Resiliency Quantification for Large Scale Systems: An IaaS Cloud Use Case

Rahul Ghosh
Xerox Research Center, India
rahul.ghosh@xerox.com

Francesco Longo
Università degli Studi di Messina, Italy
flongo@unime.it

Vijay K. Naik
IBM T. J. Watson Research Center, USA
vkn@us.ibm.com

Andrew J. Rindos
IBM, USA
rindos@us.ibm.com

Kishor S. Trivedi
Duke University, USA
ktrivedi@duke.edu

ABSTRACT

We quantify the resiliency of large scale systems upon *changes* encountered beyond the normal system behavior. General steps for resiliency quantification are shown and resiliency metrics are defined to quantify the effects of changes. The proposed approach is illustrated through an Infrastructure-as-a-Service (IaaS) Cloud use case. Specifically, we assess the impact of changes in demand and available capacity on the Cloud resiliency using interacting state-space based sub-models. Since resiliency quantification involves understanding the transient behavior of the system, fixed-point variables evolve with time leading to non-homogenous Markov chains. In this paper, we present an algorithm for resiliency analysis when dealing with such non-homogenous sub-models. A comparison is shown with our past research, where we quantified the resiliency of IaaS Cloud performance using a one level monolithic model. Numerical results show that the approach proposed in this paper can scale for a real sized Cloud without significantly compromising the accuracy.

Keywords

Cloud, resiliency, interacting sub-models, non-homogeneous Markov chains.

1. INTRODUCTION

Large enterprise systems are difficult to manage due to their scale and complexity. Although such systems are carefully designed, they may still fail to deliver expected services due to failures or unexpected variations in the load. For business critical systems, such deviations from normal behavior may cause violation of service level agreements (SLA). Hence, during the design stage of such systems, a notion of built-in *resiliency* is necessary.

However, quantifying the system resiliency is difficult because of two key reasons: (1) lack of consensus on the definition of resiliency and (2) lack of systematic approaches for quantifying resiliency measures. The term resiliency is used in different fields and several definitions are present. Many researchers interpret resiliency as a synonym for fault-tolerance. However, the effect of fault tolerance can be captured by traditional dependability measures, such as reliability, availability, maintainability, and safety [4, 1]. DeBardeleben *et al.* [3] defined resiliency of a high end computing system as the ability to keep applications running and maintain an acceptable level of service when facing transient, intermittent, and permanent faults. However, the quantification of the effects of failures/repairs on a system is covered under the umbrella of dependability, joint analysis of contention/queuing and failures/repairs is known as performability [9], and the effect of failures/repairs on individual tasks is covered under the umbrella of task completion time analysis [2, 12]. Impact of (large scale) failures (e.g., natural disasters), intrusions, or attacks on systems/services and subsequent transient behavior is quantified by survivability analysis [10, 14, 17]. Sterbenz *et al.* [16] defined resiliency as the combination of trustworthiness (dependability, security, and performability) and tolerance (survivability, disruption tolerance, and traffic tolerance).

Our work about resiliency starts from the definition of Sterbenz *et al.* and is inspired from Laprie's [13] and Simoncini's [15] work in which resiliency is defined as the persistence of service delivery that can justifiably be trusted when facing *changes* - usually not encountered under normal operating conditions. Note that dynamic redundancy features in traditional fault tolerance with detection followed by reconfiguration are already included in classic dependability measures. The *changes* we are referring to here are beyond the envelope of system configurations already considered during the design stage and thus beyond fault tolerance. We also emphasize that survivability quantification only deals with *changes* due to (large scale) failures, intrusions, or attacks. The notion of resiliency is broader than survivability as the *changes* encountered by the system can be unexpected variations in the workload, faultload, or system capacity.

This paper presents a general approach for resiliency quantification of large scale systems through analytic models. The approach is illustrated by an Infrastructure-as-a-Service (IaaS) Cloud use case. IaaS Cloud provides computational resources in the form of virtual machine (VM) instances deployed in the provider's data center. Resiliency is a key

VALUETOOLS 2016, October 25-28, Taormina, Italy

DOI 10.4108/eai.25-10-2016.2266805

feature in a Cloud environment in order to maintain quality of service offerings and to respect SLAs established with Cloud users. Our contributions are summarized below:

(1) We develop a general resiliency quantification approach for large scale systems using state-space based analytic models. Several resiliency metrics are defined to quantify resiliency for a variety of measures of interest. In contrast to the traditional view of synonymizing resiliency with dependability, we show *how resiliency can be quantified even with respect to performance measures*. Nevertheless, the resiliency analysis approach proposed in this paper can also be applied to other dependability attributes, e.g., reliability, availability, and performability.

(2) Steady state performance of a large scale system is quantified using interacting sub-models where interdependencies are resolved by fixed-point iterations [8, 7]. Resiliency quantification involves understanding the transient behavior of the system. Transient analysis of interacting sub-models coupled with fixed-point iterations becomes non-trivial as the values of fixed-point variables change with time – leading to non-homogenous Markov sub-models. This paper presents an algorithm for the resiliency quantification when dealing with such non-homogenous, interacting Markov sub-models. To the best of our knowledge, ours is the first attempt to perform transient analysis of state-space based fixed-point iterative sub-models.

(3) Considering an IaaS Cloud use case, we quantify resiliency of two performance measures: job rejection rate and provisioning response delay (a function of the mean number of jobs under provision). Using resiliency metrics, we quantify the effects of sudden changes in client demand, i.e., job arrival rate and in system capacity, i.e., the number of available physical machines (PMs). We compute the values of resiliency metrics for ten different cases that capture the effects of sudden changes in demand and capacity.

(4) A comparison of the proposed resiliency quantification approach is shown with our past research [5] in which we quantified resiliency of an IaaS Cloud using a one level monolithic performance model. Such monolithic model becomes intractable as the size of the Cloud increases. Thus, resiliency quantification shown in our past research is restricted to toy examples while the approach developed in this paper extends our previous results to real sized Clouds. We show the scalability and accuracy of the new resiliency quantification approach by comparing it to the approach shown in [5]. Numerical results show that our new approach can quickly provide model solutions for large scale IaaS Cloud without significantly compromising the accuracy.

