2014 IEEE Fourth International Conference on Big Data and Cloud Computing

Variation-Aware Resource Allocation Evaluation for Cloud Workflows using Statistical Model Checking

Saijie Huang, Mingsong Chen*, Xiao Liu, Dehui Du and Xiaohong Chen
Shanghai Key Laboratory of Trustworthy Computing, East China Normal Univeristy, Shanghai, China, 200062
Email: {sjhuang,mschen*,xliu,dhdu,xhchen}@sei.ecnu.edu.cn

***Abstract*—Aiming at minimizing service operating costs and SLA (Service Level Agreement) violations, various resource allocation strategies have been investigated to support Cloud service providers' decision making. However, due to the service execution time variation, traditional optimal resource allocation strategies cannot achieve the best performance in practice. To address this problem, we propose an automated variation-aware evaluation framework for resource allocation strategies based on statistical model checker UPPAAL-SMC. Our framework can systematically evaluate the performance of resource allocation strategies under variations, and conduct complex queries on the quality of service. The experimental results show that our framework can not only filter inferior solutions efficiently, but also can enable the tuning of requirement constraints. Since our approach can be fully automated, the human efforts in resource allocation strategy evaluation can be significantly reduced.**

I. Introduction

Due to the fierce competition in today's Cloud computing [1] market, to achieve increasing profit, Cloud service providers should look into solutions that can minimize the cost of infrastructure services without adversely affecting the Quality of Services (QoS) [2]. In a Cloud workflow system, a customer sends a workflow request with specified QoS requirements. Such requirements usually contain the information of expected price, response time, reliability (bearable failure ratio), and etc. For example, customers would like to ask the question *"Is the reliability that the workflow can be completed in x hours with a cost of y dollars larger than z?"*. Since different underlying Virtual Machines (VM) offered by different service providers (e.g., Amazon EC2, Microsoft Windows Azure) may have different capacity (e.g., CPU, memory, storage) and on-demand prices, Cloud service providers need to figure out an allocation of VMs, which should be profitable and can meet all the customer requirements.

Resource allocation is an NP-complete problem, because it deals with the assignment of services to VMs considering multiple constraints (e.g., service precedence, time, cost). Various heuristic-based strategies are proposed to quickly find a solution (i.e., a resource allocation instance) [16]. However, due to the accumulated variations in time and cost, it is hard for Cloud service providers 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 and achieve an acceptable profit for Cloud service providers, resource allocation strategy evaluation must address following issues: i) How to accurately model workflow-based services and customer requirements to enable the quantitative evaluation? And ii) How to model the time and cost variations caused by underlying infrastructures?

Although simplified probability-based approaches can be used to model execution variation, few of them can accurately model parallel service execution. Moreover, existing constraint solving approaches can only answer *yes* or *no* for the 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 of powerful evaluation approaches which can help Cloud service providers to make choices among the candidate resource allocation solutions.

Statistical Model Checking (SMC) [15], [8] 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. Therefore, SMC is suitable for reasoning the QoS of resource allocation strategies in Cloud computing. Based on SMC, this paper makes two 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 three effective evaluation strategies that can filter inferior resource allocation solutions and enable the QoS constraint tuning.

The remainder of the paper is organized as follows. Section II presents related works on QoS oriented resource allocation strategies in Cloud and SMC-based evaluation approaches. After the introduction of the preliminary knowledge of UPPAAL-SMC in Section III, Section IV describes our resource allocation strategy evaluation framework in details. To

This work was supported by NSF of China (61202103, 61300042), Innovation Program of Shanghai Municipal Education Commission (14ZZ047), and Shanghai Knowledge Service Platform Project (ZF1213).

* Mingsong Chen is the corresponding author.

978-1-4799-6719-3/14 $31.00 © 2014 IEEE
DOI 10.1109/BDCloud.2014.48

demonstrate the efficacy of our approach, Section V presents various experimental results. Section VI concludes the paper.

