Experience-Availability Analysis of Online Cloud Services using Stochastic Models

Yue Cao[1], Laiping Zhao[1*], Rongqi Zhang[2], Yanan Yang[1], Xiaobo Zhou[2], Keqiu Li[2]
[1]*School of Computer Software, Tianjin University, Tianjin, China*
[2]*Tianjin Key Laboratory of Advanced Networking, School of Computer Science and Technology, Tianjin University, Tianjin, China*
E-mail: {cy1994, laiping, zrq0312, ynyang, xiaobo.zhou, keqiu}@tju.edu.cn

***Abstract*—Recently a new performance metric called experience availability (EA) has been proposed to evaluate online cloud service in terms of both availability and response time. EA originates from the fact that from the prospective of quality of experience (QoE), an online cloud service is regarded as unavailable not only when it is inaccessible, but also when the tail latency is high. However, there still lacks analytic models for evaluating the EA of online services. In this paper, we propose an efficient EA-analytic model using stochastic reward net (SRN) to study the tail latency performance of online cloud services in the presence of failure-repair of the resources. Our EA-analytic model can predict the online service performance on EA, as well as support analysis on traditional availability and mean response time. We apply this model to an Apache Solr search service, and evaluate the prediction accuracy by comparing the results derived from the model to actual experimental results. It is shown that the proposed model overestimates the response time at lower percentiles and underestimates the response time at higher percentiles. Through attribution analysis, we further identify the list of factors that may affect the accuracy, and show that the 95^{th} percentile latency prediction error can be reduced to as low as 2.45% by tuning the configurations suggested by the attribution.**

***Keywords*—cloud computing, experience availability, online cloud service, stochastic reward net**

I. Introduction

Cloud computing is growing rapidly towards delivering computing as a public utility. Many services, including online web systems (e.g., social networking, e-commerce, search engine) and offline data-processing jobs (e.g., mapreduce, spark), are continuously deployed or processed in cloud systems [1, 2]. These services often consist of multiple tiers and tens or hundreds of tasks or micro-services, and need to handle unprecedented volumes of data. To characterize the performance of cloud service, an uptime-based *availability* measure was widely used [3], which is defined as

$$A = \frac{MTTF}{MTTF + MTTR}, \tag{1}$$

where MTTF and MTTR denote mean time to failure and mean time to repair, respectively. According to Equ. (1), availability describes only whether the service is accessible or not.

* Corresponding Author: Laiping Zhao, laiping@tju.edu.cn

In fact, from the prospective of quality of experience (QoE), the tail latency performance is at least as important as availability for a realtime online cloud service. Google's experiences on their back end services show that while majority of requests take around 50-60 ms, a fraction of requests takes longer than 100 ms, with the largest difference being almost 600 times [4]. One major reason of the performance uncertainty is due to the inevitable underlying competition on hardware resources among co-located services, resulting in the serious tail latency problem [5]. According to Nielsen [6], 0.1 second is about the limit for having the user feel that the system is reacting instantaneously; a response time of 1.0 second is about the limit for the user's flow of thought to stay uninterrupted, even though the user will notice the delay. For delays longer than 10 seconds, users are prone to perform other tasks while waiting. In this sense, *slow response* and *service unavailable* would be indistinguishable for cloud users [7].

Recently a new performance measure, named *Experience Availability* (EA), which is extended from the definition of *availability*, is proposed in [8]. It combines the traditional availability and tail latency for the first time into a single metric. According to EA, a service is *experience unavailable* not only because the service is failed, but also the γ^{th} latency exceeds a pre-defined threshold τ.

Currently, there still lacks analytic models and methodologies to analyze the EA of cloud services. According to its definition, we need to derive both the tail latency and availability to measure EA. Analyzing availability is relatively straightforward, and there are already a number of models proposed for availability modeling [9, 10]. However, calculating tail latency of online cloud services is a significant challenge due to the multi-tier architecture of cloud services. The errors could propagate across tiers and have cascading effects on overall latency distribution. Even worse, the majority of existing work focus on evaluating mean performance metrics, such as average response time, average resource utilization [11], while there are only a few of them considering to model and analyze the distribution of latencies for online services.

In this paper, we propose an efficient analytic model for analyzing the EA of online cloud services. We consider the common online search service, and use a stochastic reward net (SRN) to describe the interactions between its multi-tiers.

ISBN 978-3-903176-08-9

Through the model analysis, we can predict the service performance on EA. By comparing the prediction results to the real experimental results, we find that the proposed model shows some error on tail latency. We then give an attribution analysis on the system behavior, and find the potential factors that may affect the accuracy. The contributions can be summarized as follows:

- We design a SRN model to characterize the request response process of online cloud services. In this model, we study the mean response time of online cloud services in the presence of failure-repair of the resources.
- We propose a tagged customer model to analyze the cumulative distribution function (CDF) of response time for online cloud services. Based on the CDF, we can derive the EA of online cloud services.
- We demonstrate the accuracy of the model by comparing the analytical results to the real experimental results. It shows that the proposed model overestimates the response time at lower percentiles and underestimates the response time at higher percentiles. We further conduct experiments to identify a list of factors, including cache, number of keywords in search space, turbo boost and DVFS governor, that may affect the accuracy. It is found that by turning off the cache, increasing the search space, turning off Turbo Boost and configuring the DVFS performance governor, the prediction error can be reduced to as low as 2.45%.

The rest of the paper is organized as follows. Section II describes the basic architecture of online search service and our design of the SRN model. We present the EA-analytic model and introduce the model solving method in section III. Section IV describes the experimental evaluation of our model for online search service. In section V, we summarize the related work. We conclude and discuss future work in section VI.

II. Search Service and the SRN Model

In this section, we firstly introduce the general architecture of online search service. Then, we show how to construct the SRN model for the online search service.

A. Online Service Architecture

An online search service is a software system that is designed to search for information on the World Wide Web. Generally it consists of three main components [12]: the web crawler, the index generator, and the search engine, as shown in Fig. 1. The web crawler crawls some of the reachable web pages from site to site. The index generator associates keywords found on these web pages to their names of sites containing the keywords. It uses an update handler to process all updates, and generates distributed indexes. The indexed information is stored in database, and made available for search queries. The search engine will accept user's search requests, support text analysis, and generate the web pages list results by searching indexes. During this process, every page in the entire list must be weighted according to information in the indexes.

