Analytic Modeling of VM Failure and Repair in Cloud Datacenter

Srichandan Sobhanayak
Department of CSE, NIT Rourkela
Email: srichandan.nitr@gmail.com

Ashok Kumar Turuk
Department of CSE, NIT Rourkela
Email: akturuk@nitrkl.ac.in

Bibhudatta Sahoo
Department of CSE, NIT Rourkela
Email: bdsahu@nitrkl.ac.in

***Abstract*—Cloud Computing provides various types of services to the users, such as IaaS, PaaS, and SaaS. In IaaS cloud service, virtualization is one of the major services which helps the user to request for multiple services with the lowest price. The different resource utilization is caused by various mappings between virtual machines (VMs) and physical machines (PMs). Today for cloud service provider the central issue is how to place multiple VMs demanded by the users into the PMs to achieve workload balance and optimize the resource utilization. The proposed cloud model offers (with different levels) services by designing physical machines into four pools, with diverse provisioning delay and energy usage characteristics. We have broken down to acquire the hypothetical energy usage and performance optimization for VM placement in IaaS cloud data center. We have considered mean response delay and job rejection probability on average performance configuration with VM loss rate as our performance metrics.**

***Keywords*—Cloud, IaaS, Analytical Model, PaaS, SaaS.**

I. Introduction

Background And System Description: Cloud computing is an emerging technology. It has profound impact on the economy and the environment. The cloud datacenter provides services to the users at three levels i.e. Infrastructure as a Service (IaaS), Software as a Service (SaaS), and Platform as a Service (PaaS). IaaS cloud provides users with computational resources in the form of virtual machines (VM) instances deployed in the provider data center. In PaaS and SaaS cloud offer services in terms of specific solution stacks and application suites respectively. Inefficient resource utilization within the cloud datacenter significantly escalates power consumption by cloud datacenter equipment while increasing operational cost and carbon emission rate [1]. The largeness and scalability problems of complex models are described in [2]. To solve such problems, the hierarchical composition is introduced in [3] (and many other papers and books), where a two-level hierarchical model is proposed. A Markov chain models each subsystem and the system reliability is modeled after a series system of independent Markov components. In [3], stochastic petrinets (SPNs) and reliability block diagrams are used for the quantification of sustainability impact, cost and dependability of datacenter cooling infrastructures. However, the paper only focuses on the cooling system and does not face scalability issues for availability evaluation. There are lots of works exist [4], [5], which uses queuing models to study the performance of distributed system with various performance metrics such as weighted mean response time and arithmetic average response time [6], a novel stochastic framework for energy efficiency and performance analysis of dynamic voltage and frequency scaling (DVS) enabled cloud [7], probability of load balancing success [8], mean response ratio [9], and mean miss rate [10]. Optimal load distribution in a heterogeneous distributed computer system with both generic and dedicated applications was studied in [11], [12]. To the best of our knowledge, there has been no similar investigation in the literature. The method of optimal multicore server processor configuration has been employed for other purposes, such as managing the power and performance tradeoff [13]. In an IaaS cloud, when a request is processed, a prebuilt image is utilized to convey one VM, or a predeployed VM may be tweaked and made accessible to the clients. VMs are conveyed on PMs that may be shared by different VMs. The conveyed VMs are provisioned on the premise of a prerequisite of CPU, RAM and system bandwidth. This procedure of provisioning furthermore, conveying VMs includes delays that may be diminished by different advancement methods. One such approach is to gathering the PMs into numerous pools, portrayed by distinctive degrees of delay included in provisioning of VMs. Depending on delay involved in provisioning, in our proposed work, we have grouped PMs into four pools: ***a-type*** (running), ***b-type*** (turned on but not ready), ***c-type*** (sleep mode) and ***d-type*** (turned off). A pre-instantiated VM can be promptly provisioned and conveyed to prepared state on a running PM (***a-type***) with least provisioning delay. Instantiating a VM from an image and conveying it on a ***b-type*** PM needs extra time in provisioning. PMs in the ***c-type*** and ***d-type*** pools are sleep and turned of when not being used, and conveying a VM on such PMs experiences an extra start up delay. We have assumed that 1) the measure of a wide range of pool is foreordained, 2) pool has homogeneous PMs, 3) all requests are homogeneous, where every task may ask for one VM with altered size CPU cores, RAM and system network bandwidth requirement.