II. Related Work

With the advancement of web technology, resource allocation is becoming a key issue in distributed application deployment [9]. Unlike traditional market-based resource allocation methods which are non-pricing-based [10], in Cloud, various SLA- or QoS-based profit maximization resource allocation approaches were proposed [2], [3], [4]. Popovici and Wilkes investigated the profit-aware schedulers with resource uncertainty [11]. They studied multiple uncertainty factors that influence service providers' profit, including load, user impatience, number of resources, price, and etc. Based on a novel pricing model for Clouds, Lee et al. [12] investigated the profit driven service request scheduling for workflow. In [14], Song et al. presented a priority-based resource flowing algorithms named *RFaVM* to optimize resource allocations amongst services. To manage the dynamic change of customers, Wu et al. [6] proposed resource allocation algorithms for SaaS providers who want to minimize infrastructure cost and SLA violation. To enable efficient dynamic application scaling, Gambi and Toffetti [5] proposed a performance modeling approach that can predict the system performances of different resource allocations. However, most existing approaches focus on the modeling and optimization rather than the quantitative evaluation for resource allocation with variations.

Due to the efficacy in quantitative query of performance metrics, SMC [15] is promising in evaluating system designs. In [19], Du et al. adopted UPPAAL-SMC [8] to conduct quantitative evaluation on project schedules. Their approach supports various evaluation queries for the schedule comparison. In [17], David et al. extended the semantics of UPPAAL-SMC to enable the modeling of networks of hybrid systems. Based on stochastic hybrid automata, their approach can be used to perform modeling and evaluation of objects in various domains, including biology and energy-aware buildings. Moreover, by exploiting the stochastic semantics together with simulation, their approach can achieve best values for model parameters to enable design optimizations [18]. However, SMC-based approaches are seldom used in the modeling and evaluation of resource allocation strategies in Cloud.

Different from traditional Cloud simulators (e.g., CloudSim [13]) which conducts the full-fledged simulation of Cloud computing systems and application provisioning environments, our approach focuses on the resource allocation strategy evaluation using light-weight simulation provided by UPPAAL-SMC. To the best of our knowledge, our work is the first SMC-based approach that can evaluate resource allocation strategies under variations for Cloud.

III. Preliminary

Our approach adopts the Priced Timed Automata (PTAs) [7], [8] to model the behaviors of services. A Network of Price Timed Automata (NPTA) comprises a set of correlated PTAs that communicate via broadcast channels and shared variables. As an example, Figure 1 shows an NPTA with two PTAs A (id=ida) and B (id=idb), where each PTA has four locations and two local clock (e.g., c_1 and C_a in A) respectively. In PTAs, clocks can evolve with different rates in different locations. The value of a primed clock indicates the rate of the clock. For example, $C_a' == 2$ is used to record the cost spent in location A_3 with a rate of 2. Unprimed Clocks has a rate of 1 by default. In this example, we use an array of broadcast channels $msg[id]$ for the purpose of synchronization, where id indicates the target PTA of the message. While using message-based synchronization, we adopt the non-deterministic *selections* to filter useless messages. In Figure 1, on the outgoing edge of location B_1, the *selections* *e:msg_t* and the *guard condition* *e==idb* are used to filter messages which are not sent to B.

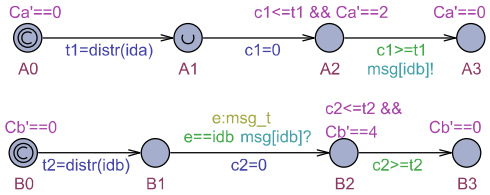

Fig. 1. An NPTA, (A | B)

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 normal and exponential distributions explicitly, which cannot cover all complex scenarios in practical designs. To enable the modeling of different kinds of stochastic behaviors, we adopt the pattern shown in Figure 1. By proper programming using the built-in function *random*(), the self-defined function *distr*() can produce values following a large set of commonly used distributions. 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 of t_1 (randomly generated by *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 normal 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 20.4. Assuming that there is no correlation between clocks n_1 and n_2, while more runs are simulated, we can find that the time of reaching composite location (A_3, B_3) will be in a normal distribution $N(9, 1^2 + 2^2)$. By using the message-based synchronization among parally running PTAs, arbitrarily complex stochastic behavior can be modeled.

Based on the stochastic semantics, SMC models enable the generation of random runs which are bounded by either time, cost or a number of discrete steps. During SMC, all these derived runs have to be monitored with some specified properties in the form of cost-constraint temporal logic [7]. At the end of the SMC, the probability range of each property with a specified confidence will be reported. Currently UPPAAL-SMC can handle following three kinds of queries:

- **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. Such check can be used for SLA negotiation by Cloud service brokers. The quantitative check can be used to conduct the interval estimate for the success ratio of the given property. In our approach, it can be used to evaluate the performance of different resource allocation strategies. The property comparison can be used to filter inferior strategies.