The paper is organized as follows. Our idea of resiliency quantification along with the definitions of resiliency metrics is presented in Section 2. Section 3 describes the IaaS Cloud system taken as a use case and formulates the problem. The proposed resiliency quantification approach is shown in Section 4. Numerical results are shown in Section 5 and concluding remarks are made in Section 6.

2. PROPOSED RESILIENCY QUANTIFICATION APPROACH

In this section, we outline the general steps of the approach for the resiliency quantification of a system via analytic models we propose. Moreover, we report formal definitions for our resiliency metrics.

Resiliency quantification. The general steps in our resiliency quantification approach are:

(1) Construct a stochastic analytic model of a given system to find measures of interest. With respect to such measures, the model can be a performance, availability, or performability model of the system.

(2) Determine the system measures of interest under normal conditions from the model developed in step (1). In this step, we compute the steady state value of the measures of interest without any application of changes.

(3) Apply changes to the system. Changes can be classified into two broad categories: (a) non-structural changes and (b) structural changes. Non-structural changes can be enforced by increasing (or decreasing) the values of input parameters of the model. Examples of such changes are variation of arrival rates or hardware/software failure rates. In a state-space model, structural changes can be enforced by adding (or removing) states and/or transitions. Examples of structural changes can be addition or removal of system components.

(4) Analyze the transient behavior of the system model to compute the measures of interest after applying the changes. Initial probabilities for this transient analysis are obtained from the steady state probabilities as computed from the normal behavior of the system model in step (2). After the changes are applied, transient response of the performance, reliability, availability, and performability measures quantifies the resiliency of the system.

Resiliency Metrics. Closely following the survivability definitions in [17], we define resiliency metrics to quantify the resiliency of a system when changes are enforced. Let $M_{ss}^{(bc)}$ and $M_{ss}^{(ac)}$ denote the steady state values of the measures (for which resiliency is computed) before and after the application of change, respectively. The following four metrics are defined:

(1) Settling Time (t_{set})**.** This is defined as the elapsed duration from the time when the change is applied to the system, until the measure of interest reaches and stays within $M_{ss}^{(ac)} \pm \delta\%$ of $|M_{ss}^{(bc)} - M_{ss}^{(ac)}|$.

(2) Peak Overshoot (PO) **or Undershoot** (PU)**.** Let $M_{peak}^{(ac)}$ be the maximum deviation of the measure from its steady state value after change is applied. Peak overshoot/undershoot (in percentage) is defined as:

$$PO(\%) = \frac{|M_{peak}^{(ac)} - M_{ss}^{(ac)}|}{M_{ss}^{(ac)}} \times 100 \quad (1)$$

(3) Peak Time (t_{peak})**.** It is defined as the elapsed duration from the time when the change is applied to the system until the measure of interest reaches the maximum deviation $M_{peak}^{(ac)}$.

(4) γ-Percentile Time ($t_{percentile}$)**.** It is defined for two cases: (1) when $M_{ss}^{(bc)} \geq M_{ss}^{(ac)}$, it is the elapsed duration from the time when the change is applied to the system until the output measure reaches $M_{ss}^{(ac)} + (100-\gamma)\%$ of $|M_{ss}^{(bc)} - M_{ss}^{(ac)}|$ for the first time, and (2) when $M_{ss}^{(bc)} \leq M_{ss}^{(ac)}$, it is the elapsed duration from the time when the change is applied to the system until the output measure reaches $M_{ss}^{(ac)} - (100-\gamma)\%$ of $|M_{ss}^{(bc)} - M_{ss}^{(ac)}|$ for the first time. $(100-\gamma)$ is always greater than δ as used in settling time definition.

We illustrate the proposed approach via resiliency quantification of an IaaS Cloud.

3. CASE STUDY FOR IAAS CLOUD PERFORMANCE

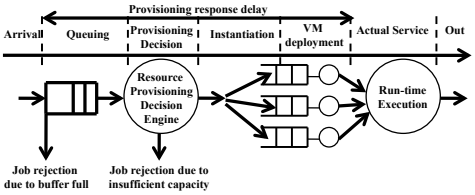


Figure 1: Provisioning and servicing steps in IaaS Cloud.

System Description. In an IaaS Cloud, when a request is processed, a pre-built image is used to deploy one or more VM instances or a pre-deployed VM is customized and made available to the requester. VMs are deployed on PMs each of which may be shared by multiple VMs. The deployed VMs are provisioned with request specific CPU, RAM, and disk capacity. In this paper, we show the analysis for an IaaS Cloud where the PMs are grouped into three pools: hot (running), warm (turned on, but not ready) and cold (turned off). A pre-instantiated VM can be readily provisioned and brought to ready state on a running PM (hot PM) with minimum provisioning delay. Instantiating a VM from an image and deploying it on a warm PM needs additional provisioning time. PMs in the cold pool are turned-off when not in use and deploying a VM on such a PM requires additional startup delays. Thus, organization of PMs in multiple pools helps to minimize power and cooling costs without incurring into high startup delays for all VMs. We assume that all PMs in a pool are identical and all requests are homogeneous, where each request is for one VM. Shown in Fig. 1 is the life-cycle of a request as it moves through the system. Submitted user requests enter a first-come, first-served (FCFS) queue. The request at the head of the queue is processed by a Resource Provisioning Decision Engine (RPDE) as follows. The request is provisioned on a hot PM if a pre-instantiated but unassigned VM exists. If no hot PM is available, a PM from the warm pool is used for provisioning the requested VM. If all warm PMs are busy, a PM from the cold pool is used. If no PM is available, the request is rejected (service unavailable). When a running job exits, the capacity used by that VM is released and becomes available for provisioning the next job.