Problem statement : Taking into account the above discussion, We have evaluated complete performance VM provisioning with Virtual Machine Placement Submodel (VMPM) of the IaaS cloud. This proposed work handles the issue of analytical modeling and analysis of IaaS cloud datacenter. We consider expected VM completion time, VM loss rate, job rejection probability and mean response delay as key metrics of performance. We employ a continuous-time Markov model to obtain analytical solutions of these metrics.

Key contributions : Based on the scalable interactingstochastic models approach, as described in [13] we make the following contributions in this paper: (1) using Markov reward approach we consider VM loss rate, in each pool of IaaS cloud datacenter and (2) VM completion time for the proposed model by considering two QoS parameter i.e. job rejection probability,

mean response delay through careful exploration of different cloud parameters and configurations.

Rest of the paper is organized as follows, Section II, presents the problem statement and model description, Section III describes the simulation result, we conclude and outline future research in Section IV.

II. Model Description

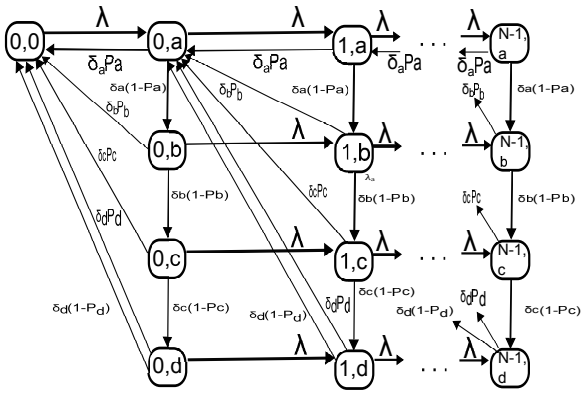


Fig. 1: SCM Model [13]

A. Server Consolidation Submodel (SCM)

SCM is modeled as a CTMC as shown in Fig. 1. A finite length decision queue is considered where decisions are made on a FCFS basis. States in this model (***a-type***, ***b-type***, ***c-type*** or ***d-type***) are labeled as (**i, s**), where i denotes the number of jobs and s denotes pool type se lected for job provisioning. When the queue is empty SCM is in state of (**0, 0**). We consider the following pa rameter as an input to the CTMC: (i) job arrival rate (λ), (ii) mean searching delay to find a PM in **extita-type**, ***b-type***, ***c-type***, and ***d-type*** pool for resource pr ovisioning as $1/\delta_a$, $1/\delta_b$, $1/\delta_c$ or $1/\delta_d$ respectively, (iii) probabilities that a extbf*a-type*, ***b-type***, ***c-type***, and ***d-type*** PM can accept a jo b for resource provisioning be P_a, P_b, P_c and P_d respectively, and (iv) size of the job queue is N. Th e different parameters mentioned this model is as, N is assumed to be given, the arrival rate λ, δ_a, ẹlta_b, δ_c, and δ_d supposed to be measured, VMPM provides probabilities P_a, P_b, P_c and P_d. The state transition diagram can be described as: with arrival rate λ the state (**0, 0**) transit to state (0,**a**. With probability P_a, the state (0, **a**) transit to state (**0, 0**), confirming that job is accepted for provisioning on one of the ***a-type*** PMs. With probability (1-P_a) , the state (0, **a**) transit to state (0, **b**), affirming that job can't be acknowledged for provisioning on any ***a-type*** PM, and the state (0, **a**) transit to state (1, **a**) when a new job arrives with rate λ, this speaks to the condition that one job is holding up in the queue and one job is experiencing provisioning decision. SCM tries to search for ***b-type*** PM in state (0, **b**) to provision the job that could not be handled by ***b-type*** PM. When no ***b-type*** PM is available (i.e. probability (1-P_b)) the state (0, **b**) transit to state (0, **c**). in state (0, **c**), SCM search for one ***c-type*** PM to provision the job (that could not be provisioned on any ***b-type*** PM) . When no ***c-type*** PM is available then the state transit from (0, **c**) to state (0, **d**) with probability (1-P_c). The job experiencing provisioning decision goes out of the queue, once a decision has been made for the ***d-type*** pool. With probability P_d, the job can be acknowledged in the ***d-type*** pool and sub-model goes to state(0, 0). With probability (1-P_d), all ***d-type*** PMs are occupied, job is rejected and the sub-model goes to state (0, 0). From this model, we can process job rejection probability because of buffer full (P_{block}), rejection probability because of deficient capacity (P_{drop}) and consequently, job rejection probability P_{reject}= P_{block}+P_{drop}. We can likewise process mean queuing delay ($E(q_{delay})$) and mean decision delay ($E[D_{delay}]$) conditional upon the job being accepted.