IV. Our Approach

In this section, we formulate the variation-aware resource allocation strategies for Cloud (Section IV-A). We propose a novel framework (Section IV-B) based on UPPAAL-SMC which can conduct modeling and evaluation of resource allocation strategies (Section IV-C).

A. Problem Definition

Our approach only considers the unit price, time variation and QoS (i.e., the success ratio of completing service workflow on time) during the resource allocation. Since unit price can be considered as a kind of special resources (e.g., power), our approach can be easily extended to solve the problems in other domains. To simplify the modeling of resource allocation problem, our approach assumes that the profit ratio required by Cloud service 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 in 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 [20], 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 binded 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 $\mathbf{C}$ 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 probability of successful workflow execution which meets 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.

B. Framework of Our Approach

Figure 2 presents our UPPAAL-SMC-based framework that can help Cloud service providers to make decisions on resource allocation. Firstly, based on the Cloud service workflow and customer requirements extracted from some SLA contract, the Cloud service 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 of services will be queried from the Cloud service provider itself or other Cloud service providers. Since no existing approaches can always generate best resource allocation instances under time and cost variations, to obtain a near-optimal resource allocation instances 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 be selected. They are selected by Cloud service providers

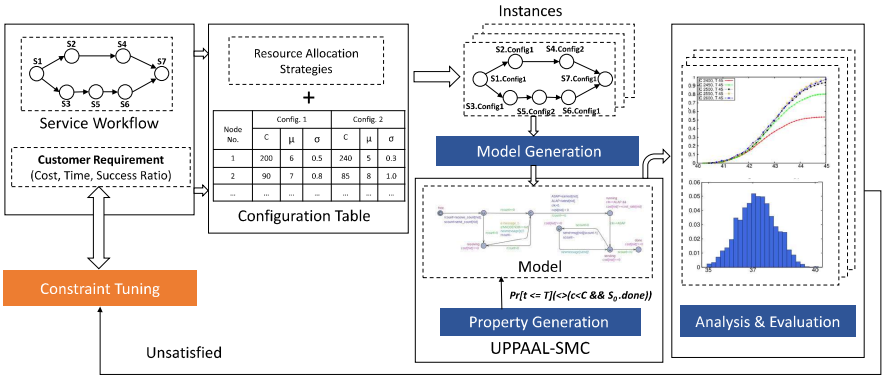

Fig. 2. Our resource allocation strategy evaluation and SLA negotiation framework

for different purposes (e.g., time optimization, cost optimization). By using our defined mapping rules, the generated resource allocation instances and user requirements can be automatically converted into NPTA models and property-based queries respectively. Based on the capability of UPPAAL-SMC model checker, our framework can automatically analyze and evaluate the generated resource allocation instances. If the customer requirement can be satisfied, the best resource allocation instance will be reported. Otherwise, if none of the resource allocation instances can meet the customer's requirement, the Cloud service provider needs to negotiate with the customer based on the evaluation results by tuning constraint parameters. Since the model and property generation as well as UPPAAL-SMC-based model checking can be fully automated, the evaluation process can be conducted without human intervention. The following sub-sections will introduce the major steps of our framework in details.

1) NPTA Model Generation: When analyzing a resource allocation instance, it needs to be automatically translated into an executable NPTA model first. To simplify the NPTA model construction, our approach decouples the syntax and semantics of a workflow with binded resources. In our approach, we divide the NPTA model for a service 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 describe both the concurrent semantics of workflows and the execution variation information. Such configuration information can be achieved from resource allocation instances automatically.

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. Based on the observation, 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 details (i.e., the execution time with variation for each services) of the corresponding resource allocation instance.

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-dimension array *distribution[N+1][2]*, which represents the expected time (saved in *distribution[i][0]*) and the standard deviation (saved in *distribution[i][1]*) for the allocated i_{th} service respectively. 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 price information for each service.

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 IV-B2), we added a dummy service with ID 0 in the workflow which merges all the final services.

To update *receive_count[N+1]* and *send_count[N+1]*, we adopt the broadcast synchronization to model the end-to-end message-passing mechanism 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 *receive_count* by 1.