Problem Statement. Performance of the above described IaaS Cloud can be quantified using scalable interacting sub-models that capture different phases in the life-cycle of a request. If all the inter-event times are assumed to be exponentially distributed, the sub-models are homogeneous continuous time Markov chains (CTMC). However, during transient analysis, the generator matrices of interacting sub-models evolve with time thus leading to non-homogeneous CTMCs. This paper describes an algorithm for resiliency quantification when dealing with non-homogeneous, interacting sub-models. We quantify the resiliency of IaaS Cloud for two performance measures - (i) job rejection rate (ρ_{reject}) and (ii) mean number of jobs in RPDE ($E[N_{RPDE}]$). Resiliency metrics are computed to quantify the effects on the system due to changes in job arrival rate and system capacity. Finally, the scalability and accuracy of the proposed resiliency quantification approach are compared with the resiliency quantification approach for a monolithic model of the system.

4. RESILIENCY QUANTIFICATION FOR A SCALABLE CLOUD PERFORMANCE MODEL

In this section, we briefly describe the interacting sub-models approach for performance quantification of IaaS Cloud followed by an algorithm for resiliency quantification using such sub-models. The detailed performance model can be found in [8]. Final solution of the overall model is obtained by fixed-point iteration over individual sub-models.

4.1 An interacting sub-models approach

Resource provisioning decision engine (RPDE) sub-model. Fig. 2 shows RPDE sub-model. States of the sub-

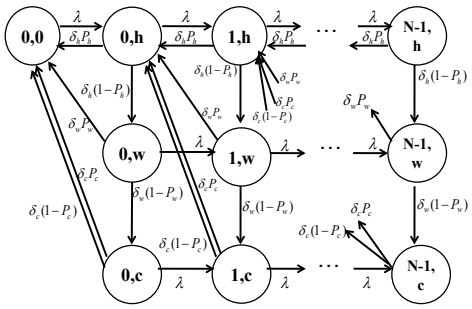


Figure 2: CTMC sub-model for RPDE.

model in Fig. 2 are indexed by (q, x), where q denotes the number of jobs in the RPDE queue and $x \in \{h, w, c\}$ denotes the pool in which the RPDE is searching for available PMs. Input parameters for this sub-model are: (i) job arrival rate (λ), (ii) mean search delays to find a PM from hot/warm/cold pool that can be used for resource provisioning ($1/\delta_h$, $1/\delta_w$ and $1/\delta_c$, respectively), (iii) probabilities that a hot/warm/cold PM can accept a job for resource provisioning (P_h, P_w and P_c, respectively) and (iv) maximum number of jobs in RPDE (N). Among all the input parameters, P_h, P_w, and P_c are computed as outputs from the VM provisioning sub-models described later. From the RPDE sub-model, using Markov reward approach [18], we compute job rejection probability due to buffer full (P_{block}), rejection probability due to insufficient capacity (P_{drop}), their sum: $P_{reject} = P_{block} + P_{drop}$, and mean number of jobs in RPDE ($E[N_{RPDE}]$). Then, multiplying by arrival rate, we can compute job rejection rate due to buffer full (ρ_{block}), rejection rate due to insufficient capacity ρ_{drop}, and their sum: $\rho_{reject} = \rho_{block} + \rho_{drop}$.

VM provisioning sub-models. VM provisioning sub-

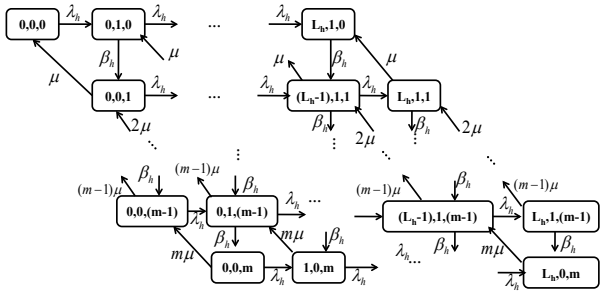


Figure 3: VM provisioning sub-model for a hot PM.

models capture the provisioning and deployment of requested

VMs to accepted jobs. For each hot, warm, and cold PM, we have one CTMC which keeps track of the number of running VMs. VM provisioning sub-model of a pool is the union of individual provisioning sub-models of each PM in that pool.

Fig. 3 shows VM provisioning sub-model for a hot PM. Conceptually, the overall hot pool is modeled by a set of independent hot PM sub-models. Note that only one PM sub-model needs to be solved. States of the sub-model in Fig. 3 are indexed by (i, j, k), where i denotes the number of jobs in the PM buffer, j denotes the number of VMs currently being provisioned, k denotes the number of running VMs. Input parameters for a hot PM CTMC are: (i) effective job arrival rate to each hot PM (λ_h), (ii) rate at which VM instantiation, provisioning, and configuration occurs (β_h), (iii) job service rate (μ), (iv) buffer size of hot PM (L_h), and (v) maximum number of VMs that can be deployed on a PM (m). Assuming a total of $n_{_h}$ PMs in the hot pool, λ_h is given by $\lambda(1 - P_{block})/n_{_h}$. Observe that P_{block} is computed from RPDE sub-model. Output of the hot pool sub-model is the steady state probability P_h that at least one PM in the hot pool can accept a job for provisioning.

CTMC for a warm PM [8] is similar to the hot PM sub-model, with few differences: (i) Effective arrival rate to each warm PM is λ_w. (ii) When no VM is running or being provisioned, warm PM is turned on but not ready for use. Upon a job arrival in this state, the warm PM requires some additional startup time to make it ready for use. Rate at which warm PM is made ready for use is γ_w. (iii) Rate of VM instantiation, provisioning, and configuration for first VM is β_w while for subsequent VMs rate is β_h. Buffer size of each warm PM is L_w. Assuming a total of n_w PMs in the warm pool, λ_w is given by $\lambda(1-P_{block})(1-P_h)/n_w$. Output of the warm pool sub-model is the steady state probability P_w that at least one PM in the warm pool can accept a job for provisioning.