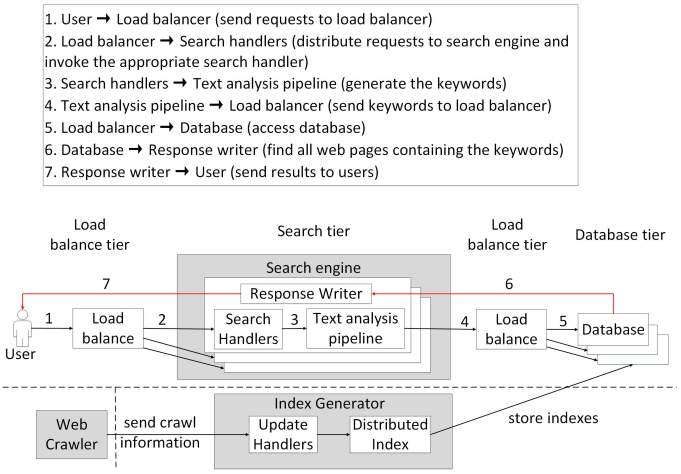


Fig. 1. General architecture of the online search service.

Since a user's perception of the latency is mainly affected by the search engine, we hereafter merely discuss the modeling of the request processing in search engine. As shown in Fig. 1, a typical infrastructure for supporting a search component mainly includes four tiers: search tier, database tier and two load balancer tiers. We do not take into account the two load balancer tiers because their processing time is negligible compared to the search tier and the database tier in our setup. Depending on the number of users, each tier can be implemented with multiple instances, which are hosted across different virtual machines (VMs) in cloud platform. If there exists more than one instance at a tier, it is required to deploy another load balance tier to distribute the customer requests among instances.

Each instance, either the search or the database, maintains a thread pool for accepting requests. When a search request arrives, it first waits in the searching queue. The load balancer reads the search queue, and dispatches the request to a specific instance of search tier for text analysis. The corresponding instance then allocates one connection thread from thread pool to the request, and the connection will be occupied until user receives search results. After getting the keywords by text analysis, the request is further forwarded to an instance of database. Then, the database instance also allocates a thread to it for searching all web pages containing the keywords. Then, the response writer in the search instance constructs a ranked list of web pages, and sends the results back to user.

B. SRN Model

According to the architecture of online search service shown in Fig. 1, we construct a SRN model to analyze the interactions between multi-tiers. SRN is scalable to model systems composed of thousands of resources and is flexible to represent different policies and strategies [13].

We assume that the times assigned to all timed transitions conforms to an exponential distribution between, following

the common assumptions in [11, 14]. We consider crash failures occurs in an instance, i.e., an instance of search or database could fail with probabilities, resulting in the lost of all connections running on the instance. If a request's connection is lost due to the instance failures, we think that its response time is infinity. The input parameters required in the SRN model include: (1) request arrival rate (denoted by λ); (2) queue sizes of search and database (denoted by M_s and M_{db}); (3) service rates of a search instance or database instance (denoted by μ_s and μ_{db}); (4) the maximum number of connections supported by search tier and database tier (denoted by N_s and N_{db}); (5) failure rates of search and database (denoted by ϕ_s and ϕ_{db}); (6) repair rates of search and database (denoted by δ_s and δ_{db}).

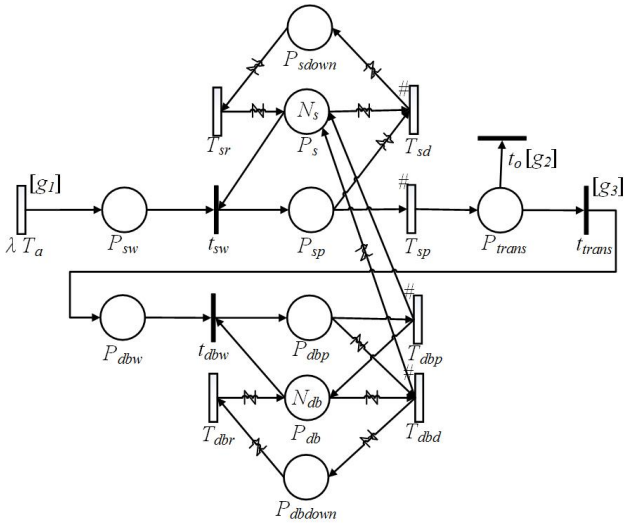


Fig. 2. SRN model of the online search service.

TABLE I
GUARD FUNCTIONS OF THE SRN MODEL

Transition	Guard Function
g1	$\#P_{sw} < M_s$? 1 : 0
g2	$\#P_{dbw} \geq M_{db}$? 1: 0
g3	$\#P_{dbw} < M_{db}$? 1 : 0

Fig. 2 describes the design of the SRN model for online search service. We only consider the two main tiers that directly affect the user's response time in the model: place P_{sw} to transition T_{sp} represents the processing in the search tier; place P_{dbw} to transition T_{dbp} represents processing in the database tier. Components such as search handlers are not studied separately because their processings have been incorporated into their corresponding tiers. Timed transition T_a represents the request arrivals to the system. Places P_{sw} and P_{dbw} represent the waiting queues of the search tier and database tier, and T_a fires only if the queue of search tier is not yet full (following guard function g_1). Once transition T_a (t_{trans}) fires, a token is deposited in place P_{sw} (P_{dbw}) showing that a request has been submitted to search tier (database tier) and it is waiting for processing from search tier (database tier). Place P_s and place P_{db} represent the remaining number of resources (i.e., connection threads) supported by search and database tier, and they are initialized with N_s and N_{db}. If there is a token in place P_{sw} and there is at least one token in place P_s, then one token from P_{sw} together with another token from P_s is removed respectively, and a token is put in place P_{sp}, representing that a request is ready for processing by search tier. Places P_{sp} and P_{dbp} represent the processing queues of search tier and database tier. The pound # in the arc from place P_{sp} to the transition T_{sp} shows that the actual firing rate of transition is marking-dependent. Thus, the actual firing rate of transition T_{sp} is calculated as $K\mu_s$, where K is the number of tokens in place P_{sp}. After that, a token is removed from place P_{sp} and deposited into place P_{trans}, which represents that the request has been processed and leaves from search tier. Transition t_{trans} represents that the request moves from search tier to database tier. Transition t_o shows that the request gets dropped from the system and it fires only when the queue of database tier is already full (following guard function g_2). And then, the request is inserted into the waiting queue of database (i.e., place P_{dbw}) following guard function g_3. At the database tier, likewise, the request is processed only if database has available resources. After firing the timed transition T_{dbp}, a token is removed from place P_{dbp}, while one token deposited into place P_{db} and another token deposited into place P_s, showing that the request is finished and the corresponding connections at search tier and database tier are both released.