B. VM provisioning sub-models

VM provisioning sub-models catch the instantiation, setup, and provisioning of a VM on a PM. For every PM in diverse pool, we have one CTMC which stays informed concerning the quantity of relegated and running VMs. VM provisioning sub-model of a pool is the union of individual provisioning sub-models of every PM in that pool. A running PM in a pool may fail. The time to-failure follows an exponential distribution with failure rate $F_a \in R^+$ in ***a-type*** pool(F_b, F_c and F_d for ***b-type***, ***c-type*** and ***d-type*** pool), where R^+ is the set of positive real numbers. Every fizzled PM can be repaired by programming or equipment repair units and restored to the typical state. The repair time follows an exponential distribution with repair rate $Z_a \in R^+$ (Z_b, Z_c, Z_d for ***b-type***, ***c-type*** and ***d-type*** pool). VM instances on these failed PMs also fail. In this case, they go back and join the arrival flow for resubmission. The time to resubmit one failed VM instance is exponentially distributed with rate R_a in ***a-type*** pool(R_b, R_c and R_d for ***b-type***, ***c-type*** and ***d-type*** pool).

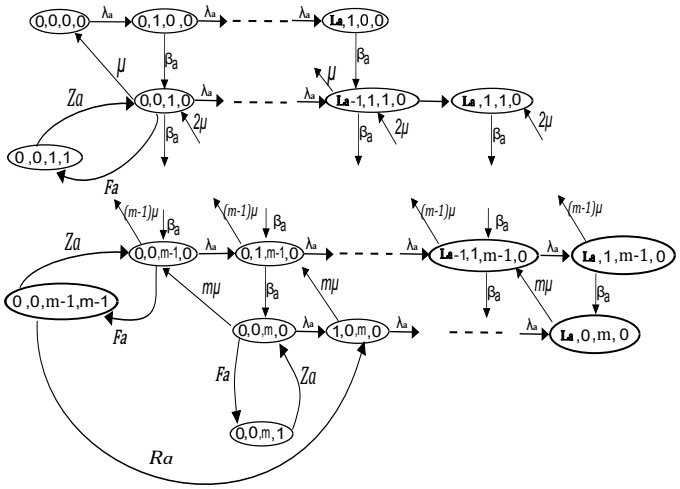


Fig. 2: *a-type* VMPM

1) ***a-type*** *PM Pool:* Our ***a-type*** pool is a two stage tandem network. We derive the closed form expressions for the steady state probabilities of the ***a-type*** PM CTMC when $L_a \leftarrow \infty$ and $m \rightarrow \infty$. We model the ***a-type*** pool PM, similar to the detail given in [13] can be referred. While the proposed model assumes homogeneous requests (one VMs for each request). We design separate VM provisioning model for four different pool of PMs. Fig.2 shows a VM provisioning sub-model for a PM in the ***a-type*** pool. Conceptually, the overall ***a-type*** pool