When a service finishes, it will send notifications to all its successors. After sending a message, the sending service will decrease its *send_count* by 1. In the back-end configuration, we use a matrix (i.e., a two-dimensional array) *msg* to keep all the messages of the workflow, which implicitly represent the workflow edges. Instead of constructing a matrix of size $(N+1)\times(N+1)$, 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. After the completion, the i_{th} service will check *send_count[i]* and find the corresponding message entry $msg[i][]$ to notify its successors via broadcasting.

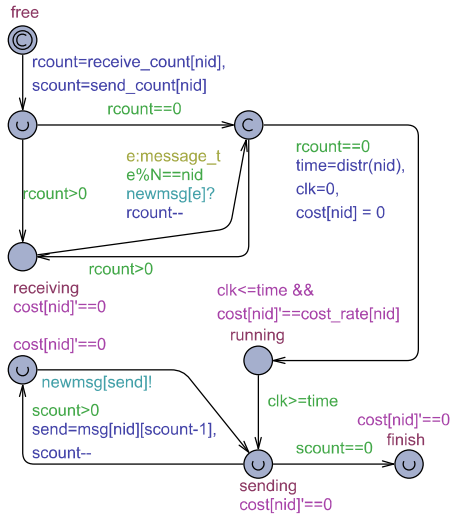

Fig. 3. Front-end model template for a workflow service

In our front-end model, 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 3 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 all the predecessor services are finished, and the current service is executing. In this state, the cost of the service will be calculated using the unit price $cost\_rate[nid]$. 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 3 and back-end configurations, the derived NPTA model from a resource allocation instance can exactly mimic the behavior workflows. Therefore, we can conduct the query of performance on it using the statistics-based approaches.

2) Property Generation: When NPTA models are generated from different resource allocation instances, 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 distribution of the probability of successful simulations will be reported to enable the quantitative analysis.

C. Resource Allocation Strategy Evaluation

Since cost, response time, and success ratio in the customer requirement are often conflicting with each other, it is very difficult to guarantee the optimality for all these three aspects at the same time, especially under the circumstance of service execution time variations. Given a specific requirement, Cloud service providers and customers may prefer to use different resource allocation strategies, since they do have different demands. For Cloud service 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 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 Cloud service providers to conduct the comparison among multiple resource allocation strategies. Therefore, for a given set of resource allocation strategies, the purposes of resource allocation strategy evaluation is to filter inferior ones and select an instance with the best performance.

To support Cloud service providers' decision making on resource allocation, our framework provides three kinds of resource allocation strategies by default: 1) time-constraint cost minimization (TCCM), 2) cost-constraint time minimization (CCTM), and 3) x_{th}-round feasible instance (xRFI). All these strategies produce resource allocation instances using the expected execution time. Note that the time variation information is considered in the evaluation stage rather than the stage of resource allocation instance generation. Here, TCCM is a strategy that searches for a cost optimal instance while the time constraint is satisfied, and CCTM is a strategy that searches for a time optimal instance while the cost constraint is not violated. Since TCCM and CCTM search for optimal solutions, they need to enumerate all the feasible solutions. Though a given 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 resource allocation instance encountered in the exhaustive enumeration. It is important to note that, due to the independence between generation and evaluation of resource allocation instances, other resource allocation strategies can be easily integrated into our framework.

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

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

V. Case Study

This section presents the evaluation details on a securities exchange workflow for the Chinese Shanghai A-Share Stock Market [21]. In the experiment, we use UPPAAL-SMC [7], [8] as the evaluation engine of our framework. During the evaluation using UPPAAL-SMC, we set the probability uncertainty (i.e., ε) to be 0.02, and set the probability of false negatives (i.e., α) to be 0.02. All the experiments were conducted on a machine with 2.8GHz Intel i7 CPU with 4 GB RAM.

A. Workflow Description

The securities exchange workflow is a typical instance-intensive workflow process which involves a large number of transactions and each of them is a relatively short workflow instance with only a few steps. Some steps of the workflow instance are executed concurrently. The example illustrated in Figure 4 is the securities exchange workflow for the Chinese Shanghai A-Share Stock Market [21].