In the SRN model, we also model and analyze the impact on response time by server failures. We use place P_{sdown} and P_{dbdown} to represent the number of lost connections in search tier and database tier, respectively. If an search instance fails, all working connections in P_{sp} and idle connections in P_s running on the instance are lost simultaneously. Transition T_{sd} represents the failure occurs in search instances. The zigzag line on an arc represents that the arc can transfer multiple tokens at once. Suppose there are I_s instances at search tier and I_{db} instances at database tier. Since the requests are distributed evenly among the instances, a fraction of $1/I_s$ tokens in place P_{sp} together with another fraction of $1/I_s$ tokens in place P_s are removed respectively, and the sum of these tokens (N_s/I_s) is put in place P_{sdown}, representing that failed connections are ready for recovering. Once transition T_{sr} fires, the number of tokens (N_s/I_s) are removed from P_{sdown} and deposited in place P_s, indicating that the lost connections are recovered. The same process also applies to the database tier, except that the fraction of $1/I_{db}$ tokens are also deposited in place P_s after transition T_{dbd} fires. It means that the affected connections at search tier are also released when a database instance fails. The guard functions of the SRN model is shown in Tab. I.

C. Mean Response Time under Steady State

Given the SRN model above, we can derive the mean response time characterizing the system behavior by defining reward functions. To analyze the mean response time, we

first need to derive the mean number of waiting requests and blocking probability of requests.

1) Mean number of waiting requests: The mean number of waiting requests is given by mean number of tokens in place P_{sw} in Fig. 2, and it can be represented by $E[\#P_{sw}]$.

2) Blocking probability of requests: The steady-state blocking probabilities of requests in Fig. 2, P_b, can be calculated by assigning the following reward to the SRN model,

$$r_i = \begin{cases} 1, if[\#P_{sw}] \geqslant M_s, \\ 0, \text{otherwise.} \end{cases} \tag{2}$$

where M_s is the maximum length of waiting queue.

3) Mean response time: Mean response time is defined as the mean time from a customer request is entered the system until its leave. According to Fig. 2, the mean response time is the time from the instant that a token is deposited into P_{sw} until it is removed from P_{dbp}. Using Little's law, the mean response time (denoted by E_t) for requests at steady-state can be calculated as follows,

$$E_t = \frac{(E[\#P_{sw}] + E[\#P_{dbw}] + E[\#P_{sp}] + E[\#P_{dbp}])}{(1 - P_b) \times \lambda} \tag{3}$$

where $E[\#P_x]$ is the mean number of tokens in place P_x at steady-state and (1 - P_b) $\times$ λ is the effective request arrival rate in the online search service system.

III. Experience Availability Measure

In this section, we show how to model and predict the EA of the online search service. Generally, it takes three steps for calculating EA: (1) Divide the total operational time *T* into *n* time slices. (2) In each time slice, derive the CDF of response time while taking into account instance failures. (3) Derive EA based on the CDF of response times in all time slices.

A. Tagged Customer Model

To derive the CDF of response time, we propose the tagged customer model [15] by modifying the SRN model in order to track the tagged customers movements through the system.

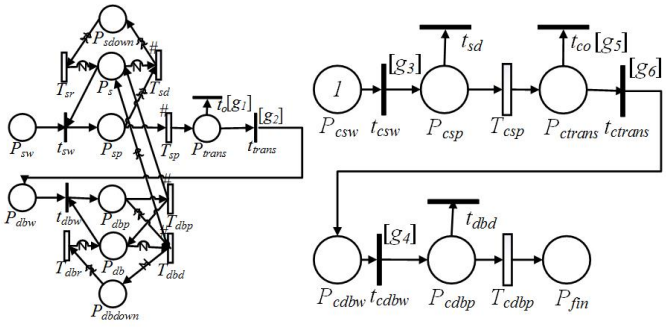


Fig. 3. Tagged customer model.

We show the design of the tagged customer model in Fig. 3. Place P_{csw} contains a single token that represents the arrival of a request. The *m*, *n*, *i*, *j*, *a*, *b* and *p*, *q* tokens initially presented in places P_{sw}, P_{dbw}, P_{sp}, P_{dbp}, P_{sdown}, P_{dbdown} and P_s, P_{db}, represent the corresponding system status before the request arrival. That is, at the time of the request arrival, there are *m* (n) requests waiting in the search (database) queue, i (j) requests being processing by search (database), a (b) connections lost in P_{sdown} (P_{dbdown}), and p (q) connection threads in the thread pool still available for accepting new requests. Transition t_{csw} is fired only when place P_{sw} is empty and the place P_s is not empty. Transition t_{cdbw} is fired only when place P_{dbw} is empty and the place P_{db} is not empty. The guard functions are shown in Tab. II.