is modeled by a set of independent ***a-type*** PM. Although, note that only one PM pool needs to be solved. For the sake of simplicity in representation Fig.2 shows the VMPM when each task has only one VM. A PM in ***b-type***, ***c-type*** and ***d-type*** pool modeled as ***a-type*** though, with some minor differences. Therefore, a task can be accepted by a PM if first, there is enough room in the PMs queue for all VMs within the task and second, if the PM has sufficient resources (CPU, RAM and Network bandwidth) for the given task. States of the sub-model in ***a-type*** are indexed by *(i, j, k, l)*, where, i denotes number of jobs in the queue, j denotes number of jobs currently being provisioned, k denotes the number of VMs which have already been deployed and *l* gives the details of status of PM i.e. a PM is working or failed. Input parameters for the ***a-type*** CTMC are: (i) effective job arrival rate to each ***a-type*** PM (λ_a), (ii) rate at which VM instantiation, provisioning, and configuration occurs (β_a), and (iii) μ be the service rate of each VM. We assume that the jobs are coming one at a time and hence, the value of j is either 0 or 1 and *l* is the either 0 indicating working condition and 1 indication failed condition. Assuming a total of n_a PMs in the ***a-type*** pool.

Among other input parameters β_a can be measured, μ_a can be computed from the run-time sub-model , L_a and m are assumed to be given. Here we describe the state changes in pool given in Fig. 2. The sub-model goes to state (0, 1, 0, 0), from state (0, 0, 0, 0), with job arrival rate λ_a. In state (0, 1, 0, 0), the job is being supplied with VM instance. Mean time to provision a VM on a ***a-type*** PM is $1/\beta_a$ and the state changes from (0, 1, 0, 0) to state (0, 0, 1, 0) with rate β_a. The VM instance is removed and the sub-model moves from state (0, 0, 1, 0) to state (0, 0, 0, 0) with rate μ when a job is completed. Arrival of a new job with rate λ_a will take the sub-model to state (1, 1, 0, 0) from state (0, 1, 0, 0) When a VM is being provisioned in state (0, 1, 0, 0), where the job is waiting in the queue. In almost the same way, new jobs can arrive more and more moving the sub-model to states (i, 1, 0, 0) with $0 \leq i \leq L_a$. In state (L_a, 1, 0, 0) the buffer is full and hence, no new job can arrive. When a VM is in execution (e.g., in state (0, 0, 1, 0)), arrival of a new job with rate λ_a will take the sub-model to state (0, 1, 1, 0), where the new job is being provisioned. Suppose a running PM fails in the state (0, 0, 1, 0) then the state become (0, 0, 1, 1), i.e. *l* value will be 1 where, 1 is failure condition and 0 is for running condition. VM instances on a failed machine are resubmitted and the state goes from (0, 0, 1, 1) to (0, 1, 0, 0). If the buffer is full then the corresponding job of failed VM will be passed to next pool, if none of the pool have sufficient space to host this job then the VM is rejected and that falls under rejection of job due to insufficient space. The output from this pool is B_a blocking probability, and P_a that at least one PM in the ***a-type*** can accept a job for provisioning. Now, the probability of failure of a machine can be given by:

$$P_{fail(a-type)}(xa) = \frac{\sum_{ya=0, z(ya)=z(xa)+1, \phi^a_{xa,ya}>0}^{L_a} \phi^a(xa, ya)}{\sum_{ya=0, x\neq ya}^{L_a} \phi^a(xa, ya)} \tag{1}$$

where $z(xa)$ is the number of failed VM at state $\phi^a(xa)$, z(ya)=z(xa)+1 indicates that one more failure happens in the transition from state $\phi^a(xa)$ to $\phi^a(ya)$. $\phi^a(xa, ya) > 0$ means that $\phi^a(ya)$ is an immediate succeeding state of $\phi^a(xa)$. The probability of VM failure, $D_{a-type,(xa)}$, can be calculated as the ratio of the aggregate probability of $P_{fail(a-type}(xa)$, to the aggregate probability of states with at least one alive PM i.e.