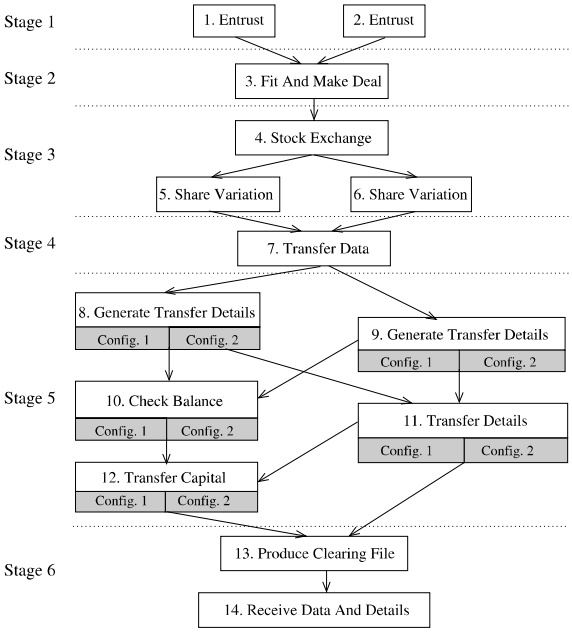

Fig. 4. A securities exchange workflow

The workflow shown in Figure 4 consists of 14 nodes which represent the major workflow activities. As introduced above, a securities exchange workflow is a typical instance and computation intensive process. Therefore, it is 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. Taking *Stage 5* of the securities exchange workflow as an example, nodes 8-12 are all provided with multiple options under different price. In this experiment, we assume that there are two options for all these services with different cost and duration. Table I shows the service configuration in details. For each node, we list two VM configurations including the unit price, mean service execution time and corresponding standard deviation. Intuitively, a faster and more stable VM has a higher price. It is important to note that in this experiment we focus on the evaluation of resource allocation strategies for the stage 5 in the securities exchange workflow. We only consider the time variation for the nodes 8-12. We assume that the other nodes in the workflow have fixed cost and execution time with no

variation. In next subsections, we will evaluate the different resource allocation strategies based on the above securities exchange workflow and corresponding configurations.

TABLE I
CONFIGURATIONS FOR THE SECURITIES EXCHANGE WORKFLOW

	Config. 1			Config. 2		
Node	Price	M.T.(μ)	S.D.(σ)	Price	M.T.(μ)	S.D.(σ)
8	70	5	0.4	50	6	0.7
9	45	10	0.8	60	8	0.5
10	40	10	0.5	30	12	0.8
11	100	7	0.6	80	8	0.9
12	60	12	0.4	40	15	0.7

B. Strategy Evaluation

In this example, we assume that the customer wants the 5_{th} step of the securities exchange workflow to be completed within 45 time units and 2550 cost units, and the success ratio to be no lower than 80%. We evaluate the securities exchange workflow using the self-contained default strategies (i.e., TCCM, CCTM, and xRFI) in our framework.

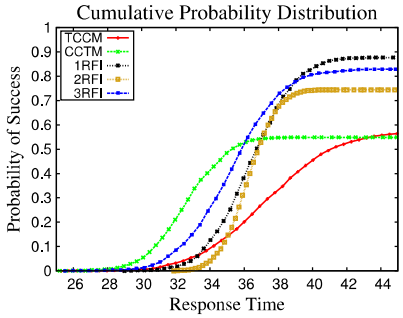

Fig. 5. CPD for R(2550, 45, 80%)

As soon as receiving a customer request, our framework will firstly apply the SRMS approach on the generated instances to check whether such requirement can be satisfied or not with our default strategies. Since we set x to be 3 for XRFI, we obtained 5 resource allocation instances using the strategies TCCM, CCTM, 1RFI, 2RFI, and 3RFI respectively. Figure 5 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 instances generated by TCCM, CCTM and 2RFI need to be discarded. It can be found that the success ratio of the 1RFI instance (i.e., probability of success has a confidence of 0.98 within [0.85, 0.89]) is higher than the success ratio of the 3RFI instance (i.e., probability of success has a confidence of 0.98 within [0.79, 0.83]). Due to the higher probability of success, the 1RFI instance will be selected.