TABLE II
Guard Function of the Tagged Customer Model in Fig. 3

Transition	Guard Function
g1	$\#P_{dbw} \geq M_{db}$? 1 : 0
g2	$\#P_{dbw} < M_{db}$? 1 : 0
g3	$\#P_s > 0$ and $\#P_{sw} == 0$? 1 : 0
g4	$\#P_{db} > 0$ and $\#P_{dbw} == 0$? 1 : 0
g5	$\#P_{dbw} \geq M_{db}$? 1 : 0
g6	$\#P_{dbw} < M_{db}$? 1 : 0

B. Response Time Distribution Calculation

Now, we can derive the response time CDF using the tagged customer model. We define the set of initial system states as follows:

$$T = [m, n, i, j, a, b, p, q] \tag{4}$$

where,

$$m \in [0, ..., M_s], n \in [0, ..., M_{db}], i \in [0, ..., N_s], j \in [0, ..., N_{db}],$$
$$a \in [0, ..., N_s], b \in [0, ..., N_{db}], p \in [0, ..., N_s], q \in [0, ..., N_{db}],$$
$$i + a + p = N_s, j + b + q = N_{db}. \tag{5}$$

Denoted by π_x the probability that the system stays in $\forall x \in T$ under steady state. Then π_x can be derived using the SRN model proposed in Section II. Clearly, we have,

$$\sum_{x \in \tau} \pi_x = 1 \tag{6}$$

In the tagged customer model, a non-empty place P_{fin} means that the processing of the tagged request has been completed. Thus, we define the absorbing state for the tagged customer as follows,

$$r_x(t) = \begin{cases} 1, if(\#(P_{fin}), t) > 0, \\ 0, \text{otherwise.} \end{cases} \tag{7}$$

where the reward function $r_x(t)$ is denoted whether an absorbing marking has been reached at time *t*.

By solving the reward function $r_x(t)$, we can obtain the probability $R_x(t)$ that the tagged customer request is absorbed at time t under initial marking x. Then, the probability that a request processing is completed at time t can be calculated as follows:

$$R(t) = \sum_{x \in \tau} [R_x(t) \times \pi_x] \tag{8}$$

Given the $R(t)$ and the pre-defined γ and τ, we can easily know if the γ^{th} percentile latency exceed τ or not in the current time slice. Then, it is straightforward to calculate the EA using Equ. (9),

$$EA(\tau,\gamma) = \frac{\sum_{i=1}^{m_i} t_{TTHTL_i(\tau,\gamma)}}{\sum_{i=1}^{m_i} t_{TTHTL_i(\tau,\gamma)} + \sum_{j=1}^{n_i} t_{TTLTL(\tau,\gamma)}} \quad (9)$$

where $t_{TTHTL_i(\tau,\gamma)}$ represents the time to high tail latency, that is, the time whose γ^{th} percentile latency is less or equal to τ: $TL(\gamma) \leq \tau$, and $t_{TTLTL_j(\tau,\gamma)}$ represents the time to low tail latency, that is, the time with $TL(\gamma) > \tau$.

IV. Experimental Evaluation

In this section, we evaluate the proposed SRN model by comparing its results with the actual experimental results. We then give an in-depth analysis on the prediction error, and list the factors that may affect the accuracy of the model.

A. Experiment Setup

We use the Stochastic Petri Net Package (SPNP) [16] to solve the proposed model. We also implement a real Apache Solr [17] search service to record and analyze the actual tail latency experienced by users. All input parameters to the SRN model are extracted from the testing of the Apache Solr service. The proposed SRN model does not take into account the two load balancers because their processing time is negligible compared to the application and database tiers in our setup.

For the real Apache Solr service, we deploy three instances of SolrCloud search engine, which is implemented using Tomcat for accepting users' search requests, supporting text analysis and interacting with database tier. Each instance of the SolrCloud supports a limited number of connections. When a request arrives at the server, a separate available connection thread will be allocated to the request. Likewise, we also deploy three instances of MySQL database to store the indexes generated by crawler and indexing component, and each database instance is configured with a limited maximum number of connected clients. There may be more than three instances in real system. Our model can still work by simply changing the number of instances in the configurations. However, the runtime of model will increase over the number of instances, and studying the scalability is left as future work. In addition, either the SolrCloud instance or the MySQL instance probably fails following some pre-configured failure rates. To demonstrate the ability of our model, we artificially set a relative high failure rate for both SolrCloud and MySQL instances in the SRN model. We also injected failures into the real system by powering off instances at the same rate. If an instance fails, all connections running on it will be lost simultaneously. The connections will not be recovered until the instance is repaired.

Search requests are automatically generated by Apache Jmeter [18]. It randomly selects a keyword from a pre-configured search space, and sends requests to Solr following a Poisson distribution with the arrival rate λ. The search space is initially configured with 3,000 keywords. We totally send 1 million search requests to Solr service. Then Jmeter records all response times of the requests. We deploy the Apache Solr service on machines configured with Intel Core i5-4670 processor, and 8GB RAM. To estimate the service rate of both search threads and database threads, we use Jmeter to access Solr and Mysql separately, and take the average of ten repeated operations as the final service rate. Tab. III lists all parameter configurations used in our model and actual experiments.

TABLE III
Configuration Parameters

Parameter	Values		
Max # of connections in search tier (N_s)	9	30	45
Max # of connections in database tier (N_{db})	9	30	45
Waiting queue size in search tier (M_s)	30		
Waiting queue size in database tier (M_{db})	30		
Request arrival rate (λ)	100	400	900
Service rate of a search thread (μ_s)	56		
Service rate of a database thread (μ_{db})	32		
Failure rate of a search instance (ϕ_s)	0.100	0.125	0.330
Failure rate of a database instance (ϕ_{db})	0.100	0.125	0.330
Repair rate of a search instance (δ_s)	0.100		
Repair rate of a database instance(δ_{db})	0.100		

B. Results

1) Mean Response Time: We first evaluate the mean response time of the Solr service using the configuration parameters listed in Tab. III.