$$D_{a-type,(xa)} = \frac{\sum_{xa=0, g(xa)>0}^{L_a} \phi^a(xa) \times P_{fail(a-type)}(xa)}{P_a} \tag{2}$$

where $g(xa)$ denotes the number of alive VM instances $\phi^a(xa)$. From the system equilibrium view point, the actual input rate satisfies the following equation:

$$n_a(1 + D_{a-type,(x)})\lambda_a = \lambda(1 - P_a) \tag{3}$$

Finally, the expression for λ_a can be given as follows:

$$\lambda_a = \frac{\lambda(1 - P_{block})}{n_a(1 + D_{a-type,(x)})} \tag{4}$$

Where, ρ is the service rate. As discussed earlier, failed VM tasks go back to the waiting queue for resubmission. In this case, the completion time for these failed ones are longer than the sojourn time of successful ones. Letting ua denote the completion time for a failed VM instance, expected completion time of failed VM is

$$E(ua) = E(\omega a) + \frac{1}{F_a} + E(sa) \tag{5}$$

where $1/F_a$ is the expected failure time and $E(sa)$ is the expected VM completion time. $E(\omega a)$ is expected waiting time of VMs, Equation 5 implies that $E(ua)$ can be calculated as the expected sojourn time of a failed VM instance plus the expected completion time as a new resubmitted VM task.

VM loss rate, la, is another important metric. Based on our stochastic model, VM tasks can be rejected and lost due to a full queue. Note that resubmitted VMs can also be rejected when they reenter the queue. Therefore

$$la = B_a + P_a \times D_{a-type,(xa)} \times la$$

where $D_{a-type,(xa)} \times la$ means the probability of VM loss when failed ones are resubmitted.

2) ***b-type*** *PM Pool:* CTMC for a ***b-type*** PM is shown in Fig. 3, which can be described as when no VM is running or being provisioned, ***b-type*** PM is turned on but not ready for use. Upon a job arrival, the ***b-type*** PM requires some additional start up delay to make it ready to use. So, the sub-model goes from state (0, 0, 0, 0) to state (0, 1, 0, 0). Time to make a ***b-type*** PM ready for use, is assumed to be exponentially distributed with mean $1/\gamma_b$. (iii) Mean time to provision a VM on a ***b-type*** PM is $1/\beta_b$ for the first VM to be deployed on this PM; mean time to provision VMs for subsequent jobs is the same as that for a ***a-type*** PM, i.e., $1/\beta_a$. When a running job exits from state (0, 1, 1, 0), the pool moves to the state (0, 1^{**}, 0, 0) (instead of state (0, 1^{*}, 0, 0)). In state (0, 1^{**}, 0, 0), the ***b-type*** PM is ready to use (behaving like a ***a-type*** PM) and hence mean time to provision a VM in this state is $\frac{1}{\beta_a}$. We

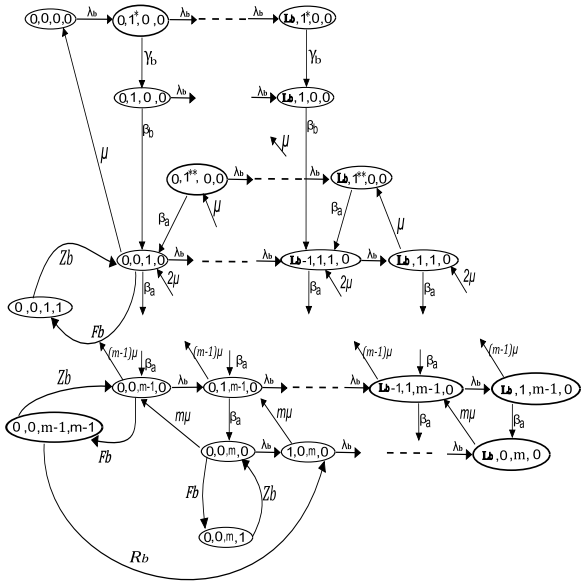


Fig. 3: *b-type* VMPM

assume that the length of buffer on each ***b-type*** PM is L_b. The ***c-type***, ***d-type*** pool can be described in similar manner. The output from this pool is B_b blocking probability, and P_b that at least one PM in the ***b-type*** can accept a job for provisioning. The other parameter for ***b-type***, ***c-type***, ***d-type*** pool is similar to ***a-type*** type pool, and is not mentioned in this paper due to brevity.

