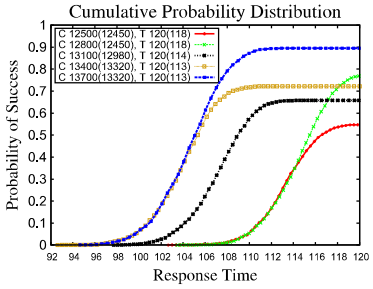


Fig. 12. CPD for CCTM Strategy

From Figure 11, we can find that by using TCCM, when relaxing the time constraint, the expected cost can be reduced accordingly. Generally, the more the time constraint is relaxed, the higher the probability of success will be achieved. However, compared to the constraint *R(13500, 121, -)*, relaxing the time constraint to 124 (i.e., *R(13500, 124, -)*) does not get a better probability of success. This is because that the RAI generated under the constraint *R(13500, 124, -)* may have higher overall variations, which can affect the overall RAI success ratio. Figure 11 shows that under the constraint *R(13500, 127, -)*, with the lowest cost, we can achieve the best probability of success. From Figure 12, we can observe that the increase of cost can lead to the reduction of workflow response time. For example, under the constraint *R(13700, 120, -)*, it only needs 105.5 time units to achieve a success ratio of 55%. However, when the constraint is *R(12500, 120, -)*, achieving a success ratio of 55% needs at least 120 time units. This is because that more high-end VMs are

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2586067, IEEE Transactions on Cloud Computing

used in the newly derived instances when increasing the budget. In can be found that under the constraint *R(13700, 120, -)*, we can achieve the best RAI success ratio. Interestingly, from this figure we can observe that when using CCTM strategy, the instances (i.e., in yellow and black color) with higher cost may have worse success ratio than the cheaper RAI (i.e., in green color). This is because that in our experiment the generation of RAIs using CCTM strategy does not take the variation into consideration.

5.3 SLA Negotiation

Workflow response time and success ratio are two most important QoS issues in real-time Cloud applications. However, due to the budget limit, when a customer initially bargains with an SaaS provider, he/she may bid the lowest price first. Then the SaaS provider will try to evaluate the RAIs generated under the specified constraints and strategies. If there exist RAIs that meet the QoS requirement, the RAI with the best performance will be selected as the solution. However, if all of the generated RAIs cannot guarantee the QoS requirement, the SaaS provider will utilize our framework to conduct the negotiation with the customer. In this experiment, we assumed that the SaaS provider increases the price by 5% each time without modifying response time constraint.

TABLE 2
Resource Allocation Instances in Different Round

Round No.	Requirement Constraints	Allocation (Service → Config.)				
		8	9	10	11	12
1	(12500, 120, 85%)	2	2	2	2	1
2	(13125, 120, 85%)	1	2	2	2	1
3	(13780, 120, 85%)	1	2	1	1	2
4	(14450, 120, 85%)	1	1	1	1	2

Assume that a customer requires that securities exchange workflow should be finished in 120 time units and the success ratio cannot be smaller than 85%. Initially, the customer only offers a price of 12500 cost units. Due to the space limit, to demonstrate how SaaS providers utilize our approach to bargain with the customer, we only use the CCTM strategy to generate RAIs. It is important to note that our framework supports the negotiation using multiple strategies simultaneously, which may achieve a better price with fewer bargain rounds. In this example, we conducted 4 rounds of bargains in total. Table 2 shows the change of customer requirements as well as the corresponding resource allocation upgrade information in each bargain round.

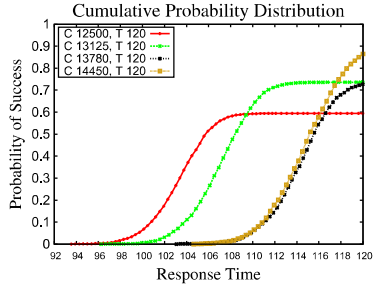


Fig. 13. A negotiation example using our framework

Figure 13 shows the evaluation results for the 4 bargains. From this figure, we can find that in the first round of the bargain the requirement *R(12500, 120, 85%)* fails to be satisfied since the success ratio reported by our evaluation framework is smaller than 60%, which is not acceptable by the customer. Then in the second round, the SaaS provider raises the cost by 5%. However, the newly generated RAI fails again. In the fourth round of bargain, the customer and SaaS provider close the deal with a price of 14450 cost units. Since the evaluation of each RAI using UPPAAL-SMC needs around 4 minutes, the overall bargain process last for around 16 minutes in total. In fact, all the four rounds can be evaluated simultaneously. In this way, the overall bargain time can be reduced.