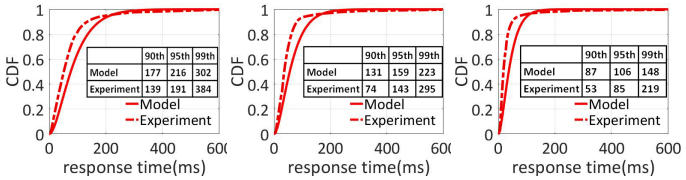
TABLE IV
Mean Response Time

Configurations					Mean response time (ms)		
N_s	N_{db}	λ	ϕ_s	ϕ_{db}	Model	Experiment	Error
9	9	400	0.125	0.125	89.52	74.49	20.18%
30	30	400	0.125	0.125	57.93	49.31	17.48%
45	45	400	0.125	0.125	40.16	33.97	18.22%
30	30	100	0.125	0.125	43.29	37.15	16.53%
30	30	900	0.125	0.125	97.58	82.83	17.81%
30	30	400	0.330	0.330	68.32	57.14	19.57%
30	30	400	0.100	0.100	51.86	43.79	18.43%

Tab. IV shows the SRN model prediction results and the actual experimental results. Fixing the request arrival rate λ at 400 and failure rates ϕ_s and ϕ_{db} at 0.125, the response times of both the SRN model and actual experiments decrease as the number of maximum supported connections increase from 9 to 45. While setting $N_a = N_{db} = 30$, the response time increase significantly as the job arrival rate increase from 100 to 900.

Moreover, the response time decrease if we reduce the failure rate of both search instance and MySQL instance. We see that the relative error on mean response time is generally within 20.18%, and the SRN model results on mean response time are mostly larger than the actual experimental results. This is primarily because, there are some acceleration techniques adopted by either the hardware layer or software layer, for example, CPU pipeline, multiple instruction issue, cache, our SRN model is unable to analyze these techniques, while they indeed reduce the response time significantly in the actual experiments.

2) Response Time CDF: By changing the maximum number of connections (N_s and N_{db}), request arrival rate (λ), and failure rate (ϕ_s and ϕ_{db}), respectively, we conduct three groups of experiments to evaluate the cumulative distribution function (CDF) of the response time.

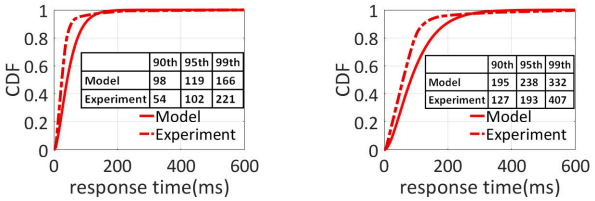


(a) N_s = 9, N_{db} = 9 (b) N_s = 30, N_{db} = 30 (c) N_s = 45, N_{db} = 45

Fig. 4. Response time CDF when changing the max. number of connections.

The first experiment evaluates the response time CDF under settings with different maximum number of connections (N_s and N_{db}). For the other parameters, we set $\lambda = 400$, $\phi_s = \phi_{db} = 0.125$. Fig. 4 shows the response time CDF and the 90^{th}, 95^{th}, 99^{th} percentile latency, when N_s and N_{db} are set to 9, 30 and 45, respectively. We find that increasing the number of connections would reduce the 90^{th}, 95^{th}, 99^{th} percentile latency in both model analysis and actual experiments. On the one hand, our model overestimates the response time at lower percentile due to the lack of consideration on accelerating technologies such as cache, CPU pipeline, multiple instruction issue etc. On the other hand, our model underestimates the response time at higher percentile (99^{th}). This is because, there are many processes, from either other instances/applications or operating system processes, running on the same underlying hardware simultaneously, yet the hardware does not support performance isolation between these processes, resulting in resource contention and disorder. Hence, the higher percentile latency in actual experimental is usually much larger than model results.

The second experiment evaluates the response time CDF under settings with different request arrival rate (λ). Fig. 5 shows the response time CDF and the 90^{th}, 95^{th}, 99^{th} percentile latency, when $N_s = N_{db} = 30$ and λ is set to 100 and 900, respectively. We see that the 90^{th}, 95^{th}, 99^{th} percentile latency all increases as we increase the arrival rate. Like the first experiment, our model overestimates the response time at lower percentile, and underestimates the response time at higher percentile (99^{th}). The two lines cross at around the 96^{th} percentile.

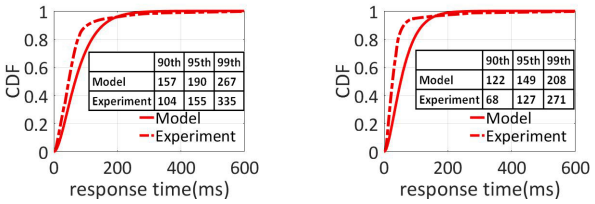


(a) λ = 100 (b) λ= 900

Fig. 5. Response time CDF under different request arrival rates.

TABLE V
EXPERIENCE AVAILABILITY

Configurations					Results			
N_s	N_{db}	λ	ϕ_s	ϕ_{db}	250ms percentile		EA	A
					Model	Experi.		
9	9	400	0.125	0.125	97.3%	96.4%	70%	96.3%
30	30	400	0.125	0.125	99.5%	98.9%	85%	96.5%
45	45	400	0.125	0.125	99.9%	99.9%	95%	97.0%
30	30	100	0.125	0.125	99.9%	99.9%	95%	96.3%
30	30	900	0.125	0.125	95.9%	96.3%	80%	96.2%
30	30	400	0.330	0.330	98.6%	97.4%	75%	90.7%
30	30	400	0.100	0.100	99.7%	99.1%	90%	97.1%



(a) ϕ_s = 0.33, ϕ_{db} = 0.33 (b) ϕ_s = 0.1, ϕ_{db} = 0.1

Fig. 6. Response time CDF under different failure rate.

The third experiment evaluates the response time CDF under settings with different failure rate (ϕ_s and ϕ_{db}). Fig. 6 shows the response time CDF and the 90^{th}, 95^{th}, 99^{th} percentile latency, when ϕ_s and ϕ_{db} are set to 0.1 and 0.33, respectively. When failures are injected to search and database instances, all the affected connections will be lost, and their response times become infinity. Moreover, since the maximum number of connections supported by search tier and database tier is reduced due to the failures, the newly arrived requests more probably wait in the queue, or even are rejected if the waiting queue becomes full. Thus, increasing the failure rate would increase the response time.

TABLE VI
ERROR ATTRIBUTION FACTORS.