CTMC for a cold PM is similar to the warm PM sub-model and cold pool sub-model is the set of $n_{_c}$ independent cold PM sub-models. Main differences between a warm and a cold PM sub-model are: (i) effective arrival rate (λ_c), (ii) rate at which startup is executed (γ_c), (iii) initial VM provisioning rate (β_c) and buffer size (L_c). λ_c is given by $\lambda(1 - P_{block})(1 - P_h)(1 - P_w)/n_{_c}$. Output of the cold pool sub-model is the steady state probability P_c that at least one PM in the cold pool can accept a job for provisioning. Probabilities P_h, P_w, and P_c as obtained from hot, warm, and cold pool sub-models, respectively, are used in RPDE sub-model as input parameters.

4.2 Resiliency quantification using interacting sub-models

As described in Section 2, resiliency quantification requires analyzing transient behavior of a system after a change is applied. However, in interacting sub-models approach, generator matrices of sub-models evolve with time, leading to sub-models that are non-homogeneous continuous time Markov chains (NHCTMC) [18]. We explain this phenomenon with an example. During transient analysis of interacting sub-models, $P_{_{block}}$, obtained as an output measure from RPDE sub-model, is used in VM provisioning sub-models as an input parameter to compute P_h, P_w, and P_c. Since the transient value of $P_{_{block}}$ changes with time, values of job-arrival rates in hot, warm, and cold sub-models, as derived from $P_{_{block}}$, also change with time. As a result, the generator matrices of hot, warm, and cold sub-models become time-dependent. The transient values of P_h, P_w, and P_c are used to compute transition rates in the RPDE sub-model. This leads to a time-dependent generator matrix of RPDE sub-model. To solve this problem, we use an approach inspired from piece-wise constant approximation method [18]. We divide the time interval for transient analysis into small sub-intervals and we assume that the values of NHCTMC transition rates are constant within these sub-intervals. Fixed-point iteration among the sub-models is carried out for each sub-interval and outputs from one sub-interval are transferred as inputs to the next sub-interval. In order to automate such operations, in the next section we propose an algorithm for resiliency quantification using interacting sub-models.

Algorithm 1 Resiliency quantification using interacting sub-models

Input: (i) hot, warm, cold and RPDE sub-models, (ii) ω_1 and ω_2: values of an input parameter before and after the change respectively, (iii) $[0, T]$: length of the time interval for transient analysis, (iv) Δ: length of the sub-intervals.
Output: Values of performance measures at the end of each sub-interval.

1: declare $\mathbf{H}, \mathbf{H'}, \mathbf{H''}, \mathbf{W}, \mathbf{W'}, \mathbf{W''}, \mathbf{C}, \mathbf{C'}, \mathbf{C''}, \mathbf{R}, \mathbf{R'}, \mathbf{R''}$: vector of *probability*;
2: declare $P_h, P'_h, P''_h, P_w, P'_w, P''_w, P_c, P'_c, P''_c, P_{block}, P'_{block}, P''_{block}, P'''_{block}$: *probability*;
3: declare ϵ_f: *upper bound on error*;
4: declare *stop*: *boolean*;
5: declare t: *time instant*;
6: $t \leftarrow 0$;
7: $\{\mathbf{H'}, \mathbf{W'}, \mathbf{C'}, \mathbf{R'}, P_h, P_w, P_c, P_{block}\} \leftarrow fp\_steady(\omega_1)$;
8: output the values of performance measures at steady state;
9: $\{\mathbf{H}, \mathbf{W}, \mathbf{C}, \mathbf{R}\} \leftarrow map(\mathbf{H'}, \mathbf{W'}, \mathbf{C'}, \mathbf{R'})$;
10: $stop \leftarrow$ false;
11: do:
12: $t \leftarrow t + \Delta$;
13: $P'_{block} \leftarrow P_{block}$; $P'_h \leftarrow P_h$; $P'_w \leftarrow P_w$; $P'_c \leftarrow P_c$;
14: do:
15: $P'''_{block} \leftarrow P_{block}$;
16: $\{\mathbf{H''}, P''_h\} \leftarrow hot\_transient(\mathbf{H}, P_{block}, \omega_2, \Delta)$;
17: $P_h \leftarrow (P''_h + P'_h)/2$;
18: $\{\mathbf{W''}, P''_w\} \leftarrow warm\_transient(\mathbf{W}, P_{block}, P_h, \omega_2, \Delta)$;
19: $P_w \leftarrow (P''_w + P'_w)/2$;
20: $\{\mathbf{C''}, P''_c\} \leftarrow cold\_transient(\mathbf{C}, P_{block}, P_h, P_w, \omega_2, \Delta)$;
21: $P_c \leftarrow (P''_c + P'_c)/2$;
22: $\{\mathbf{R''}, P''_{block}\} \leftarrow rpde\_transient(\mathbf{R}, P_h, P_w, P_c, \omega_2, \Delta)$;
23: $P_{block} \leftarrow min(P_{block}, P''_{block}) + |P''_{block} - P_{block}|/2$;
24: $P_{block} \leftarrow (P_{block} + P'_{block})/2$;
25: while $|P_{block} - P'''_{block}| < \epsilon_f$
26: repeat steps 15-24 to update: $\mathbf{H''}, \mathbf{W''}, \mathbf{C''}, \mathbf{R''}, P_h, P_w, P_c, P_{block}$, at t
27: $\mathbf{H} \leftarrow \mathbf{H''}$; $\mathbf{W} \leftarrow \mathbf{W''}$; $\mathbf{C} \leftarrow \mathbf{C''}$; $\mathbf{R} \leftarrow \mathbf{R''}$;
28: output the values of performance measures at time t;
29: if $t \geq T$:
30: $stop \leftarrow$ true;
31: while *stop* is false

Resiliency quantification algorithm for large scale IaaS Cloud. Algorithm 1 presents the pseudocode for resiliency quantification using interacting sub-models approach. Input parameters to this algorithm are: (i) hot, warm, cold, and RPDE sub-models, (ii) values of an input parameter before and after the change (ω_1 and ω_2 respec-

tively), (iii) length of the time interval ($[0, T]$) for transient analysis, and (iv) length of the sub-intervals (Δ) used in the piece-wise constant approximation method. In this paper, for IaaS Cloud resiliency quantification, parameters ω_1 and ω_2 represent values of job arrival rate or number of PMs per pool. At the end of each sub-interval, output measures from Algorithm 1 are the values of performance measures.