5.4 Resource Allocation Optimization

In the optimization experiment, we consider all the combination of VM configurations listed in Table 2. We assume that the customer requirement is $(13500, 115, 75\%)$, and the optimization target is $(13500, 115, 95\%)$. It is important to note that in our framework, the RAI generation step tries to explore all the feasible schedules without considering the resource variation information. The RAI generation costs 7.37 seconds. Compared with the following optimization steps, this time is negligible. Filtered by the customer requirement, 79850 feasible RAIs were generated as the candidates of optimal RAIs. Based on our experimental results, the evaluation of an RAI in this set using UPPAAL-SMC will cost around 5 minutes on average. If we evaluate all the RAIs sequentially, the optimization will cost around 277 days, which is not acceptable by both customers and service providers. Therefore, our framework resorts to the supervised learning approaches to reduce the optimization efforts.

Since there are 14 nodes in the workflow, for each RAI, we generated a feature vector containing $14 \times 3 + 2 = 44$ features indicating both the workflow execution and resource variation information for each RAI. We conducted the experiment using the two proposed sampling methods individually. We use the RAIS and VMS to indicate the RAI-level sampling and VM-level sampling separately. When using RAIS, we set the sampling frequency to 300. When using VMS, we set the sampling interval to 2. To ease the comparison between the two sampling methods, the two training sets have the same number of RAIs (i.e., 256). In other words, during the RAI sampling, if the size of the training set equals to 256, the following sampling process will be skipped. After the generation of RAI training sets, we calculated the target value of each sampled RAI using the tool UPPAAL-SMC. The target value calculation costs around 13 hours for each RAI training sets generated by the two sampling methods. Unfortunately, during the training process, we did not find any RAIs that have a success ratio larger than 95%. We applied the three regression approaches (i.e., ε-SVR, BPNN and *M5 model tree*) on the training sets separately. We utilized the parameter settings described in Section 4.5.3. Based on the generated regression models, we ranked all the remaining $79850 - 256 = 79594$ feasible RAIs in the test set. Since the prediction and the ranking just need around 1 second, their time can be negligible compared to the evaluation time for a single RAI using SMC.

Table 3 shows the comparison of the RAI optimization with different sampling and regression methods. The unit of time in this table is the second. In this table, the first column presents the name of sampling approaches. The second column gives the SMC-based target value calculation time for the training sets generated by the two sampling approaches. The third column lists the adopted regression approaches. The fourth column denotes the time of regression model generation. Since ε-SVR employs the multiple-fold cross validation, it needs much more time than the other regression approaches. For example, when using RAIS sampling, ε-SVR costs 4680 seconds, which is far more than the other two regression methods. The fifth column has two sub-columns which

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2586067, IEEE Transactions on Cloud Computing

present the details of iterative RAI checking information after the RAI ranking. The first sub-column N indicates the number of iterations, and the second sub-column T denotes the total time of all the iterations before finding an optimal RAI. For instance, $N = 8, T = 2371$ means that the $8th$ top ranked RAI is an optimal solution whose success ratio is better than 95%, and the total evaluation time all the top ranked 8 RAIs is 2371 seconds. Although the predicted success ratios of all the previous 7 RAIs are no worse than the 8_{th} RAI, their evaluation results reported by UPPAAL-SMC are all below 95%. For each combination of sampling and regression methods, the sixth and seventh columns present the regression-based predicted value and UPPAAL-SMC-based evaluated value of the found optimal RAI.

TABLE 3
Comparison of Different Optimization Methods

	SMC Time	Regression	Tr.	Iteration		P.V.	R.V.
				N	T		
RAIS	63070	ε-SVR	4680	8	2371	0.953	0.951
		BPNN	4.33	9	2792	0.951	0.950
		M5P	17.56	3	923	0.958	0.954
VMS	63955	ε-SVR	4970	10	3190	0.947	0.962
		BPNN	4.56	7	2041	0.954	0.959
		M5P	16.43	2	577	0.961	0.951

From Table 3, we can find that *M5P model tree* has the best prediction accuracy, since it needs less validation iterations than other regression methods after the RAI ranking. In other words, it can quickly find an optimal RAI. Although ε-SVR spends more time than other approaches in the regression model generation, its prediction accuracy is not the best among three regression methods. Moreover, in Table 3, the optimization that adopts VMS and ε-SVR methods has the highest time cost (72115 seconds, or 20 hours). To compare with our approach, we also tried to find an optimal RAI without using the supervised learning-based approach. We evaluated the RAIs one by one in the order of feasible RAI generation. However, after the first 10000 feasible RAIs were evaluated, we still did not find an RAI with a success ratio larger than 95%. The whole process costs around 1200 machine hours, which is far more than the time using our approach.