Factor	Level 1	Level 2
Cache	off	on
Number of Keywords	8000	3000
Turbo Boost	off	on
DVFS Governor	performance	ondemand

3) Experience Availability Calculation: Suppose the user's QoE requirement on tail latency is defined as: in each time slice, the 99^{th} percentile latency should be less than 250ms. Tab. V shows the percentiles at the response time of 250ms in different configurations, calculated by solving the SRN model and actual experiments, respectively. We see that the percentiles calculated by SRN model is larger than that in the actual experiment. By dividing the 1 million requests into 20 equal-time-intervals, we calculate a statistical analysis of the response times for each time slice, and calculate the EA according to Equ. (9). We find that, even the arrival rates are the same in all time slices, but the response time we measure is changing across runs, resulting in different percentiles at 250ms. This is due to the underlying hardware contention interferences from other processes in the same server. We also calculate the traditional availability according to Equ. (1). Clearly, its values are generally much larger than EA, implying that a service may be experience-unavailable even when it is available.

C. Error Attribution

The analysis has shown the possible prediction error by our model. To reduce the prediction error, we need to identify the potential factors that may affect the accuracy. In this section, we list all the factors we suspect to have an impact on the response time, and evaluate their impact on the response time.

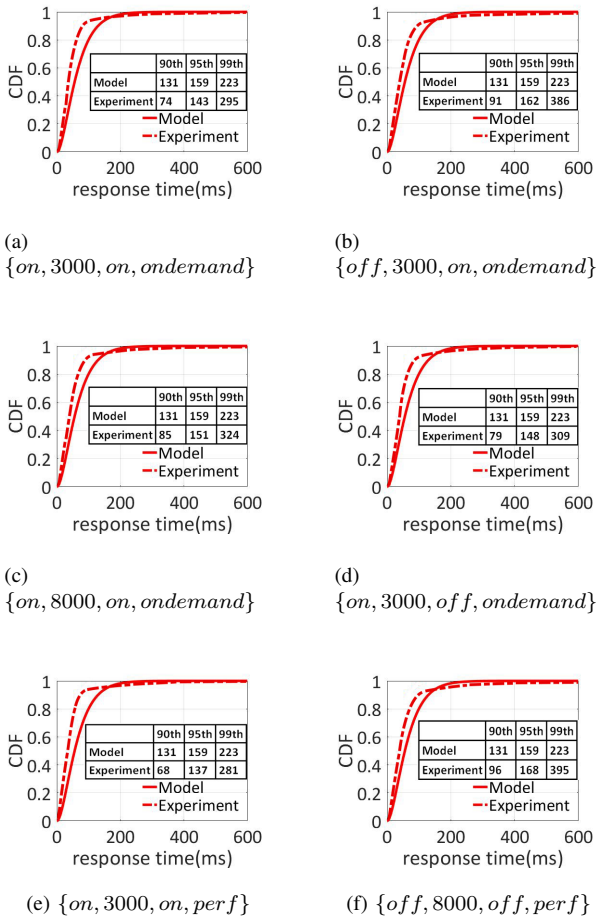
1) Cache: The cache includes CPU cache and page cache. CPU cache is used by the CPU of a computer to reduce the average time to access data from the main memory. The page cache is kept by the operating system to speed-up the access to the contents of cached pages and improve the overall performance.

2) Number of keywords in search space: Since Jmeter randomly selects keywords from the search space in each request, a small search space could lead to high frequent repeated requests. A repeated request could be completed by the cache not only in the service side, but also in the client.

3) Turbo boost: Turbo boost is a feature implemented on many modern processors, which automatically raises the processors' operating frequency, and thus performance, depending on the task demand and dynamic power.

4) DVFS Governor: Dynamic voltage and frequency scaling is a power management technique in CPU. It allows the operating system to dynamically adjust the CPU frequency to boost performance or save power.

Tab. VI lists all possible configurations of the factors, which are divided into two levels, where level 1 indicates {*cache off, 8000 keywords, turbo boost off, DVFS governor performance*}, and level 2 indicates {*cache on, 3000 keywords, turbo boost on, DVFS governor ondemand*}. We conduct another group of experiments to evaluate the impact of these factors on the model prediction accuracy as shown in Fig. 7. We find that, setting all the four factors at level 1 could help to reduce the prediction error on mean response time, 90^{th} and 95^{th} percentile latency. In particular, as shown in Tab. VII, the prediction error on mean response time could be reduced to as low as 7.12%, the prediction error on 95^{th} percentile latency could be reduced to 2.45%. Among the four factors, we find that turning off the cache feature is the most effective way to reduce the error, the next is increasing the number of keywords, turbo boost and DVFS governor make little effect on the results. For the 99^{th} percentile latency, we see that the configuration of level 1 actually increase the prediction error to 43.54%. This is because the interferences coming from hardware resource contention still exists, and we are not able to eliminate their impact through simple configurations.



(a) $\{on, 3000, on, ondemand\}$ (b) $\{off, 3000, on, ondemand\}$

(c) $\{on, 8000, on, ondemand\}$ (d) $\{on, 3000, off, ondemand\}$

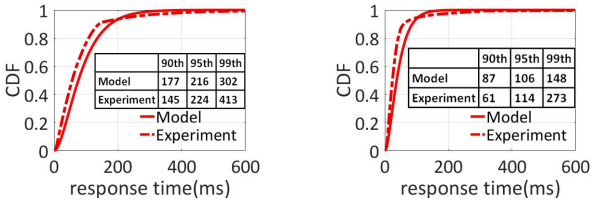
(e) $\{on, 3000, on, perf\}$ (f) $\{off, 8000, off, perf\}$

Fig. 7. Response time CDF under different configurations in Tab. VI.

Fig. 8 shows the response time CDF under the settings of $N_s = N_{db} = 9$ and $N_s = N_{db} = 45$, respectively, after we change all the factors into level 1. Compared with the results in Fig. 4, when $N_s = N_{db} = 9$, the prediction error on mean

TABLE VII
ERROR ANALYSIS CORRESPONDING TO FIG. 7.