To understand the system behavior before the change is applied, we solve the sub-models iteratively for steady state analysis, using the function *fp_steady(.)*. From steady state analysis, we obtain: (i) state probability vectors of the hot, warm, cold, and RPDE sub-models, (ii) steady state values of fixed-point variables, and (iii) steady state values of performance measures.

Before the transient analysis, it is necessary to map the state spaces of the sub-models before the change into the state spaces of the sub-models after the change. This is performed by function *map(.)*. A mapping is one-to-one if non-structural change is applied to a sub-model (i.e., if the state space of the sub-model before and after the change is exactly same). Otherwise, one-to-many or many-to-one mapping is performed (i.e., if the state space of the sub-model is modified after the change).

Next, the sub-models are solved for transient analysis. In Algorithm 1, outer loop (step 11 to step 31) runs until we finish the transient analysis for all sub-intervals. The inner loop (step 14 to step 25) represents the fixed-point iteration within each sub-interval. For transient analysis in the first sub-interval, initial state probability vectors ($\mathbf{H}, \mathbf{W}, \mathbf{C}$, and $\mathbf{R}$) and fixed point variables (P_h, P_w, P_c, and P_{block}) are obtained from the steady state analysis. For subsequent sub-intervals, values of $\mathbf{H}, \mathbf{W}, \mathbf{C}, \mathbf{R}, P_h, P_w, P_c$, and P_{block} are obtained from transient analysis in previous sub-intervals. However, an average is computed between the current value of each fixed-point variable and the value obtained from previous sub-interval. Such averaging process reduces the errors introduced due to non-homogeneous behavior of the system. Before starting the fixed-point iteration within a sub-interval, we store the values of fixed point variables (P'_h, P'_w, P'_c, and P'_{block}) obtained from previous sub-intervals. It can be shown that fixed-point variables P_h, P_w, and P_c can be expressed as a function of P_{block}. So, we check the values of P_{block} in successive iterations to determine the condition for convergence in fixed-point iteration. Fixed point iteration for a sub-interval starts by calling the function *hot_transient(.)* to obtain the transient solution of the hot sub-model. Observe that we use ω_2 to enforce the change in the system behavior. Outputs of *hot_transient(.)* are the transient state probability vector of the hot sub-model ($\mathbf{H}''$) and the transient probability that the job can be accepted in the hot pool (P''_h). P_h is updated by taking an average of its current value (P''_h) and the value obtained from the previous sub-interval (P'_h). Function *warm_transient(.)* uses this updated P_h as an input parameter and solves the warm sub-model for transient solution. Outputs of *warm_transient(.)* are the transient state probability vector of the warm sub-model ($\mathbf{W}''$) and the transient probability that job can be accepted in the warm pool (P''_w). P_w is updated by taking average of P''_w and P'_w. Function *cold_transient(.)* uses updated values of P_h and P_w to solve the cold sub-model for transient solution. Outputs of *cold_transient(.)* are the transient state probability vector of the cold sub-model ($\mathbf{C}''$) and the transient probability that the job can be accepted in the cold pool (P''_c). P_c is updated by taking average of P''_c and P'_c. Function *rpde_transient(.)* uses updated values of P_h, P_w, and P_c to solve the RPDE sub-model for transient solution. Outputs of *rpde_transient(.)* are the transient state probability vector of the RPDE sub-model ($\mathbf{R}''$) and P''_{block}. Value of P_{block} is updated twice: (i) for improving the rate of convergence in successive fixed-point iteration (step 23 in Algorithm 1) and (ii) for reducing the errors due to non-homogenous behavior of the system (step 24 in Algorithm 1). After the fixed-point iteration in a sub-interval finishes, we update the values of initial state probability vectors ($\mathbf{H}, \mathbf{W}, \mathbf{C}$, and $\mathbf{R}$) as well as the fixed-point variables (P_h, P_w, P_c, and P_{block}). We use these updated values in the next sub-interval. Observe that within a sub-interval, fixed-point variables (used as input parameters to the sub-models) are updated in each iteration while the initial state probability vectors used to obtain the transient solutions of the sub-models remain unchanged.

5. NUMERICAL RESULTS

Case	Description
case-I	Increase in λ, 100 PMs in each pool
case-II	Increase in λ, 1000 PMs in each pool
case-III	Decrease in λ, 100 PMs in each pool
case-IV	Decrease in λ, 1000 PMs in each pool
case-V	Removal of 900 PMs from hot, 750 PMs from warm and cold pool each
case-VI	Removal of 850 PMs from hot and warm each, 700 PMs from cold pool
case-VII	Removal of 800 PMs from each pool
case-VIII	Addition of 150 PMs to hot pool
case-IX	Addition of 75 PMs to hot and warm pool each
case-X	Addition of 50 PMs to each pool

Table 1: Description of the considered cases.

In this section, we discuss key results for large scale IaaS Cloud resiliency quantification using scalable interacting sub-models. Subsequently, model solution time and accuracy of our proposed resiliency algorithm for interacting sub-models are compared with the monolithic model shown in [5].