To investigate the effects of different requirement constraint parameters, we applied the *MRMS* approach. Based on the example shown in Figure 5, we tune the cost (increased by 50 cost units) and time (increased by 3 time units) respectively as shown in Figure 6 and Figure 7. In Figure 6, due to the increase of price, the workflow can get better VMs in their resource allocation instances. Therefore, CCTM can achieve better success ratio. It is important to note, though the success ratio of 2RFI does not change at time 45 in Figure 6, the response time performance is improved. We can find that 2RFI can get a success ratio of 70% at time 35 in Figure 6, while 2RFI needs at least of a time of 38 to achieve such a ratio in Figure 5. From Figure 5 and Figure 6, we can find that increasing the cost by 50 does not make any obvious change for the instances TCCM, 1RFI and 3RFI, since the instances TCCM, 1RFI and 3RFI shown in Figure 5 and Figure 6 are the same. However, when the user relaxes the time limit to 48 as shown in Figure 7, we can see the significant increase of success ratio for the instances TCCM and 2RFI. This is because that the resource allocation instances TCCM and 2RFI in Figure 5 suffer from the tight response time constraints. In Figure 7, if customers care about short response time, then the 2RFI instance will be a better choice, since it can reach the required success ratio 80% within 35 time units, while the instance 1RFI, 3RFI and TCCM need around 39, 41 and 47 time units respectively. If the customer does not care more about the response time, due to the higher probability of success, the 1RFI instance will be selected.

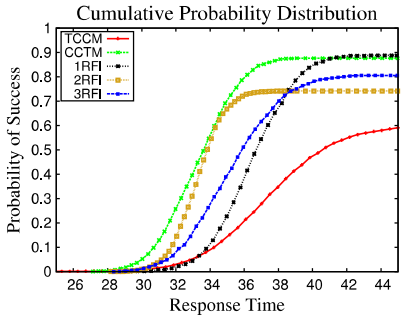

Fig. 6. CPD for R(2600, 45, 80%)

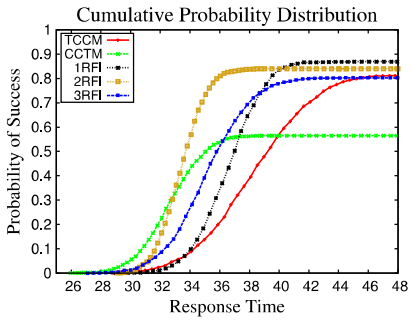

Fig. 7. CPD for R(2550, 48, 80%)

MRSS approach can be used to investigate the performance of single strategy under different requirements. Figure 8 and Figure 9 show the application of MRSS for both TCCM and CCTM strategies respectively. In Figure 8, the legend item *C 2550(2540), T 41(40)* means that the requirement provided by the customer is *R(2550, 41, -)*. But by using the TCCM strategy, we can get an optimal instance which needs 2540 cost units and 40 time units based on the expected

service response time in the workflow. For TCCM strategy, the minimum overall cost cannot be tuned. Therefore, in Figure 8, we create different requirements with the same cost but different response time. Similarly, for CCTM strategy, the minimum overall execution time cannot be tuned. Therefore, Figure 9 shows the results of different requirements with the same response time but different cost.

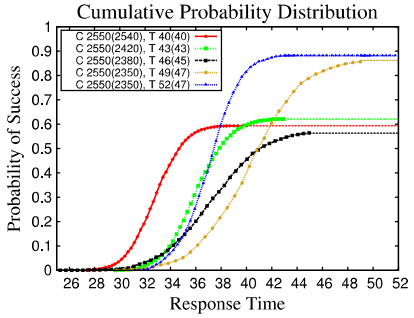

Fig. 8. CPD for TCCM Strategy

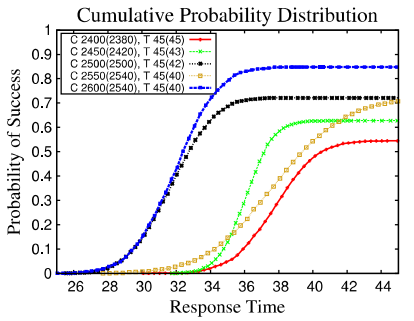

Fig. 9. CPD for CCTM Strategy

From Figure 8, we can find that when relaxing the time constraint, the expected cost can be reduced accordingly. However, this does not help the strategy to improve its probability of success drastically for the first 3 of 5 presented legend items. This is because that, when the time limit is smaller than or equal to 47, the expected time of each generated instances is almost same as the time limit, which strongly affects the success ratio due to the execution time variation. When the time limit is increased to 52 , we can get a resource allocation instance with an expected response time of 47. Due to the big gap between the time limit of the requirement and the expected response time, the success ratio can be drastically improved. From Figure 9 we can observe that the increase of cost can lead to the reduction of workflow response time. This is because more high-end VMs are used in the newly generated resource allocation instances.