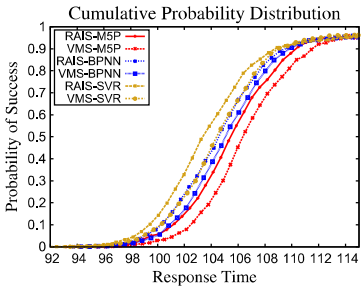


Fig. 14. Comparison of optimal RAIs generated in Table 3

Figure 14 shows the results of the 8 optimized RAIs generated in Table 3. Although all these 8 RAIs can achieve a success ratio of 95% under the customer requirement, the response time using different sampling and regression methods is different. From this figure, we can find that both the RAIS-SVR and VMS-SVR RAIs generated by ε-SVR method have the best response time, and the optimal RAIs generated by the RAIS sampling method can achieve better response time than their VMS counterparts.

Figure 15 compares the non-optimized RAIs generated using the default resource allocation strategies of our framework with the optimized RAIs using the supervised learning-based approach. In Figure 15, we only show the optimized RAIs using the VMS sampling method. We can find that the performance (i.e., success ratio and response time) of the RAIs derived by the default RAI generation strategies can be significantly improved by our optimization approach. Someone may argue that the RAI optimization process is too time-costly. However, due to the benefit of the improved response time and success ratio, it is still worthy to make such optimization from the perspectives of service providers.

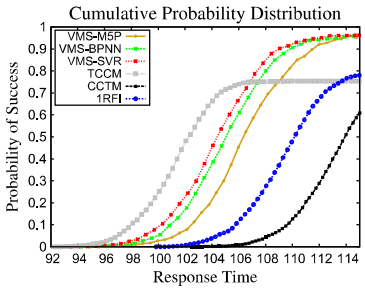


Fig. 15. Comparison of the optimized and non-optimized RAIs

6 Conclusions

For SaaS providers, an effective cloud workflow resource allocation strategy can not only reduce overall operating costs, but also reduce SLA violations. However, due to the inherent complexity of accumulative variations caused by individual services in a workflow, so far there is no approach that can quantitatively reason the capability of resource allocation strategies. To address this problem, this paper proposes a UPPAL-SMC-based framework that enables the accurate modeling and evaluation of resource allocation strategies under different kinds of variations. Besides the comparison among resource allocation strategies, our framework enables the SLA negotiation as well as QoS optimization automatically and efficiently. Comprehensive experimental results demonstrate the effectiveness of our framework.

References

[1] X. Liu, D. Yuan, G. Zhang, W. Li, D. Cao, Q. He, J. Chen and Y. Yang. "The Design of Cloud Workflow Systems," *Springer*, 2012.
[2] A. Rai, R. Bhagwan and S. Guha. "Generalized Resource Allocation for the Cloud," *ACM Symposium on Cloud Computing (SOCC)*, 2012, 15.
[3] Z. Wu, X. Liu, Z. Ni, D. Yuan and Y. Yang. "A Market-Oriented Hierarchical Scheduling Strategy in Cloud Workflow Systems," *The Journal of Supercomputing*, vol. 63, no. 1, pp. 256–293, 2013.
[4] T. D. Braun, H. Siegel, A. A. Maciejewski and Y. Hong. "Static resource allocation for heterogeneous computing environments with tasks having dependencies, priorities, deadlines, and multiple versions," *Journal of Parallel and Distributed Computing*, vol. 68, no. 11, pp. 1504-1516, 2008.
[5] P. Bulychev, A. David, K. Larsen, M. Mikucionis, D. Poulsen, A. Legay and Z. Wang. "UPPAAL-SMC: Statistical Model Checking for Priced Timed Automata," *International Workshop on Quantitative Aspects of Programming Languages and Systems (QAPL)*, 2012, pp. 1–16.
[6] A. David, K. G. Larsen, A. Legay, M. Mikucionis and Z. Wang. "Time for Statistical Model Checking of Real-Time Systems," *International Conference on Computer Aided Verification (CAV)*, 2011, pp. 349–355.
[7] T. Hastie, R. Tibshirani and J. Friedman. "The Elements of Statistical Learning: Data Mining, Inference, and Prediction," *Springer*, 2009.
[8] Y. Lee, C. Wang, A. Y. Zomaya and B. Zhou. "Service Level Agreement Based Distributed Resource Allocation for Streaming Hosting System," *Int. Conf. on Cluster, Cloud and Grid Computing (CCGrid)*, 2010, pp. 15-24.
[9] L. Wu, S. Kumar Garg and R. Buyya. "SLA-Based Resource Allocation for Software as a Service Provider (SaaS) in Cloud Computing Environments," *International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, 2011, 195–204.
[10] S. K. Garg, C. Vecchiola and R. Buyya. "Mandi: A Market Exchange for Trading Utility and Cloud Computing Services," *The Journal of Supercomputing*, vol. 64, no. 3, pp. 1153-1174, 2013.
[11] ViTLive. *http://vitlive.com/*.
[12] F. H. Zulkernine and P. Martin. "An Adaptive and Intelligent SLA Negotiation System for Web Services," *IEEE Transactions on Services Computing*, vol. 4, no. 1, pp. 31-43, 2011.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2586067, IEEE Transactions on Cloud Computing