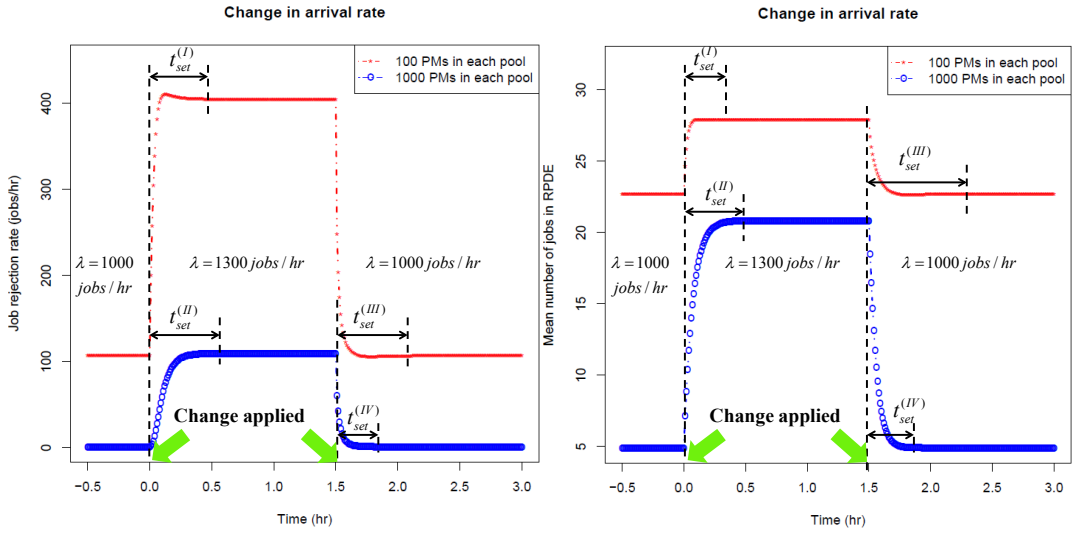
5.1 Resiliency quantification for large scale IaaS Cloud

We use SHARPE [19] software package to solve interacting sub-models and quantify the IaaS Cloud resiliency of the performance measures. Using Python scripts, we automate the generation of SHARPE input files and implement the following tasks: (i) mapping the state space of the sub-models before the change into the state space of the sub-models after the change, (ii) transferring the initial probability vectors and fixed-point variables from current sub-interval to the next sub-interval. We quantified large scale IaaS Cloud resiliency upon two types of changes: change in job-arrival rate and change in the number of PMs. In Table 1, we define ten cases that we have analyzed and we numerically compare the corresponding resiliency metrics (with $\gamma = 90$ and $\delta = 0.1$) in Table 2.

Effect of changing job-arrival rate (λ). Fig. 4(a) shows the transient effects of changing the job arrival rate (λ) on job rejection rate. We analyze the resiliency of two Cloud configurations with 100 PMs and 1000 PMs in each pool. For both configurations, maximum 2 VMs can run simultaneously on a PM. At time instance $t = 0$, arrival rate is increased from 1000 to 1300 jobs/hr. In case-I and case-II, settling times are $t_{set}^{(I)}$ and $t_{set}^{(II)}$ respectively. At $t = 1.5$ hr,

Cases	Resiliency metrics for job rejection rate				Resiliency metrics for mean number of jobs in RPDE			
	t_{set}	$PO(\%)$	t_{peak}	$t_{percentile}$	t_{set}	$PO(\%)$	t_{peak}	$t_{percentile}$
case-I	0.47	1.45	0.13	0.06	0.35	0.16	0.13	0.05
case-II	0.52	-	-	0.21	0.48	-	-	0.16
case-III	0.55	1.1	0.29	0.06	0.78	0.27	0.32	0.13
case-IV	0.28	-	-	0.07	0.36	-	-	0.13
case-V	0.87	-	-	0.37	0.75	-	-	0.29
case-VI	1.22	-	-	0.61	1.17	-	-	0.56
case-VII	1.73	-	-	0.90	1.75	-	-	0.88
case-VIII	0.78	-	-	0.15	0.89	-	-	0.20
case-IX	0.98	-	-	0.21	1.05	-	-	0.26
case-X	0.91	-	-	0.22	0.97	-	-	0.27

Table 2: Comparison of resiliency metrics for the different cases described in Table 1.



(a) (b)

Figure 4: Effect of changing job arrival rate (a) job rejection rate and (b) mean number of jobs in the RPDE.

job arrival rate is reduced to initial value, i.e., 1000 jobs/hr. When the change is removed, settling times are $t_{set}^{(III)}$ and $t_{set}^{(IV)}$ respectively.

(#PMs in each pool, maximum # VMs per PM)	Monolithic model	Interacting sub-models
(1, 1)	912	27
(1, 2)	4, 655	35
(1, 4)	20, 691	47
(1, 8)	116, 603	71
(1, 16)	768, 075	119
(1, 32)	5, 543, 915	215
(1, 38)	9, 848, 061	251
(1, 39)	Memory overflow	257
(2, 1)	43, 776	27
(3, 1)	2, 101, 248	27
(4, 1)	Memory overflow	27
(500, 64)	–	407

Table 3: Comparison of model states.

In Fig. 4(b), similar transient effects are shown for the mean number of jobs in RPDE (a measure of congestion in the Cloud).

Effect of changing system capacity. Next, we quantify the resiliency of IaaS Cloud due to addition or removal of PMs. Fig. 5(a) shows the transient effects of removing PMs on the job rejection rate. We start with a scenario where 1000 PMs are equally distributed in each pool (as denoted by $(1000, 1000, 1000)$). At $t = 0$, we remove total 2400 PMs in three different ways: (i) 900 PMs are removed from hot pool and 750 PMs are removed from warm and cold pool each (denoted by $(100, 250, 250)$), (ii) 850 PMs are removed from hot and warm pool each and 700 PMs are removed from cold pool (denoted by $(150, 150, 300)$), and (iii) 800 PMs are removed from each pool (denoted by $(200, 200, 200)$). After the removal of the PMs, rejection rate increases and finally reaches a steady state. Settling time is maximum when 800 PMs are removed from each pool, i.e., the Cloud configuration is changed from $(1000, 1000, 1000)$ to $(200, 200, 200)$. Although, observe that the job rejection rate is maximum when the Cloud configuration is changed from $(1000, 1000, 1000)$ to $(100, 250, 250)$. Similar effects on the mean number of jobs in RPDE are shown in Fig. 5(b).