3) Overall outputs for VM provisioning sub-models: As discussed earlier, failed VM tasks go back to the waiting queue for resubmission. In this case, the completion time for these failed ones are longer than the sojourn time of successful ones. Letting u denote the completion time for a failed VM instance, we have $E[u]$ be the expected completion time of all failed VMs. Thus, the mean response delay is then given by:

$$\begin{aligned} E\left[T_{resp}\right] =& E(q_{delay}) + E\left[D_{delay}\right] \\ &+ E\left[T_{q\_vm}\right] + E\left[T_{prov}\right] + E[u] \end{aligned} \tag{6}$$

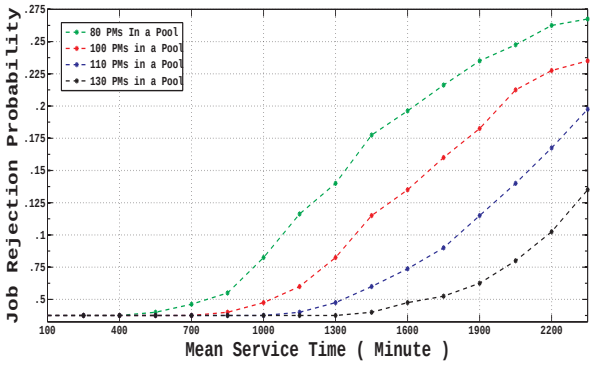
where $E\left[T_{prov}\right]$ is for a job, conditional upon being accepted, $E\left[T_{prov}\right]$ and $E\left[T_{q\_vm}\right]$ is mean queuing delay for VM provisioning. The details are not mentioned due to brevity.

III. Experimental Results

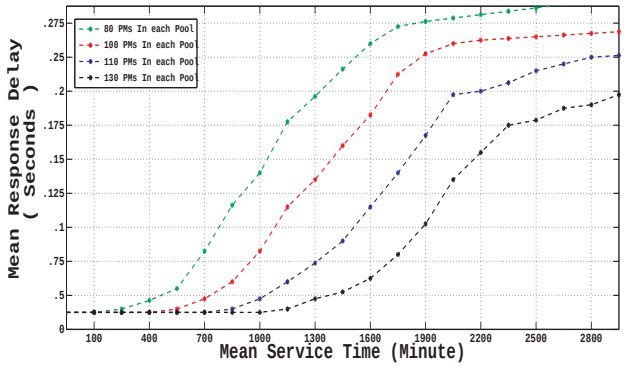
We consider a datacenter sample built on Intel Xscale PXA270 uniprocessors. We use MATLAB 13.0 to implement the stochastic model for the system. We have used the algorithm from [13] where the input data to this algorithm are as follows ***a-type***, ***b-type***, ***c-type***, ***d- type*** and SCM Sub model and the outcome is steady state values of performance measures and steady state values of fixed-point variables (P_{Block}, P_a, P_b, P_c and P_d). Increase in mean service time increases job rejection probability increases with as depicted in Fig. 4a, with the job arrival rate (1500 jobs/h) and fixed number of PMs in each pool (e.g., 100 PMs in each pool). The job rejection probability can be minimized by increasing the PM capacity in each pool at a constant mean service time. Mean response delay increases if we increase mean service time for at a constant PMs in each pool as depicted in Fig. 4b. In our proposal the gain can be defined as the minimization of job rejection probability or mean response delay with increase in number of PMs in each pool keeping other input parameter constant. When the pm capacity increases from 80 to 130 PMs in each pool then the marginal gain change with increasing mean service time which is shown in Fig. 4b.

We have researched in our illustration that in the medium extent mean service time (around 100-2800 minute) the gain is greatest in light of the fact that, for a settled number of PMs, mean response delay has three sections those are, mean queuing delay before SCM, mean decision delay and conditional mean provisioning delay. The marginal gain fluctuates with expanding mean service time contingent on which component of delay is more prevailing. The gain due to increase in PM is inconsequential when the mean service time is low (100 - 400 moment) as to keep low response delay, jobs needs to leave the server farm for the new job to arrive. The queue before the SCM increments when the mean service time of jobs increases (say 1000 min), for low capacity frameworks (e.g., 80 PMs in every pool for our illustration). This shows that adding more PMs decrease mean queuing delay in front of SCM. The marginal gain decreases when the mean service time increases to 1900 for larger capacity systems (e.g., 130 PMs in each pool for our example) increasing mean queuing delay in front of SCM. This outcome indicates in general mean response delay, how the parameter mean queuing delay changes as the mean service time and PMs in every pool are changed.