[13] A. Va. Dastjerdi and R. Buyya. "An Autonomous Reliability-Aware Negotiation Strategy for Cloud Computing Environments," *Int. Conference on Cluster, Cloud and Grid Computing (CCGrid)*, 2012, 184–291.
[14] L. Wu, S. Kumar Garg, R. Buyya, C. Chen and S. Versteeg. "Automated SLA Negotiation Framework for Cloud Computing," *Int. Conference on Cluster, Cloud and Grid Computing (CCGrid)*, 2013, 235–244.
[15] P. Xiong, Y. Chi, S. Zhu and H. J. Moon. "Intelligent Management of Virtualized Resources for Databased Systems in Cloud Environment," *Int. Conference on Data Engineering (ICDE)*, 2011, pp. 87–98.
[16] C. Huang, Y. Wang, C. Guan, H. Chen and J. Jian. "Application of Machine Learning to Resource Management in Cloud Computing," *Int. Journal Modeling and Optimization*, vol. 3, no. 2, pp. 148–152, 2013.
[17] I. Menache, O. Shamir and N. Jain. "On-Demand, Spot, or Both: Dynamic Resource Allocation for Executing Batch Jobs in the Cloud," *Int. Conf. on Autonomic Computing (ICAC)*, 2014, pp. 177–187.
[18] M. Chen, D. Yue, X. Qin, X. Fu and P. Mishra. "Variation-Aware Evaluation of MPSoC Task Allocation and Scheduling Strategies using Statistical Model Checking," *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2015, pp. 199–204.
[19] F. Gu, X. Zhang, M. Chen, D. Grosse and R. Drechsler. " Timing Analysis of UML Activity Diagrams Using Statistical Model Checking," *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2016, accepted.
[20] D. Du, M. Chen, X. Liu and Y. Yang. "A Novel Quantitative Evaluation Approach for Software Project Schedules using Statistical Model Checking," *International Conference on Software Engineering (ICSE) Companion*, 2014, pp. 476–479.
[21] A. David, D. Du, K. G. Larsen, A. Legay and M. Mikucionis. "Optimizing Control Strategy Using Statistical Model Checking," *International Symposium on NASA Formal Methods*, 2013, pp. 352-367.
[22] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. Rose and R. Buyya. "CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms," *Software - Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.
[23] W. Chen and E. Deelman. "WorkflowSim: A Toolkit for Simulating Scientific Workflows in Distributed Environments," *International Conference on E-Science*, 2012, pp. 1-8.
[24] A. Rozinat, M. Wynn, W. Aalst, A. Hofstede and C. Fidge. "Workflow Simulation for Operational Decision Support," *Data & Knowledge Engineering*, vol. 68, no. 9, pp. 834–850, 2009.
[25] D. Schuller, U. Lampe, J. Eckert, R. Steinmetz and S. Schulte. "Cost-driven Optimization of Complex Service-based Workflows for Stochastic QoS Parameters," *International Conference on Web Services (ICWS)*, 2012, pp. 66–73.
[26] X. Liu, Y. Yang, Y. Jiang and J. Chen. "Preventing Temporal Violations in Scientific Workflows: Where and How," *IEEE Transactions on Software Engineering (TSE)*, vol. 37, no. 6, pp. 805-825, 2011.
[27] M. Rahman, M. R. Hassan, R. Ranjan and R. Buyya. "Adaptive Workflow Scheduling for Dynamic Grid and Cloud Computing Environment," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 13, pp. 1816-1842, 2013.
[28] S. Huang, M. Chen, X. Liu, D. Du and X. Chen. "Variation-Aware Resource Allocation Evaluation for Cloud Workflows using Statistical Model Checking," *IEEE International Conference on Big Data and Cloud Computing (BDCloud)*, 2014, pp. 201–208.
[29] X. Liu, Z. Ni, J. Chen, and Y. Yang. "A Probabilistic Strategy for Temporal Constraint Management in Scientific Workflow Systems," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 16, pp. 1893-1919, 2011.
[30] M. A. Rodriguez and R. Buyya. "Deadline based Resource Provisioning and Scheduling Algorithm for Scientific Workflows on Clouds," *IEEE Transactions on Cloud Computing*, vol. 2, no. 2, pp. 222-235, 2014.
[31] H. Drucker, C. J. C. Burges, L. Kaufman, A. J. Smola and V. Vapnik. "Support Vector Regression Machines," *Advances in Neural Information Processing Systems (NIPS)*, 1996, pp. 155–161.
[32] D. F. Specht. "A General Regression Neural Network," *IEEE Transactions on Neural Networks*, vol. 2, no. 6, pp. 568-576, 1991.
[33] J. R. Quinlan. "Learning with Continues Classes," *5th Australian Joint Conference on Artificial Intelligence*, 1992, pp. 343–348.
[34] M. Bhattacharya and D. P. Solomatine. "Neural Networks and M5 Model Trees in Modelling Water Level-Discharge Relationship," *Neurocomputing*, vol. 63, pp. 381-396 , 2005.
[35] Chinese Shanghai A-Share Stock Market. *http://www.sse.com.cn/ sseportal/en/*.
[36] Clearing Corporation of China. *http://www.chinaclear.cn/*.
[37] LIBSVM. *http://www.csie.ntu.edu.tw/∼cjlin/libsvm*.
[38] MATLAB. *http://www.mathworks.com*.
[39] Weikato Environment for Knowledge Analysis (WEKA). *http://www.cs.waikato.ac.nz/∼ml/weka*.