VI. Conclusions

Due to the increasing demand of QoS in Cloud, variation-aware resource allocation is becoming an important issue. For Cloud service providers, an effective 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, there is no strategy that can always guarantee the best performance in practice. Therefore, to reduce the decision making efforts of Cloud service providers, it is necessary to develop an approach that can automatically compare and tune the performance of different resource allocation strategies. To address this issue, we propose an UPPAL-SMC-based evaluation framework. Our framework can not only support the complex QoS queries to filter inferior resource allocation solutions, but also enable the tuning of QoS constraints to achieve the required customer satisfaction. Comprehensive experimental results based on an industry example demonstrated the effectiveness of our framework.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica and M. Zaharia. *A View of Cloud Computing*. Communications of the ACM, 53(4): 50-58, 2010.

[2] 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, 63(1): 256-293, 2013.

[3] W. Chen and J. Zhang. *An Ant Colony Optimization Approach to a Grid Workflow Scheduling Problem With Various QoS Requirements*. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, 39(1): 29-43, 2009.

[4] S. Frey, F. Fittkau and W. Hasselbring. *Search-Based Genetic Optimization for Deployment and Reconfiguration of Software in the Cloud*. International Conference on Software Engineering (ICSE), 512-521, 2013.

[5] A. Gambi and G. Toffetti. *Modeling Cloud Performance with Kriging*. International Conference on Software Engineering (ICSE), 1439-1440, 2012.

[6] 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), 195-204, 2011.

[7] 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), 349–355, 2011.

[8] K G. Larsen. *Priced Timed Automata and Statistical Model Checking*. International Conference on Integrated Formal Methods (IFM), 154-161, 2013.

[9] K. Czajkowski, I. T. Foster, C. Kesselman. *Resource Co-Allocation in Computational Grids*. International Symposium on High Performance Distributed Computing (HPDC), 219 - 228, 1999.

[10] J. Broberg, S. Venugopal and R. Buyya. *Market-Oriented Grids and Utility Computing: The State-of-the-Art and Future Directions*. Journal of Grid Computing, 6(3): 255-276, 2008.

[11] F. I. Popovici and John Wilkes. *Profitable Services in An Uncertain World*. International Conference for High Performance Computing Networking, Storage, and Analysis (SC), 36, 2005.

[12] Y. Lee, C. Wang, A. Y. Zomaya and B. Zhou. *Service Level Agreement Based Distributed Resource Allocation for Streaming Hosting System*. International Conference on Cluster, Cloud and Grid Computing (CCGrid), 15-24, 2010.

[13] 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, 41(1): 23-50, 2011.

[14] Y. Song, Y. Li, H. Wang, Y. Zhang, B. Feng, H. Zang and Y. Sun. *A Service-Oriented Priority-Based Resource Scheduling Scheme for Virtualized Utility Computing*. International Conference on High Performance Computing (HiPC), 220-231, 2008.

[15] K. Sen, M. Viswanathan and G. Agha. *Statistical Model Checking of Black-Box Probabilistic Systems*. International Conference on Computer Aided Verification (CAV), 202-215, 2004.

[16] 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 (JPDC), 68(11): 1504-1516, 2008.

[17] A. David, D. Du, K. G. Larsen, A. Legay, M. Mikucionis, D. Poulsen and S. Sedwards. *Statistical Model Checking for Stochastic Hybrid Systems*. International Workshop on Hybrid Systems and Biology, 122-136, 2012.

[18] 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, 352-367, 2013.

[19] 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) NIER Track, 2014.

[20] M. A. Rodriguez and R. Buyya. *Deadline based Resource Provisioning and Scheduling Algorithm for Scientific Workflows on Clouds*. IEEE Transactions on Cloud Computing, 2(2): 222-235, 2014.

[21] Chinese Shanghai A-Share Stock Market. *http://www.sse.com.cn/sseportal/en/*.