Results		Mean Time		Tail Latency (ms)					
				90^{th}		95^{th}		99^{th}	
Model		57.93		131		159		223	
Experiment		Value	Error	Value	Error	Value	Error	Value	Error
	a	49.31	17.48%	74	77.03%	143	11.19%	295	24.41%
	b	53.47	8.34%	91	43.96%	163	2.45%	386	42.23%
	c	52.81	9.70%	85	54.12%	151	5.30%	324	31.17%
	d	50.36	15.03%	79	65.82%	148	7.43%	309	27.83%
	e	48.25	20.06%	68	92.65%	137	16.06%	281	20.64%
	f	54.08	7.12%	96	36.46%	168	5.36%	395	43.54%



(a) N_s = 9, N_{db} = 9 (b) N_s = 45, N_{db} = 45

Fig. 8. Response time CDF for different resource number.

response time is reduced to 9.79%, and the 95^{th} percentile latency is reduced to 3.57 %. When $N_s = N_{db} = 45$, the prediction error on mean response time is reduced to 8.47%, and the 95^{th} percentile latency is reduced to 7.02%.

V. RELATED WORK

A. Availability Evaluation

There has been a large amount of work on evaluating the availability of cloud services, which can be categorized into model based methods and measurement based methods.

In model based methods, three types of widely used models are combinatorial models, state-space models and hierarchical models. Combinatorial models include Reliability Block Diagrams (RBD) and fault tree analysis. RBD are proposed to model the availability of virtual data centers (VDCs) [19] and fault tree [20] has been used to evaluate the reliability of multi-nodes SDDC (software-defined data center). Although it is easy to implement combinatorial model due to its explicit presentation, it cannot model large and complicated systems.

State-space models mainly include Markov chain, semi-Markov processes, stochastic petri net (SPN) or stochastic reward net (SRN) [21] . Wu et al. [22] presented a stochastic method based on semi-Markov model to evaluate the availability of Infrastructure-as-a-Service (IaaS) cloud. Longo et al. [23] presented an SRN model to analyze the availability of a large-scale IaaS cloud. State-space models are capable of modeling large and complicated systems. However, it is impractical to use a single state space model to model the whole system due to the state space explosion problem.

Hierarchical models combine the combinatorial models and state-space models to evaluate the availability of large-scale systems. Wei et al. [24] constructs a hybird dependability model based on RBD and GSPN to model the virtual data center of cloud computing. Dantas et al. [25] combined RBDs and Markov models to analyze availability of Eucalyptus architecture. Because hierarchical model is decomposable, it solves the problem of state space explosion. However, the decomposability is not always manually controllable due to the automaticity of the model generation.

Besides the model based methods, there are a lot of measurement based methods to evaluate the availability. Fujita et al. [26] developed DS-Bench toolset to evaluate the dependability of a cluster of physical machines and a cloud computing environment. Sangroya et al. [27, 28] proposed a MapReduce benchmark suite to estimate the dependability and performance of MapReduce systems. Furthermore, there are also a number of prior works review the performance benchmarking for IaaS cloud [29, 30]. However, standard benchmarking solutions cannot be used directly for the prediction of cloud availability.

B. Tail Latency Evaluation

Tail latency, or response time, is another important metric reflecting the quality of user experience for online cloud services. Most existing work focus on evaluating the mean response time instead of the response time CDF [31, 32]. However, mean response time is far from sufficient to describe the user experience.

Generally, the response time consists of queuing time and service time. To evaluate waiting time, Sakuma et al. [33], construct a M/M/s queue model to analyze the tail approximation of the waiting time distribution of both patient and impatient customers. Bruneo et al. [13] also propose a tagged customer model based on SRN to calculate the waiting time distribution. The waiting time CDF is calculated by the probability that a tagged customer's request is absorbed and the probability of its corresponding initial state. However, the two above methods only calculate the waiting time CDF instead of the total response time CDF.

Muppala et al. [15] propose a tagged customer methods based on SRN to evaluate the response time CDF. Their model can apply to the closed queuing system with a fixed number of customers. However, most online cloud services are built on an open architecture and it can response to an arbitrary number of customer requests at the actual arrival rate. Grottke et al. [34] analyze the response time CDF using an open queuing network model. However, it is not easy to construct and solve the model due to the state space explosion problem.

VI. CONCLUSION

In this paper, we design an effective tagged customer model based on SRN to study the tail latency performance of online cloud services in the presence of failure-repair of the resources. In our method, the tagged customer model was used to analyze the CDF of online cloud service and predict the tail latency at any percentile. By solving the tagged customer model, we also can calculate the EA of online cloud services. We

conduct experiments by changing environment settings and compare model results to real experimental results to verify the accuracy of the proposed model. Experimental results show that the model results generally overestimates the response time at lower percentiles and underestimates the response time at higher percentiles. We also identified the potential factors that may impact the accuracy of online cloud services. It was found that by turning off the cache, increasing the search space, turning off Turbo Boost and configuring the DVFS performance governor, the prediction error can be reduced to as low as 2.45%. Taking these potential factor into account and modify our model accordingly may further reduce the prediction error, this is left as a future study.

In order to simulate the real utilization of the system whose request rate may vary significantly over time, we will further improve our model by adopting a Markov Modulated Poisson Process (MMPP) in the future.

ACKNOWLEDGMENT

This work is supported by the National Key Research and Development Program of China under Grant No. 2016YFB1000205, the National Natural Science Foundation of China under Grant No. 61402325. We would like to thank Binlei Cai, Zhuoxiao Zhang for their contributions. We would also like to thank Zitong Ji for the discussions on the initial drafts of this paper.

REFERENCES

[1] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[2] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, p. 4, 2008.

[3] I. Neamtiu and T. Dumitraş, "Cloud software upgrades: Challenges and opportunities," in *Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA)*, 2011, pp. 1–10.

[4] D. Krushevskaja and M. Sandler, "Understanding latency variations of black box services," in *22nd International World Wide Web Conference*, 2013, pp. 703–714.

[5] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[6] J. Nielsen, "The usability engineering life cycle," *IEEE Computer*, vol. 25, no. 3, pp. 12–22, 1992.

[7] L. Zhao and X. Zhou, "Slow or down?: Seem to be the same for cloud users," in *The first Workshop on Emerging Technologies for software-defined and reconfigurable hardware-accelerated Cloud Datacenters*, 2017, pp. 1–2.