Fig. 6(a) shows the transient effects of adding new PMs

(#PMs in each pool, maximum # VMs per PM)	Monolithic model	Interacting sub-models
(1, 1)	0.124	0.066
(1, 2)	0.196	0.074
(1, 4)	0.556	0.088
(1, 8)	2.998	0.141
(1, 16)	79.563	0.208
(1, 32)	188.174	0.346
(1, 38)	293.672	0.399
(1, 39)	Memory overflow	0.406
(2, 1)	2.024	0.063
(3, 1)	166.704	0.062
(4, 1)	Memory overflow	0.060
(500, 64)	–	0.055

Table 4: Comparison of model solution times in seconds.

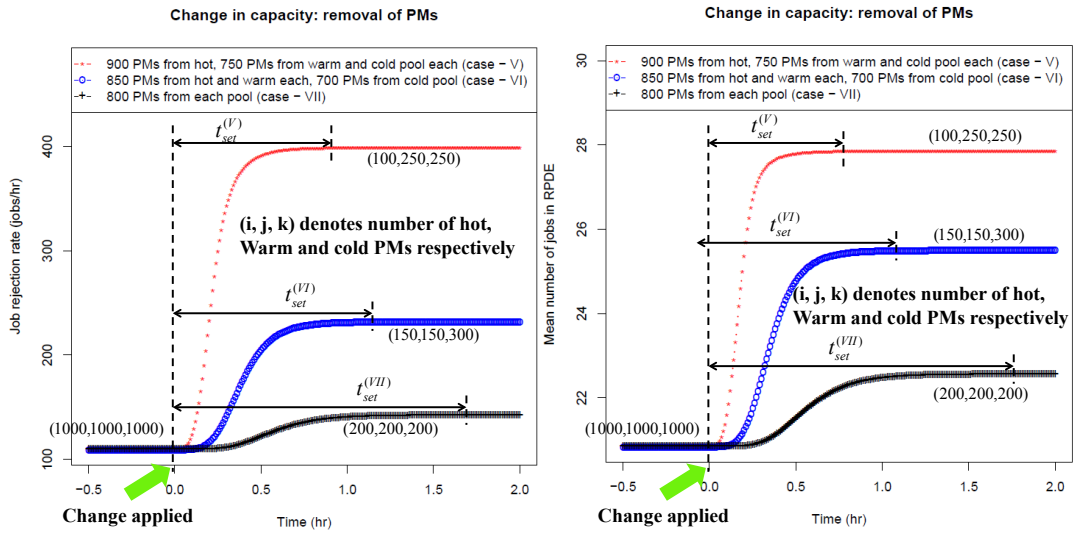


Figure 5: Effect of removing PMs on (a) job rejection rate and (b) mean number of jobs in the RPDE.

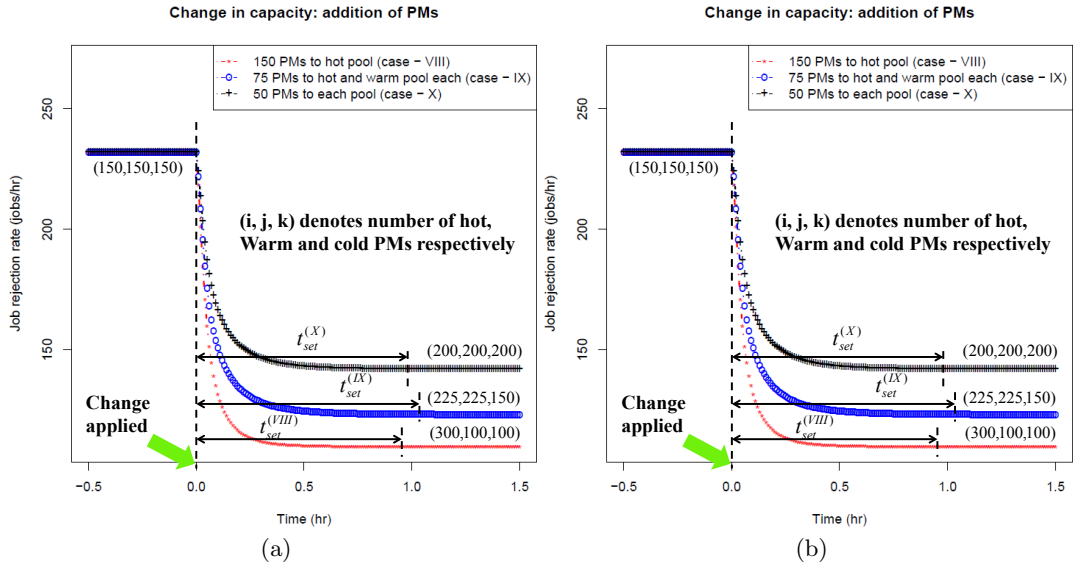


Figure 6: Effect of adding PMs on (a) job rejection rate and (b) mean number of jobs in the RPDE.

on job rejection rate. We start with a scenario where 150 PMs are equally distributed in each pool (as denoted by $(150, 150, 150)$). At $t = 0$, we add total 150 PMs in three different ways: (i) 150 PMs are added to hot pool (denoted by $(300, 150, 150)$), (ii) 75 PMs are added to hot and warm pool each (denoted by $(225, 225, 150)$), and (iii) 50 PMs are added to each pool (denoted by $(200, 200, 200)$). After the addition of the PMs, rejection rate decreases and finally reaches a steady state. Among the three changes, settling time is minimum when PMs are added only to the hot pool. Similar effects on the mean number of jobs in RPDE are shown in Fig. 6(b).