Mingsong Chen (S'08–M'11) received the B.S. and M.E. degrees from Department of Computer Science and Technology, Nanjing University, Nanjing, China, in 2003 and 2006 respectively, and the Ph.D. degree in Computer Engineering from the University of Florida, in 2010. He is currently a Professor with the Software Engineering Institute of East China Normal University. His research interests include design automation of complex systems, formal verification and software engineering.

Saijie Huang received the B.S. degree from Department of Computer Science and Technology, Tongji University, Shanghai, China, in 2012. He is currently a master student with the Software Engineering Institute of East China Normal University. His research interests are in the area of cloud computing, formal verification techniques and software testing.

Xin Fu (S'05-M'10) received the Ph.D. degree in Computer Engineering from the University of Florida in 2009. From 2010 to 2014, she was an Assistant Professor at the Department of Electrical Engineering and Computer Science, the University of Kansas, Lawrence. Currently, she is an Assistant Professor at the Electrical and Computer Engineering Department, the University of Houston, Texas. Her research interests include computer architecture, high-performance computing, hardware reliability and variability, and energy-efficient computing. Dr. Fu is a recipient of 2014 NSF Faculty Early CAREER Award and 2012 Kansas NSF EPSCoR First Award.

Xiao Liu (M'11) received his master degree in management science and engineering from Hefei University of Technology, Hefei, China, 2007, and received his PhD degree in the Faculty of Information and Communication Technologies at Swinburne University of Technology, Melbourne, Australia, 2011. He worked as an associate professor in Software Engineering Institute at East China Normal University from 2013 to 2015. He is currently a senior lecturer at School of Information Technology, Deakin University, Melbourne, Australia. His research interests include software engineering, workflow management systems and cloud computing.

Jifeng He received the B.S. degree from Department of Mathematics, Fudan University, Shanghai, China, in 1965. From 1984 to 1998, He Jifeng was a senior research fellow at the programming research group in the Oxford University computing laboratory (now the Department of Computer Science at Oxford University). He is currently a distinguished professor and the dean of the Software Engineering Institute at East China Normal University. He is an academician of Chinese Academy of Sciences. His research interests include Internet of things, cyber-physical systems, formal aspects of computing science and software engineering.