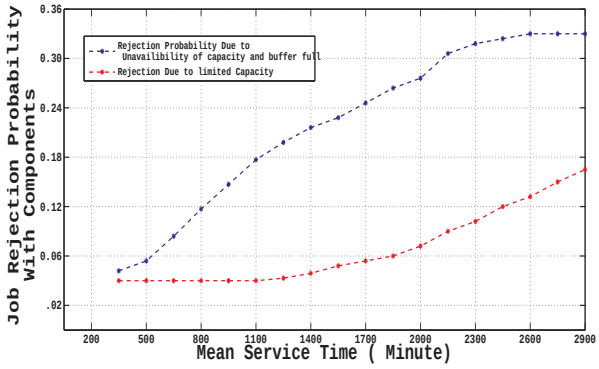
It is intriguing to take a gander at the parts of mean response delay and rejection probability. They are indicated in Fig. 4d for 80 PMs in every pool. For the illustration indicated in Fig. 4c, beyond mean service time of 100 minute, rejection because of buffer full is prevailing contrasted with rejection because of deficient capacity. This is a case that shows how our models can give an ides to imagine a scenario where investigation to better comprehend framework bottlenecks. In the sample demonstrated in Fig 4d, mean provisioning delay is predominant when mean service time is under 150 minute and mean queueing delay is overwhelming when mean service time is higher than 150 minute. We show the effects of changing arrival rate, task service time, number of PMs in each pool, number of VMs on each PM and job size on the interested performance indicators. Arrival rate ranges from 50 to 1500 jobs per hour. Mean service time of each task within service time ranges from 40 to 220 minutes. We assume 90 to 130 PMs in pools. The results in Figs. 4e shows that jobs arrives with at a particular rate and there occurs VM loss or machine failure and the failed machine need to be resubmitted to the PMs to increase the rate of success in VM execution. It also reveals that with the increase in task arrival the VM failure increases which affect increase in service time will result in longer total delay on job completion. In Fig 4f, we examine the effect of job size on task rejection probability using of geometric. Here we have kept a constant arrival rate of 800 jobs per hour. As can be seen by increasing the size of jobs the rejection probability reduces sharply for both geometric and uniform distribution. Since the size of job is truncated to the maximum number of VMs allowed on each PM (here, up to 6 VMs), the bigger job size will result in the lower number of arrivals as well as lower deviation of jobs.



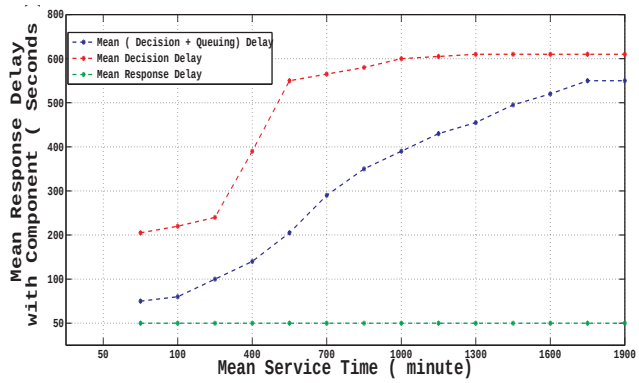
(a) Mean response delay vs. mean job service time



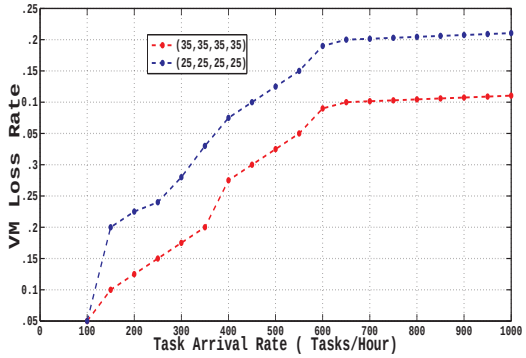
(b) Job rejection probability vs. mean job service time