5.2 Comparison of scalability with monolithic model.

We use Stochastic Petri Net Package (SPNP) [11] for resiliency quantification using the monolithic model. SPNP allows to easily map the state space of the model before the change into the state space of the model after the change and to compute the initial probability vectors in a straightforward manner. Both the interacting sub-models and the monolithic model were solved using a desktop PC with Intel Core 2 Duo processor (E8400, 3.0 GHz) and 4 GB memory. In Table 3, we report the state space for both the monolithic model and interacting sub-models. Monolithic model runs into a memory overflow problem when the number of PMs in each pool increases beyond 3 and the number of VMs per PM increases beyond 38. We observe that the state space size of the monolithic model increases quickly and becomes too large to construct the reachability graph even for small

Algorithms	Resiliency metrics for job rejection rate				Resiliency metrics for mean number of jobs in RPDE			
	t_{set}	$PO(\%)$	t_{peak}	$t_{percentile}$	t_{set}	$PO(\%)$	t_{peak}	$t_{percentile}$
Interactive sub-models	0.12	$1.52e-04$	0.20	0.04	0.10	$0.21e-04$	0.18	0.04
Monolithic model	0.10	$0.85e-04$	0.21	0.04	0.11	$0.24e-04$	0.23	0.04

Table 5: Comparison of resiliency metrics between the two algorithm when arrival rate is increased.

number of PMs. However, with interacting sub-models approach, the state space increases at a slower rate as the maximum number of VMs per PM is increased and remains constant with increasing number PMs. A comparison of solution times is shown in Table 4. Solution time for monolithic model increases almost exponentially with the increase in model size. Solution time for interacting sub-models remains almost constant with the increase in model size. Clearly, interacting sub-models approach facilitates fast quantification of large scale IaaS Cloud resiliency.

5.3 Comparison of accuracy

Using 2 PMs per pool and 1 VM per PM, we compare the accuracy of resiliency quantification using monolithic model and interacting sub-models approach. We quantify the resiliency of two performance measures: job rejection rate and mean number of jobs in RPDE respectively, when subjected to a change in arrival rate (λ). At $t = 0$, we change the job arrival rate from 350 jobs/hr to 400 jobs/hr. In Table 5, we observe that the resiliency metrics are comparable in both the approaches. There are two sources of errors in interacting sub-models based resiliency quantification approach. They are: (i) errors due to decomposition of the monolithic model and (ii) errors due to non-homogeneous behavior of the system during transient phase. In this paper, we only focus on resiliency metrics that capture the errors in the transient analysis. Analysis of errors due to model decomposition, is shown in [6].

6. CONCLUSIONS AND FUTURE WORK

In this paper, we quantify the resiliency of IaaS Cloud upon changes in job arrival rate and system capacity. To the best of our knowledge, ours is perhaps the first work that quantifies resiliency of large IaaS Cloud using a scalable and fast analytic approach. Resiliency metrics described in this paper can be used to quantify the transient behavior of a IaaS Cloud under sudden changes and can help during design stage of the IaaS Cloud. In fact, using the resiliency metrics, different configurations and design alternatives (faster provisioning vs. slower provisioning, different distributions of PMs in pools) can be evaluated that can meet provider specific service goals. Although, the resiliency approach described in this paper is built around performance models, proposed approach is general and can be applied to any state-space based model.

7. REFERENCES

[1] D. Bruneo, S. Distefano, F. Longo, A. Puliafito, and M. Scarpa. Reliability assessment of wireless sensor nodes with non-linear battery discharge. 2010.

[2] P. F. Chimento and K. S. Trivedi. The completion time of programs on processors subject to failure and repair. *IEEE Trans. on Computers*, 42(10):1184–1194, 1993.

[3] N. DeBardeleben et al. *High-End Computing Resilience: Analysis of Issues Facing the HEC Community and Path-Forward for Research and Development*. 2010.

[4] S. Distefano, F. Longo, and M. Scarpa. Symbolic representation techniques in dynamic reliability evaluation. pages 45–53, 2010.

[5] R. Ghosh, F. Longo, V. K. Naik, and K. S. Trivedi. Quantifying resiliency of IaaS cloud. In *RACOS Workshop*, 2010.

[6] R. Ghosh, F. Longo, V. K. Naik, and K. S. Trivedi. Modeling and performance analysis of large scale iaas clouds. *Elsevier Future Generation Computer Systems*, 29(5):1216–1234, 2013.

[7] R. Ghosh, F. Longo, R. Xia, V. Naik, and K. Trivedi. Stochastic model driven capacity planning for an infrastructure-as-a-service cloud. *IEEE Transactions on Services Computing*, 7(4):667–680, 2014.

[8] R. Ghosh, K. S. Trivedi, V. K. Naik, and D. S. Kim. Performability analysis for Infrastructure-as-a-Service cloud: An interacting stochastic models approach. In *PRDC*, 2010.

[9] B. Haverkort, R. Marie, G. Rubino, and K. S. Trivedi (eds.). *Performability Modeling Tools and Techniques*. Wiley, 2001.

[10] P. Heegard and K. S. Trivedi. Network survivability modeling. *Computer Networks*, 53(8):1215–1234, 2009.

[11] C. Hirel, B. Tuffin, and K. S. Trivedi. SPNP: stochastic petri nets. version 6. In *Lecture Notes in Computer Science*, 2000.

[12] V. G. Kulkarni et al. The completion time of a job on multimode systems. *Adv. Appl. Prob.*, 19(4):932–954, 1987.

[13] J. C. Laprie. From dependability to resilience. In *DSN*, 2008.

[14] Y. Liu and K. S. Trivedi. Survivability quantification: The analytical modeling approach. *Intl. Journal of Performability Eng.*, 2(1):29–44, 2006.

[15] L. Simoncini. Resilient computing: An engineering discipline. In *IPDPS*, 2009.

[16] J. P. Sterbenz et al. Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines. *Elsevier Computer Networks*, 54(8):1245 – 1265, June 2010.

[17] T1A1.2 Working Group on Network Survivability Performance. Enhanced Network Survivability Performance. Technical Report T1.TR.68-2001, February 2001.

[18] K. S. Trivedi. *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. Wiley, 2001.

[19] K. S. Trivedi and R. Sahner. Sharpe at the age of twenty two. *SIGMETRICS Perform. Eval. Rev.*, 36(4):52–57, Mar. 2009.