[8] B. Cai, R. Zhang, X. Zhou, L. Zhao, and K. Li, "Experience availability: Tail-latency oriented availability in software-defined cloud computing," *Journal of Computer Science and Technology*, vol. 32, no. 2, pp. 250–257, 2017.

[9] Q. Lu, X. Xu, L. Zhu, L. Bass, Z. Li, S. Sakr, P. L. Bannerman, and A. Liu, "Incorporating uncertainty into in-cloud application deployment decisions for availability," in *IEEE Sixth International Conference on Cloud Computing*, 2013, pp. 454–461.

[10] X. Xu, Q. Lu, L. Zhu, Z. Li, S. Sakr, H. Wada, and I. Weber, "Availability analysis for deployment of in-cloud applications," in *Proceedings of the 4th international ACM Sigsoft symposium on Architecting critical systems*, 2013, pp. 11–16.

[11] R. Entezari-Maleki, K. S. Trivedi, and A. Movaghar, "Performability evaluation of grid environments using stochastic reward nets," *IEEE Transactions on Dependable and Secure Computing*, vol. 12, no. 2, pp. 204–216, 2015.

[12] T. Grainger, T. Potter, and Y. Seeley, *Solr in action*. Manning Cherry Hill, 2014.

[13] D. Bruneo, "A stochastic model to investigate data center performance and qos in iaas cloud computing systems," *IEEE Transactions on Parallel Distributed Systems*, vol. 25, no. 3, pp. 560–569, 2014.

[14] R. Han, M. M. Ghanem, L. Guo, Y. Guo, and M. Osmond, "Enabling cost-aware and adaptive elasticity of multi-tier cloud applications," *Future Generation Computer Systems*, vol. 32, no. 2, pp. 82–98, 2014.

[15] J. K. Muppala, K. S. Trivedi, V. Mainkar, and V. G. Kulkarni, "Numerical computation of response time distributions using stochastic reward nets," *Annals of Operations Research*, vol. 48, no. 2, pp. 155–184, 1994.

[16] G. Ciardo, J. K. Muppala, and K. S. Trivedi, "SPNP: stochastic petri net package," in *International Workshop on Petri Nets and PERFORMANCE MODELS*, 1989, pp. 142–151.

[17] H. V. Karambelkar, *Scaling big data with Hadoop and Solr; 2nd ed.* Birmingham: Packt Publ., 2015.

[18] "Apache jmeter," http://jmeter.apache.org/, 2017.

[19] B. Wei, C. Lin, and X. Kong, "Dependability modeling and analysis for the virtual data center of cloud computing," in *Proc. High Performance Computing and Communications (HPCC)*, 2011, pp. 784–789.

[20] A. Volkanovski, M. Čepin, and B. Mavko, "Application of the fault tree analysis for assessment of power system reliability," *Reliability Engineering & System Safety*, vol. 94, no. 6, pp. 1116–1127, 2009.

[21] K. S. Trivedi, *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. John Wiley & Sons, 2008.

[22] Q. Wu, M. Zhang, R. Zheng, Y. Lou, and W. Wei, "A qos-satisfied prediction model for cloud-service composition based on a hidden markov model," *Mathematical Problems in Engineering*, vol. 2013, no. 3, pp. 275–289, 2013.

[23] F. Longo, R. Ghosh, V. K. Naik, and K. S. Trivedi, "A scalable availability model for infrastructure-as-a-service cloud," in *Proc. the 41st IEEE/IFIP International Conference on Dependable Systems & Networks*, 2011, pp. 335–346.

[24] B. Wei, C. Lin, and X. Kong, "Dependability modeling and analysis for the virtual data center of cloud computing," in *Proc. High Performance Computing and Communications*, 2011, pp. 784–789.

[25] J. Dantas, R. Matos, J. Araujo, and P. Maciel, "Models for dependability analysis of cloud computing architectures for eucalyptus platform," *International Transactions on Systems Science and Applications*, vol. 8, pp. 13–25, 2012.

[26] H. Fujita, Y. Matsuno, T. Hanawa, M. Sato, S. Kato, and Y. Ishikawa, "Ds-bench toolset: Tools for dependability benchmarking with simulation and assurance," in *Proc. IEEE/IFIP International Conference on Dependable Systems and Networks*, 2012, pp. 1–8.

[27] A. Sangroya, D. Serrano, and S. Bouchenak, "Benchmarking dependability of mapreduce systems," in *Proc. the 31st IEEE Symposium on Reliable Distributed Systems*, 2012, pp. 21–30.

[28] A. Sangroya, S. Bouchenak, and D. Serrano, "Experience with benchmarking dependability and performance of mapreduce systems," *Performance Evaluation*, vol. 101, pp. 1–19, 2016.

[29] Y. Zhang, D. Meisner, J. Mars, and L. Tang, "Treadmill: Attributing the source of tail latency through precise load testing and statistical inference." *Acm Sigarch Computer Architecture News*, vol. 44, no. 3, pp. 456–468, 2016.

[30] H. Kasture and D. Sanchez, "Tailbench: a benchmark suite and evaluation methodology for latency-critical applications," in *IEEE International Symposium on Workload Characterization (IISWC)*, 2016, pp. 1–10.

[31] B. Urgaonkar, G. Pacifici, P. J. Shenoy, M. Spreitzer, and A. N. Tantawi, "An analytical model for multi-tier internet services and its applications," in *ACM SIGMETRICS Performance Evaluation Review*, 2005, pp. 291–302.

[32] N. Roy, A. S. Gokhale, and L. W. Dowdy, "Impediments to analytical modeling of multi-tiered web applications," in *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2010, pp. 441–443.

[33] Y. Sakuma, A. Inoie, K. Kawanishi, and M. Miyazawa, "Tail asymptotics for waiting time distribution of an m/m/s queue with general impatient time," *Journal of Industrial & Management Optimization*, vol. 7, no. 3, pp. 593–606, 2013.

[34] M. Grottke, V. Apte, K. Trivedi, and S. Woolet, "Response time distributions in networks of queues," *Queueing Networks*, vol. 154, pp. 587–641, 2011.