(c) Job rejection probability with components vs. mean service time



(d) Mean response delay with component vs. mean service time



(e) Total delay vs. mean service time on job arrival rate 1000 task per hour

(f) Rejection probability vs. with arrival rate 1200 jobs per hour with VM request

Fig. 4: Experimental Results

IV. Conclusion

In this paper, we present a performance model suitable for IaaS cloud datacenter environment with homogeneous requests and resources using interacting stochastic models. We measure the impacts of varieties in workload (e.g., job arrival rate, job service rate, failure and repairing of VMs) and framework limit (PMs per pool, VMs per PM) on the execution of a class of IaaS clouds where PMs are sorted out in a set of pools for administration and power consumption costs reduction. Moreover, under different configurations, the behavior of IaaS cloud datacenter is characterized so that an effective admission control can be achieved. Results show that our approach enables the cloud service providers to detect system bottlenecks. We plan to extend the heterogeneity in jobs (i.e., RAM and disk) and server pools (i.e. different PMs in each pool) and energy management in a cloud datacenter in our future work.

References

[1] S. Marston, Z. Li, S. Bandyopadhyay, J. Zhang, and A. Ghalsasi, "Cloud computingthe business perspective," *Decision Support Systems*, vol. 51,

no. 1, pp. 176–189, 2011.

[2] R. A. Sahner, K. Trivedi, and A. Puliafito, *Performance and reliability analysis of computer systems: an example-based approach using the SHARPE software package*. Springer Publishing Company, Incorporated, 2012.

[3] J. T. Blake and K. S. Trivedi, "Reliability analysis of interconnection networks using hierarchical composition," *Reliability, IEEE Transactions on*, vol. 38, no. 1, pp. 111–120, 1989.

[4] H. Kameda, J. Li, C. Kim, and Y. Zhang, *Optimal load balancing in distributed computer systems*. Springer Publishing Company, Incorporated, 2011.

[5] A. N. Tantawi and D. Towsley, "Optimal static load balancing in distributed computer systems," *Journal of the ACM (JACM)*, vol. 32, no. 2, pp. 445–465, 1985.

[6] K. Li, "Minimizing mean response time in heterogeneous multiple computer systems with a central stochastic job dispatcher," *International journal of computers & applications*, vol. 20, no. 1, pp. 32–39, 1998.

[7] Y. Xia, M. Zhou, X. Luo, S. Pang, and Q. Zhu, "A stochastic approach to analysis of energy-aware dvs-enabled cloud datacenters," *IEEE Transaction on SYSTEM,MAN, CYBERNATICS: SYSTEMS*, vol. 45, no. 1, 2015.

[8] C. Rommen, "The probability of load balancing success in a homogeneous network," *Software Engineering, IEEE Transactions on*, vol. 17, no. 9, pp. 922–933, 1991.

[9] X. Tang and S. T. Chanson, "Optimizing static job scheduling in a network of heterogeneous computers," in *Parallel Processing, 2000. Proceedings. 2000 International Conference on*. IEEE, 2000, pp. 373–382.

[10] L. He, S. A. Jarvis, D. P. Spooner, H. Jiang, D. N. Dillenberger, and G. R. Nudd, "Allocating non-real-time and soft real-time jobs in multiclusters," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 17, no. 2, pp. 99–112, 2006.

[11] F. Bonomi and A. Kumar, "Adaptive optimal load balancing in a nonhomogeneous multiserver system with a central job scheduler," *Computers, IEEE Transactions on*, vol. 39, no. 10, pp. 1232–1250, 1990.

[12] K. W. Ross and D. D. Yao, "Optimal load balancing and scheduling in a distributed computer system," *Journal of the ACM (JACM)*, vol. 38, no. 3, pp. 676–689, 1991.

[13] R. Ghosh, F. Longo, V. K. Naik, and K. S. Trivedi, "Modeling and performance analysis of large scale iaas clouds," *Future Generation Computer Systems*, vol. 29, no. 5, pp. 1216–1234, 2013.